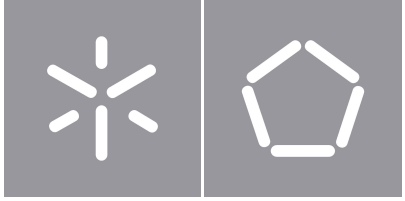


University of Minho
School of Engineering

Diogo Gomes Matos

The Evolution of Intelligence Efficiency in Local, Open Large Language Models: A Controlled Empirical Study

October 2026



University of Minho
School of Engineering

Diogo Gomes Matos

The Evolution of Intelligence Efficiency in Local, Open Large Language Models: A Controlled Empirical Study

Master's Dissertation in Informatics Engineering

Dissertation supervised by
João Alexandre Batista Vieira Saraiva
Roberto Verdecchia

October 2026

Copyright and Terms of Use for Third Party Work

This dissertation reports on academic work that can be used by third parties as long as the internationally accepted standards and good practices are respected concerning copyright and related rights.

This work can thereafter be used under the terms established in the license below.

Readers needing authorization conditions not provided for in the indicated licensing should contact the author through the RepositóriUM of the University of Minho.

License granted to users of this work:



CC BY-NC-SA

https://creativecommons.org/licenses/by-nc-sa/4.0/

Acknowledgements

First and foremost, I would like to express my gratitude to the Erasmus+ mobility programme for providing the opportunity to conduct a portion of this research abroad in Italy. I am eternally grateful to the University of Florence for hosting me, and to my co-supervisor Roberto Verdecchia for his invaluable guidance and support during my time there. I am also particularly thankful to Dr. Imad Zaza for his crucial technical expertise and patience, without which this work would not have been possible, and to Dr. Ivano Malavolta, from the University of Amsterdam, for his thoughtful input and willingness to engage with my work. I could not have asked for a more welcoming and supportive environment, thank you all for making my time in Florence so memorable and enriching.

I am equally grateful to my supervisor João Saraiva for all his guidance, feedback and advice during the entirety of this journey. From his instrumental role in connecting me with Roberto to his unforgettable visit to Florence, Professor Saraiva's genuine commitment and mentorship made all the difference and I am forever thankful for that.

Este estudo marca o fim da jornada inesquecível que foram os últimos cinco anos da minha vida. Um período por vezes desafiador, mas sobretudo marcado por crescimento e descoberta, de onde levo memórias e amizades que me acompanharão para sempre. Se hoje consigo olhar para trás com um enorme sorriso no rosto, é graças a todos aqueles que me apoiaram, acompanharam e fizeram parte desta aventura comigo, e aos quais gostaria de deixar o meu mais sincero obrigado.

Aos meus pais, Rosa e Luís, obrigado pelo vosso amor e apoio incondicional. Obrigado por me terem sempre incentivado a seguir o meu próprio caminho e por nunca duvidarem de mim. Obrigado por todo o vosso esforço para suportar os meus sonhos, sei que por vezes não é fácil. Independentemente das circunstâncias ou da distância, sei que posso contar convosco e que vou ter sempre em vocês um lugar ao qual chamar casa.

Ao meu irmão, Tiago, obrigado por fazeres parte da minha vida. Tem sido um prazer ver-te crescer e lutar pelos teus sonhos, mesmo que por vezes seja à distância. Tenho imenso orgulho em ti e espero poder celebrar novas etapas contigo no futuro.

Aos amigos mais especiais que a universidade me trouxe: Inês Ferreira, Joana Branco, Inês Castro e Adriana Frazão, obrigado por todo o vosso apoio e por terem sido uma parte tão bonita do meu percurso. Inês e Joana, nunca me esquecerei de todas as memórias felizes que partilhámos, obrigado do fundo do coração por terem estado sempre lá para mim, nos bons e nos maus momentos, e por terem sido uma parte tão especial da minha vida nos últimos anos. Inês (fã da Katya), não há nenhuma conversa contigo com a qual não nos consigamos rir desalmadamente. Obrigado por seres quem és e por teres tornado tão especial o tempo em que vivemos juntos. Adriana, conhecer-te foi das coisas mais bonitas que me podia ter acontecido. Obrigado por seres a pessoa tão genuína que és, tenho imenso orgulho em ti.

Ao Pedro Mota, obrigado por seres a pessoa incrível que és e por me mostrares o quão bonito é conhecer e gostar profundamente de alguém. Obrigado por seres parte de mim e dos momentos mais felizes da minha vida.

To all the amazing friendships I had the privilege to make during my Erasmus experience: Nikol Mlechenkova, Sara Correia, Rosa Hodde, Tiago Duarte, Luísa Gaspar, Naïs, Mafalda Costa, Munise Guclu, Maria Madalina, Dionysia Bourli, Rachita Cheetah and Aymeric, thank you for making my 5 months abroad one of the most beautiful and unforgettable experiences of my life. I will forever hold close all the memories and happiness we shared together, and will never forget how your company made me feel at home in a foreign country. You made working on this dissertation a lot more fun and enjoyable, and I will always be grateful for that.

Obrigado a todos por fazerem parte da minha vida e por me terem ajudado a crescer, a aprender e a tornar-me na pessoa que sou hoje.

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethics and Conduct of the University of Minho.

University of Minho, Braga, October 2026

Diogo Gomes Matos

Declaration on the Use of Artificial Intelligence (AI) Tools

I hereby declare having acted this academic work with integrity and confirm that I have not resorted to any form of undue use of Artificial Intelligence (AI) tools, nor to the falsification or omission of information, sources, data, results or processes in any of the stages leading to its elaboration.

I further declare that any use of AI tools was done in a transparent and responsible manner, being duly noted in the work, and that I critically verified the accuracy of the content produced with their support, assuming full responsibility for what I submit.

I further declare that I have fully acknowledged the Code of Ethics and Conduct of the University of Minho, as well as the applicable standards of academic integrity and data protection.

University of Minho, Braga, October 2026

Diogo Gomes Matos

Resumo

The Evolution of Intelligence Efficiency in Local, Open Large Language Models: A Controlled Empirical Study

Os grandes modelos de linguagem (LLMs, do inglês *Large Language Models*) são usados por centenas de milhões de pessoas e lançados a um ritmo acelerado, o que torna o seu consumo de energia uma preocupação crescente de sustentabilidade. Ao contrário dos modelos fechados, os modelos abertos podem ser executados localmente e medidos em condições idênticas, mas os estudos existentes comparam os modelos disponíveis num único momento, alteram o *hardware* juntamente com os modelos, ou não consideram a exatidão. Continua, por isso, por esclarecer se cada lançamento é energeticamente mais eficiente do que aquele que substitui, ou se a eficiência está a ser sacrificada em favor do desempenho.

Esta dissertação mede a evolução da eficiência de inteligência dos LLMs abertos, isto é, da exatidão que oferecem em relação à energia que consomem, ao longo de lançamentos sucessivos. São avaliados 47 modelos de 20 lançamentos de seis famílias, publicados entre julho de 2023 e março de 2026, numa única GPU, com o mesmo motor de inferência e o mesmo *benchmark*, através de um *pipeline* automatizado, reprodutível e público.

Os modelos lançados tornaram-se mais exatos, em cerca de 20 pontos percentuais por ano, mas a sua eficiência de inteligência não melhorou: as respostas alongaram-se, enquanto a energia de cada *token* se manteve determinada pelo número de parâmetros ativos. Ainda assim, um orçamento de energia fixo passou a permitir uma exatidão mais elevada, pelo que os ganhos de eficiência foram reinvestidos em exatidão. Os modelos com raciocínio (do inglês *reasoning models*) consumiram entre 2,6 e 19,6 vezes a energia por pergunta das respetivas versões *instruct*, para um ganho modesto de exatidão. Entre lançamentos do mesmo tamanho, a eficiência de inteligência só melhora, assim, se a exatidão crescer mais depressa do que o comprimento das respostas. Foi nesse comprimento, que uma classificação pela exatidão não revela, que os ganhos de eficiência foram absorvidos, e é nele que podem ser conservados.

Palavras-chave Consumo de Energia, Eficiência de Inteligência, Grandes Modelos de Linguagem, Green AI, Green Software

Abstract

The Evolution of Intelligence Efficiency in Local, Open Large Language Models: A Controlled Empirical Study

Large Language Models (LLMs) are used by hundreds of millions of people and released at a rapid pace, which makes their energy consumption a growing sustainability concern. Open models, unlike closed ones, can be executed locally and measured under identical conditions, yet existing studies compare the models available at a single moment, change the hardware together with the models, or leave accuracy out. It therefore remains unclear whether each release is more energy-efficient than the one it replaces, or whether efficiency is being sacrificed for raw performance.

This dissertation measures how the intelligence efficiency of open LLMs, that is, the accuracy they deliver relative to the energy they consume, evolved across successive releases. It evaluates 47 models from 20 releases of six families, published between July 2023 and March 2026, on a single GPU, with the same inference engine and benchmark, through an automated, reproducible and public pipeline.

The released models became more accurate, by approximately 20 percentage points per year, but their intelligence efficiency did not improve: their responses grew longer, while the energy of each token remained determined by the number of active parameters. A fixed energy budget nonetheless came to yield a higher accuracy, so that the gains in efficiency were reinvested in accuracy. Reasoning models consumed 2.6 to 19.6 times the energy per question of their instruct counterparts for a modest gain in accuracy. Between releases of the same size, intelligence efficiency thus improves only if accuracy grows faster than the length of the responses. This length, which a ranking by accuracy does not reveal, is where the gains in efficiency were absorbed, and therefore where they can be retained.

Keywords Energy Consumption, Green AI, Green Software, Intelligence Efficiency, Large Language Models

Contents

1 Introduction **1**

1.1 Context . . . 1

1.2 Motivation . . . 2

1.3 Aims and Objectives . . . 3

1.3.1 Research Questions . . . 4

1.4 Contributions . . . 5

1.5 Related Publications . . . 6

1.6 Document Structure . . . 7

2 State of the Art **8**

2.1 Green Software and Sustainability in AI . . . 8

2.1.1 Green AI . . . 10

2.2 Large Language Models: Evolution and Efficiency . . . 12

2.2.1 From Prompt to Response . . . 15

2.2.2 Instruction-tuned and Reasoning Models . . . 16

2.3 The Landscape of Energy Measurement Tools . . . 17

2.3.1 Intel RAPL . . . 20

2.3.2 NVIDIA SMI and NVML . . . 21

2.3.3 AMD SMI . . . 23

2.3.4 Scaphandre . . . 23

2.3.5 EnergiBridge . . . 24

2.3.6 CodeCarbon . . . 25

2.3.7 CarbonTracker . . . 26

2.3.8 EnviroLLM . . . 26

2.4 Designing Reliable Energy Experiments . . . 27

2.5 Local Execution of LLMs . . . 29

2.5.1 Obtaining Quantized Models . . . 30
2.5.2 Dense vs. Mixture-of-Experts Architectures . . . 30
2.5.3 Llama.cpp . . . 32
2.5.4 Ollama . . . 33
2.5.5 vLLM . . . 34
2.5.6 TensorRT-LLM . . . 35
2.6 Evaluating the Performance of LLMs . . . 35
2.6.1 Humanity's Last Exam (HLE) . . . 37
2.6.2 GPQA Diamond . . . 39
2.6.3 MMLU . . . 40
2.6.4 MMLU-Pro . . . 40
2.6.5 MATH . . . 42
2.6.6 AIME 2025 . . . 43
2.6.7 HumanEval . . . 43
2.6.8 MBPP . . . 44
2.6.9 LiveCodeBench . . . 45
2.6.10 LiveCodeBench Pro . . . 47
2.6.11 SWE-bench Verified . . . 49
2.6.12 FACTS Benchmark Suite . . . 50
2.6.13 Global PIQA . . . 51
2.6.14 SuperGLUE . . . 51
2.6.15 Summary . . . 52
2.7 Related Work . . . 53
2.7.1 Measuring the Energy of Inference . . . 53
2.7.2 Energy and Accuracy . . . 54
2.7.3 Efficiency Over Time . . . 55
2.7.4 Research Gap . . . 57

3 The Problem and Its Challenges 60

3.1 Model Characteristics . . . 60
3.2 Inference Settings . . . 63
3.3 Benchmark Limitations . . . 65
3.4 Hardware Constraints . . . 66

3.5 Inference Engine Selection . . . 66
3.6 Energy Measurement . . . 68

4 Methodology **70**

4.1 Experimental Environment . . . 70
4.1.1 Hardware Configuration . . . 70
4.1.2 Inference Engine . . . 71
4.1.3 Energy Measurement Tools . . . 72
4.1.4 Orchestration and Software Versions . . . 73
4.2 Model Selection . . . 74
4.2.1 Families and Releases . . . 74
4.2.2 Filtering Rules . . . 75
4.2.3 Model Metadata . . . 78
4.2.4 Prioritization of Instruction-tuned Variants . . . 79
4.2.5 Size Categories . . . 80
4.2.6 Final Selection . . . 81
4.2.7 Composition of the Final Set . . . 82
4.3 Experimental Design . . . 84
4.3.1 Model Runs and Benchmark . . . 84
4.3.2 Repetitions and Randomization . . . 84
4.4 Execution of a Run . . . 85
4.4.1 Model Loading and Inference Configuration . . . 87
4.4.2 Benchmark Sampling . . . 89
4.4.3 Prompt Construction . . . 90
4.4.4 Warmup and Thermal Control . . . 93
4.4.5 Inference and Data Collection . . . 94
4.5 Answers and Efficiency Metrics . . . 96
4.5.1 Answer Extraction . . . 97
4.5.2 Summary of a Run . . . 98
4.5.3 Efficiency Metrics . . . 98
4.6 Data Analysis . . . 100
4.6.1 Values of a Model Run . . . 100
4.6.2 Trends over Time . . . 101

4.6.3 Accuracy at a Fixed Energy and Energy at a Fixed Accuracy . . . 102
4.6.4 Comparison of Two Model Runs . . . 103
4.6.5 Relation with the Characteristics of the Models . . . 103

5 Implementation **105**
5.1 Architecture of the Pipeline . . . 105
5.2 Model Selection Notebook . . . 107
5.2.1 Collection and Filtering . . . 108
5.2.2 Sources of the Metadata . . . 108
5.2.3 Identification of the Operating Mode . . . 110
5.2.4 Classification and Selection . . . 111
5.3 Experiment Orchestration . . . 112
5.3.1 Structure of an Experiment . . . 112
5.3.2 Run Table . . . 113
5.3.3 Preparation of the Environment . . . 115
5.3.4 Launching a Run and Collecting Its Results . . . 115
5.4 Evaluation Script . . . 116
5.4.1 Loading and Configuring the Model . . . 116
5.4.2 Sampling and Prompt Construction . . . 117
5.4.3 Warmup and Thermal Control . . . 117
5.4.4 Measured Inference . . . 118
5.4.5 Processing of the Responses . . . 119
5.4.6 Outputs of a Run . . . 120
5.5 Reproducibility and Extension . . . 122
5.5.1 Reproducing the Experiment . . . 122
5.5.2 Adding New Models . . . 122
5.6 Corrections After the Experiment . . . 123
5.6.1 Energy of the Processors and of Their Memory . . . 124
5.6.2 Summary of a Run . . . 125
5.6.3 Selection of the Models . . . 125

6 Results and Discussion **127**
6.1 Overview of the Results . . . 127
6.1.1 Energy and Time of the Study . . . 130

6.2 Evolution Across Successive Releases (RQ1) . . . 130
6.2.1 Accuracy and Energy per Question . . . 131
6.2.2 Intelligence Efficiency . . . 132
6.2.3 The Three Factors of the Intelligence Efficiency . . . 133
6.3 The Energy Cost of Accuracy (RQ2) . . . 135
6.3.1 The Cost Between Successive Releases . . . 136
6.3.2 The Accuracy Available at a Given Energy . . . 136
6.3.3 The Marginal Cost of Accuracy . . . 139
6.4 The Influence of Design Choices (RQ3) . . . 140
6.4.1 Model Size . . . 141
6.4.2 Architecture . . . 143
6.4.3 Numerical Precision . . . 144
6.4.4 Operating Mode . . . 144
6.4.5 Relative Influence of the Design Choices . . . 146
6.4.6 Execution Time . . . 147
6.5 Trajectories of the Model Families (RQ4) . . . 147
6.6 Discussion . . . 150
6.6.1 The Invariance of the Energy per Token . . . 150
6.6.2 The Origin of the Accuracy Gains . . . 151
6.6.3 The Rising Cost of Accuracy . . . 151
6.6.4 The Reinvestment of Efficiency Gains in Accuracy . . . 152
6.6.5 The Limits of a Single Ratio . . . 153
6.6.6 Implications . . . 153
6.7 Threats to Validity . . . 154
6.7.1 Construct Validity . . . 154
6.7.2 Internal Validity . . . 155
6.7.3 External Validity . . . 156
6.7.4 Conclusion Validity . . . 157

7 Conclusions and Future Work **158**

7.1 Answers to the Research Questions . . . 158
7.2 The Trajectory of Open Models . . . 159
7.3 Contributions . . . 160

7.4 Recommendations for Greener Models . . . 161
7.5 Limitations . . . 162
7.6 Future Work . . . 162
7.7 Final Remarks . . . 163

A Details of Results **184**

Results of Every Model Run . . . 184
Precision of the Energy Measurement . . . 186
Temperature of the Hardware . . . 186
Evolution Across Successive Releases . . . 188
The Energy Cost of Accuracy . . . 191
The Influence of Design Choices . . . 194
Trajectories of the Model Families . . . 197
Robustness of the Results . . . 199
Subjects of the Benchmark . . . 200

B Implementation Reference **202**

Model Selection Notebook . . . 202
Experiment Orchestration . . . 203
Evaluation Script . . . 205

C Disclosure of the Use of Generative Artificial Intelligence **207**

List of Figures

1 Weekly active ChatGPT users on consumer plans (Free, Plus, Pro), shown as point-in-time snapshots every six months, November 2022–September 2025. (Reprinted from [1]) . . 1

2 Conceptual map of sustainable software terminology. (Reprinted from [2]) 9

3 SRs published per year. (Reprinted from [2]) 10

4 Number of publications per year on Green AI. (Reprinted from [3]) 11

5 Industry involvement in Green AI scientific literature. (Reprinted from [3]) 11

6 The trends of the cumulative numbers of arXiv papers that contain the keyphrases "*language model*" (since June 2018) and "*large language model*" (since October 2019), respectively. (Reprinted from [4]) 13

7 Chronological display of LLM releases: blue cards represent "pre-trained" models, while orange cards correspond to "instruction-tuned" models. Models on the upper half signify open-source availability, whereas those on the bottom are closed-source. (Reprinted from [5]) 13

8 Accuracy of frontier models on Humanity's Last Exam compared with earlier benchmarks. (Reprinted from [6]) 38

9 Distribution of HLE's questions across domain categories. (Reprinted from [6]) 38

10 MMLU-Pro dataset construction pipeline. (Reprinted from [7]) 41

11 Distribution of disciplines and question sources in the MMLU-Pro dataset. (Reprinted from [7]) 42

12 Example prompt (few-shot examples excluded) and program synthesized by the largest evaluated model in the original MBPP paper. (Adapted from [8]) 45

13 LiveCodeBench Pro dataset statistics. (Reprinted from [9]) 47

14 Model performance on LiveCodeBench Pro. (Reprinted from [9]) 48

15 SWE-bench summary of the evaluation process. (Reprinted from [10]) 49

16 Complete pipeline of LLM data collection & selection. 74

17 Distribution of models over time by size category. 81

18 Distribution of models by operating mode (left) and architectural paradigm (right). 82
19 Distribution of models by size category. 84
20 Distribution of models over time by size category (final set). 84
21 Calculation of the total number of experimental runs. 85
22 Overview of the model evaluation and execution workflow. 87
23 Structure of the final prompt sent to each model. 91
24 Architecture of the experimental pipeline: its four stages and the files through which they communicate. 106
25 Experiment orchestration workflow. 112
26 Accuracy and GPU energy per question of every model run, with the Pareto front. 127
27 Accuracy of the model runs by release date. 131
28 GPU energy per question of the model runs by release date. 132
29 Intelligence efficiency of the model runs by release date. 132
30 Contribution of each factor to the change of intelligence efficiency between successive releases of the same family. 133
31 Pareto front of the model runs released until the end of each year. 136
32 Best accuracy among the model runs released so far, for every energy budget per question. 137
33 Energy per correct answer of every model run (left) and energy of each additional point of accuracy along the Pareto front (right). 139
34 GPU energy per token of the models against their total parameters (left) and their active parameters (right). 141
35 Best accuracy among the instruct runs released so far in each size category. 142
36 Difference of accuracy between the reasoning run and the instruct run of each pair, in three views (left), and ratio of their GPU energy per question (right). 145
37 Instruct runs of each family by accuracy and GPU energy per question, with an arrow from each release to the next. 148
38 Contribution of each factor to the change of intelligence efficiency between the first and the last release of each trajectory. 148
A.1 GPU energy of the repetitions of each model run, relative to its median. 186
A.2 GPU and CPU temperature along the questions of a run, as the mean of the runs of each operating mode and size category. 187

A.3 Average GPU and CPU temperature of each run of the main execution, in the order of execution. 188
A.4 GPU energy per question of the Tiny/Mobile model runs, by release date. 188
A.5 GPU energy per question of the Small/Edge model runs, by release date. 188
A.6 GPU energy per question of the Standard/Desktop model runs, by release date. 188
A.7 Tokens per question (top) and energy per token (bottom) of the model runs by release date. 188
A.8 Energy per token of the model runs relative to the value expected for their size, by release date. 188
A.9 Contribution of each factor to the change of intelligence efficiency between release years (instruct runs). 191
A.10 Best accuracy within each energy budget (left) and lowest energy per question for each accuracy target (right), among the model runs released so far. 193
A.11 Best accuracy within each energy budget (top) and lowest energy per question for each accuracy target (bottom), among the model runs of each size category released so far. 193
A.12 Energy per token of each model, measured and predicted from its active parameters by a power law fitted without that model. 194
A.13 Contribution of each factor to the difference of intelligence efficiency between the reasoning run and the instruct run of each pair. 195
A.14 Rank correlation between the characteristics and the results of the model runs. 196
A.15 Last instruct release of each family in each year, by size category: accuracy, tokens per question and GPU energy per question. 197
A.16 Difference of each family from the average of the families, at equal release date and size category, with 95% confidence intervals (number of instruct runs in parentheses). . . . 199
A.17 Accuracy and GPU energy per question of each subject, relative to that of all the questions, as the median of the instruct runs and of the reasoning runs. 201
A.18 GPU energy per question of each subject relative to that of all the questions, for every model run. 201

List of Tables

1 Annual compute for inference and model training, various workload models (A100 GPU-hrs). (Data from [11]) . . . 14
2 Overview of the energy measurement tools explored in this section. . . . 19
3 Practices for reducing bias in energy measurements. (Adapted from [12]) . . . 27
4 Comparison of the inference engines explored in this section. . . . 32
5 Overview of key LLM evaluation benchmarks across domains. . . . 36
6 Statistics of problems collected in LiveCodeBench, as of May 2024. (Adapted from [13]) . . . 46
7 Comparison of the studies most closely related to this thesis. . . . 58
8 Specifications of the STLab server, of each of its two CPUs and of its GPU. . . . 71
9 Versions of the main software components used in the experiment. . . . 73
10 Selected LLM families with corresponding developers and geographic origins. . . . 75
11 Selected LLM releases with corresponding HF collection IDs. . . . 75
12 Rules of the initial filtering, with examples of the keywords and types used to identify each kind of model. . . . 77
13 Size categories and corresponding total parameter count ranges. . . . 80
14 Distribution of MMLU-Pro questions by category before and after stratified sampling. . . . 89
15 The values recorded for every prompt, grouped by what they describe, with their unit and source. . . . 96
16 Metadata collected for each model and the sources used to obtain it. . . . 108
17 Columns of the `selected_models.csv` file, which has one row per model run. . . . 111
18 Methods of the `RunnerConfig` class implemented for the experiment. . . . 113
19 Values of the model-dependent arguments of the evaluation script, according to the type of model. . . . 116
20 Parameters of the vLLM engine that implement the configuration described in the methodology. . . . 117
21 Dynamic actions of the thermal-regulation loop according to the state of each component. 117

22 The fields of the aggregated `run_data.json` file, grouped by the kind of quantity they report. 121
23 Corrections made after the main execution of the experiment. 124
24 Model runs on the Pareto front of accuracy and energy per question (all in instruct mode). 128
25 Yearly change of each metric among the instruct runs of each size category, with the p-value of its association with the release date. 131
26 Best model run released until the end of each year, at each energy budget and accuracy target (the last column covers the releases until March 2026). 137
27 Instruct run of each sparsely activated model compared with the dense instruct runs released up to six months before or after it. 143
28 Share of the variation of the energy per token, of the tokens per question and of the energy per question explained by the size and by the operating mode of the model runs. 146
29 Approach of each family in the final selection: form of the reasoning mode, ratio of its energy per question to that of the instruct run, and distinguishing choice. 149
A.1 Results of the instruct runs, by size category and release date. 184
A.2 Results of the reasoning runs, by size category and release date. 185
A.3 Temperature of the GPU and of the CPU during the runs: median of the model runs of each operating mode and size category. 187
A.4 Change of accuracy and of GPU energy per question between successive instruct releases of the same family. 191
A.5 Instruct runs of each size category released until the end of 2024 and afterwards: the typical model of each period, and the best accuracy available within each energy budget. 191
A.6 Instruct runs of the Ministral 3 models with 8-bit and with 16-bit weights. 195
A.7 Reasoning run against the instruct run of each pair: difference of accuracy in the three views and ratio of the GPU energy per question. 195
A.8 Responses at the output limit in the reasoning runs, and in the instruct runs with more than 5% of them. 197
A.9 Releases and model runs of each family in the final selection. 197
A.10 Change between the first and the last instruct release of each trajectory, and contribution of each factor to the change of intelligence efficiency. 197
A.11 Main results with the strict accuracy, with the GPU energy above the idle consumption and with the accuracy above the chance level. 199
B.1 Models excluded through the explicit list, and the reason for each exclusion. 202

B.2 Patches applied to the `tokenizer_class` entry of models that could not be loaded. . 203

List of Acronyms

LLM	Large Language Model
GPU	Graphics Processing Unit
TPU	Tensor Processing Unit
NPU	Neural Processing Unit
OSI	Open Source Initiative
AI	Artificial Intelligence
ML	Machine Learning
MoE	Mixture-of-Experts
GQA	Grouped-Query Attention
JIT	Just-In-Time
SoA	State of the Art
SSE	Sustainable Software Engineering
SR	Systematic Review
GSE	Green Software Engineering
NLP	Natural Language Processing
LM	Language Model
SLM	Statistical Language Model
NLM	Neural Language Model
PLM	Pre-trained Language Model
API	Application Programming Interface
REST API	Representational State Transfer Application Programming Interface
CPU	Central Processing Unit
RAPL	Running Average Power Limit
DRAM	Dynamic Random-Access Memory
RAM	Random-Access Memory
NVIDIA SMI	NVIDIA System Management Interface

NVML	NVIDIA Management Library
AMD SMI	AMD System Management Interface
ROCm SMI	ROCm System Management Interface
TDP	Thermal Design Power
HLE	Humanity's Last Exam
GPQA	Graduate-Level Google-Proof Q&A
MMLU	Massive Multitask Language Understanding
AIME	American Invitational Mathematics Examination
PIQA	Physical Interaction Question Answering
STEM	Science, Technology, Engineering and Mathematics
GGUF	GPT-Generated Unified Format
ARM	Advanced RISC Machine
UNESCO	United Nations Educational, Scientific and Cultural Organization
WWW	World Wide Web
RQ	Research Question
MBPP	Mostly Basic Python Problems
STLab	Software Technologies Laboratory
VRAM	Video RAM
IMO	International Mathematical Olympiad
PhD	Doctor of Philosophy
SFT	Supervised Fine-Tuning
MAA	Mathematical Association of America
AMC 10	American Mathematics Contest 10
AMC 12	American Mathematics Contest 12
USAMTS	United States of America Mathematical Talent Search
ICPC	International Collegiate Programming Contest
IOI	International Olympiads in Informatics
LCB	LiveCodeBench
KV	Key-Value
USA	United States of America
HF	Hugging Face
JSON	JavaScript Object Notation
Regex	Regular Expression

CSV Comma-Separated Values

OOM Out-Of-Memory

Chapter 1

Introduction

1.1 Context

As adoption rates climb rapidly across industries and among the general public, Large Language Models (LLMs) are transforming daily life. Enabled by advancements in deep learning and specialized hardware, including Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), and Neural Processing Units (NPUs), and propelled by increasing data diversity and algorithmic innovations, these models have demonstrated significant potential in domains such as Software Engineering, Drug Discovery, Finance, Medicine, Law, and Education [14]. This technical maturity is further democratized by user-friendly chat interfaces from major providers, which make *state-of-the-art* models, such as OpenAI's GPT-5 [15], Anthropic's Claude Sonnet 4.5 [16] or Google's Gemini 3 [17], highly accessible. By July 2025, ChatGPT alone had reached 700 million weekly active users, around 10% of the global adult population, who sent roughly 18 billion messages per week [1] (Figure 1).

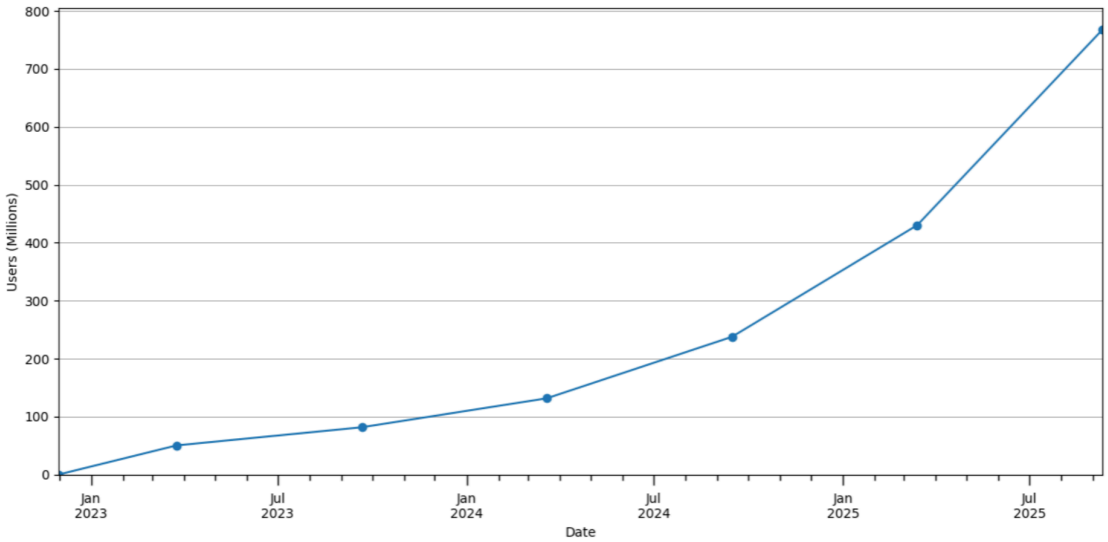


Figure 1: Weekly active ChatGPT users on consumer plans (Free, Plus, Pro), shown as point-in-time snapshots every six months, November 2022–September 2025. (Reprinted from [1])

Today, successive iterations of these models are released at a rapid cadence, requiring vast infrastructure and energy resources to sustain development and accommodate a growing volume of users and use cases. This trajectory introduces significant sustainability challenges [18], [19], while the ubiquitous nature of these models raises critical concerns regarding data privacy and the security of user-generated content [20]. These privacy concerns, in particular, have increased the interest in local, open alternatives to closed, cloud-based services such as ChatGPT or Claude [21].

Currently, the diversity of available open models[1] is unprecedented. While several benchmarks have emerged to evaluate their performance across domains such as code generation, reasoning, and instruction following, the Green Software community has simultaneously sought to quantify their environmental footprint by measuring energy consumption and runtime [18], [19]. Open models are also the only ones whose efficiency can be studied independently. The energy consumed by closed models is known only through the figures that their providers choose to disclose, or through estimates made from the outside [24], [25]. An open model, in contrast, can be executed locally, on hardware and software that stay the same for every model, so that the differences measured between two models can be attributed to the models themselves.

However, despite the economic and environmental pressure to develop more sustainable solutions [26], there is still little research on how energy efficiency actually evolves from one model release to the next. The few studies that follow efficiency over time change the hardware together with the models, rely on figures reported by the providers, or leave accuracy out of the comparison, as discussed in Section 2.7. It remains unclear whether energy efficiency is improving with each consecutive release, or whether it is being systematically compromised in favour of raw performance.

With the continuous expansion of the LLM landscape and its associated environmental footprint, there is a crucial need to analyse the efforts being made to improve energy efficiency, identifying the energy cost of accuracy improvements.

1.2 Motivation

The rapid evolution of LLMs is currently defined by a fundamental tension between efficiency-oriented innovations and performance-driven complexity. On the one hand, architectural advancements such as Mixture-of-Experts (MoE), Grouped-Query Attention (GQA) and post-training quantization techniques, paired

[1] Models whose learned parameters (weights) and inference code are publicly released for local execution and fine-tuning, but whose full training data, code, or datasets may remain proprietary. Unlike "Open Source AI", which requires full transparency of the training pipeline under Open Source Initiative (OSI)-approved licences, open models like Google's Gemma [22] or Meta's Llama [23] are governed by permissive custom licences that facilitate accessibility and private deployment while restricting certain use cases.

with optimized inference frameworks like vLLM [27] and high-throughput hardware like NVIDIA H100 GPUs [28], have significantly reduced the energy costs of inference [29], [30], [31]. On the other hand, the industry-wide push towards "raw performance", with capabilities like enhanced reasoning, multi-modal integration and vastly extended context windows, has significantly increased computational costs [32], [33], [34]. These advancements often demand a higher number of tokens[2] or longer processing times, potentially offsetting the gains achieved through architectural optimization.

This research is motivated by the need to shift the paradigm from "Red AI", which prioritizes state-of-the-art results regardless of computational cost, towards "Green AI", where efficiency is treated as a core evaluation metric [35], [36, Ch. 3, p. 4]. This thesis investigates the responsible advancement of LLMs, highlighting the importance of increasing "intelligence efficiency": the accuracy that a model delivers relative to the energy it consumes. The term follows Saad-Falcon et al. [37], who measure the intelligence efficiency of local models as task accuracy per unit of power ("intelligence per watt") and per unit of energy. Throughout this document, *energy efficiency* is used as the broader term, covering any relation between the energy a model consumes and what it delivers, such as the tokens it generates or the questions it answers. Intelligence efficiency is the particular case that relates energy to accuracy, and is the main focus of the analysis. Establishing this balance is imperative for a future where LLMs are sustainably integrated into both technology ecosystems and daily life.

Furthermore, the transition towards models that maximize intelligence per joule is a matter of global equity. According to UNESCO, most AI infrastructure is concentrated in high-income countries. It reports an estimate from the International Telecommunication Union that only 5% of Africa's AI talent has access to the computing power needed to build or use generative AI [38]. Developing efficient yet highly capable models is, therefore, a crucial step towards democratizing LLMs, making them viable in environments characterized by limited connectivity, scarce energy resources and restricted access to high-end hardware.

1.3 Aims and Objectives

The main goal of this dissertation is to perform a comparative analysis of successive releases within open LLM families, such as Gemma [22] and Llama [23]. By evaluating energy consumption, execution time, and response accuracy against established benchmarks, this research aims to investigate the trade-offs between performance gains and energy consumption as the LLM landscape advances, identifying the

2 The basic discrete unit of information used by an LLM. Tokens represent sequences of characters—ranging from a single punctuation mark to a full word—mapped to numerical vectors. Tokenization allows the model to efficiently process a fixed vocabulary while decomposing complex or unknown words into manageable subcomponents.

progress being made to improve the energy efficiency of models. In pursuit of this goal, the following objectives will be tackled:

- Perform a comprehensive review of the literature on Green AI, Large Language Models, energy measurement and model evaluation, and identify what existing studies leave unanswered about how the efficiency of LLMs has evolved.
- Identify the factors that have to be controlled in order to fairly compare models that differ in size, design and release date.
- Select a diverse and representative set of open models, covering several families, sizes and release dates, within the limits of the available hardware.
- Engineer an automated, reproducible and publicly available experimental pipeline that runs every model under the same conditions and measures the energy consumption, execution time and accuracy of each of its responses.
- Carry out the experiment with repeated measurements, verify the reliability of the collected data, and make it publicly available.
- Conduct a statistical analysis of the results and interpret it in order to answer the main research questions, ultimately attempting to characterize the evolutionary trajectory of open LLMs regarding energy efficiency, determining whether the open LLM landscape is collectively following a similar path or diverging in its developmental approach.

1.3.1 Research Questions

To achieve its main goal, this study addresses the following Research Questions (RQs).

RQ1. **How has the energy efficiency of open LLMs evolved across successive releases?**

This RQ investigates whether the open LLM community is improving the efficiency of its models, i.e. whether newer releases use less energy to deliver similar or better results. It addresses a fundamental question: as models advance and improve, do they become more or less efficient?

RQ2. **What is the energy cost of the accuracy gained by newer models?**

While the first question follows efficiency over time, this RQ looks at the relation between the two quantities behind it. It examines whether improvements in model capability necessarily require increased energy consumption, or whether the LLM community is successfully developing models that are both more accurate and more energy-efficient, i.e. more intelligence-efficient. This reveals the true cost of performance improvements.

RQ3. **How do model design choices, such as size, architecture and operating mode, influence energy efficiency?**

Different LLM families and design approaches may have varying impacts on energy consumption and performance. This RQ explores how three design decisions influence the energy efficiency of the resulting models: the size of the model, its architecture, in particular whether it uses all of its parameters at once or only part of them, and its operating mode, that is, whether it answers directly or reasons before answering.

RQ4. **Do open model families follow a common trajectory, or do they diverge in their approach to efficiency?**

This RQ examines whether the openly accessible LLM ecosystem is collectively moving towards more efficient models (a convergent trend) or whether different model families are pursuing divergent strategies. This characterizes the current state of practice and the direction the field is heading.

1.4 Contributions

The main contributions of this dissertation are the following:

- **Evidence on how the efficiency of open LLMs has evolved.** The energy and accuracy of 47 open models, from 20 releases of six families published between July 2023 and March 2026, were measured under fixed conditions, a combination that none of the studies reviewed in Section 2.7 offers. Successive releases became more accurate but not more intelligence-efficient, as their responses grew longer while the energy per token remained fixed by the size of the model, even though a given accuracy came to require less energy (Section 6.6).
- **A method for comparing successive releases.** Chapter 3 identifies the factors that must be controlled to compare models of different sizes, architectures and operating modes, and Chapter 4 describes how each was handled. The analysis decomposes the intelligence efficiency into three

factors, the accuracy, the tokens generated per question and the energy per token, and complements it with the accuracy at a fixed energy and the energy at a fixed accuracy (Section 6.6.5).

- **A public dataset.** The records of the 202,740 inferences of the 545 runs, 530 in the main execution of the experiment and 15 in a supplementary one (Section 5.6), are available in the repository of the experiment [39], together with the summary and efficiency metrics of each run. Each record includes the energy, duration, token counts and hardware temperatures of the inference, and the complete response of the model, so that the answers can be analysed further without repeating the 50 days of computation of the experiment.
- **A reproducible experimental pipeline.** The same repository contains the complete pipeline used in the study, described in Chapter 5: the selection of the models from public metadata, the orchestration of the experiment, the control of the temperature of the hardware before each run, the measurement of every inference individually, and the notebooks that correct the recorded data and produce every table and figure of Chapter 6. It can be reused to evaluate new models as they are released (Section 5.5.2).

1.5 Related Publications

In addition, during my Master's degree, I analysed the energy efficiency of the Lua programming language across various interpreters and compilers, resulting in the following co-authored contributions:

[40] A. Brandão, D. Matos, M. Guimarães, S. Cunha, and J. Saraiva, "The Green Side of the Lua," in *2026 IEEE International Conference on Software Analysis, Evolution and Reengineering - Companion (SANER-C)*, 2026, pp. 201–209. doi: 10.1109/SANER-C67878.2026.00033.

[41] D. Matos, A. Brandão, M. Guimarães, S. Cunha, and J. Saraiva, "The Extended Green Side of the Lua," *SSRN preprint*, 2026, doi: 10.2139/ssrn.7661830.

In the first paper, we present an empirical study of Lua's runtime performance and energy efficiency across 25 official interpreter versions and Just-In-Time (JIT) compilers, demonstrating the substantial benefits of JIT compilation for improving both performance and energy efficiency in interpreted languages. By comparing our results with the findings reported by the popular energy ranking of programming languages

[42], we also found that Lua improved its ranking by 5 positions for energy consumption and 9 positions for execution time.

The second paper, a journal article that extends this study, has been submitted to Information and Software Technology and is currently under review. It evaluates 26 Lua execution environments, comprising 17 interpreter releases, 8 JIT compiler variants and an ahead-of-time compiler, on five benchmark suites, and compares them with C. A preprint of the article is available on SSRN, and its results are presented in detail in an online appendix[3].

Both studies take the same perspective as this dissertation, the evolution of energy efficiency across successive versions of a technology, and apply it to a programming language instead of language models.

1.6 Document Structure

This document adopts the following structure.

Chapter 1 introduces the thesis topic, establishes its context and motivation, outlines the core objectives and RQs, and summarizes the contributions of the work.

Chapter 2 establishes the State of the Art (SoA) by reviewing current literature on green software, large language models, energy measurement, and the execution and evaluation of LLMs. It ends with a review of the related work and of the gap that this dissertation addresses.

Chapter 3 presents the unique complexities and challenges of conducting an evolutionary analysis of LLMs, explaining how key factors must be controlled to fairly evaluate them.

Chapter 4 details the methodology of the experiment, specifying the experimental environment, the selection of the models, the experimental design, the execution and measurement of each run, the efficiency metrics and the analysis of the collected data, together with the reasons behind each decision.

Chapter 5 describes the software pipeline that implements this methodology: its architecture, the selection of the models, the orchestration of the experiment and the evaluation script, as well as the means to reproduce and extend the experiment and the corrections made after its main execution.

Chapter 6 presents and discusses the results of the experiment, answering each of the RQs and identifying the threats to the validity of the study.

Chapter 7 summarizes the main findings and contributions of the work, derives recommendations for the development of more energy-efficient models, acknowledges the limitations of the study, and outlines directions for future research.

[3] https://the-green-side-of-the-lua.wasmer.app/

Chapter 2

State of the Art

2.1 Green Software and Sustainability in AI

The concept of sustainable development, formally introduced by the Brundtland Report in 1987 [43], established a foundational framework centred on meeting present needs without compromising the ability of future generations to meet their own. This principle has since permeated various domains, including software engineering, where it evolved into the notion of Sustainable Software Engineering (SSE).

As noted by Venters et al. [44], there is a notable lack of consensus on how to define the concept of sustainability within the software domain, where it takes on a dual role: the sustainability of the software itself and of its development lifecycle, and the use of software as a tool to support sustainability goals outside it. Calero et al. [45] call these two perspectives *software sustainability* and *software as part of sustainability*, and they are also known as *sustainability in software* and *sustainability by software* [2].

SSE is intrinsically linked to the concept of Green Software, which primarily represents the environmental dimension of sustainability. The term "sustainability" encompasses diverse dimensions, namely environmental, social, economic, technical, and individual [2]. However, as Lago et al. [46] observe, within the environmental sphere, software can be characterized as "greener" when it minimizes power consumption through enhanced energy efficiency. While SSE aims to address the full spectrum of sustainability concerns, Green Software specifically targets the reduction of software's ecological footprint, most commonly through energy efficiency [47].

The intrinsic relationship between these concepts gives rise to the notion of *green in software* and *green by software*. "Green in software" refers to the development of software that is itself energy-efficient and resource-conscious, focusing on the software product and its development process [45]. In contrast, "Green by software" describes the use of software as a tool or instrument to support sustainability goals in other domains, such as optimizing energy consumption in buildings or managing supply chains [45].

The conceptual map in Figure 2 illustrates the relationships among these terms, specifically highlighting the connection between sustainable and green software.

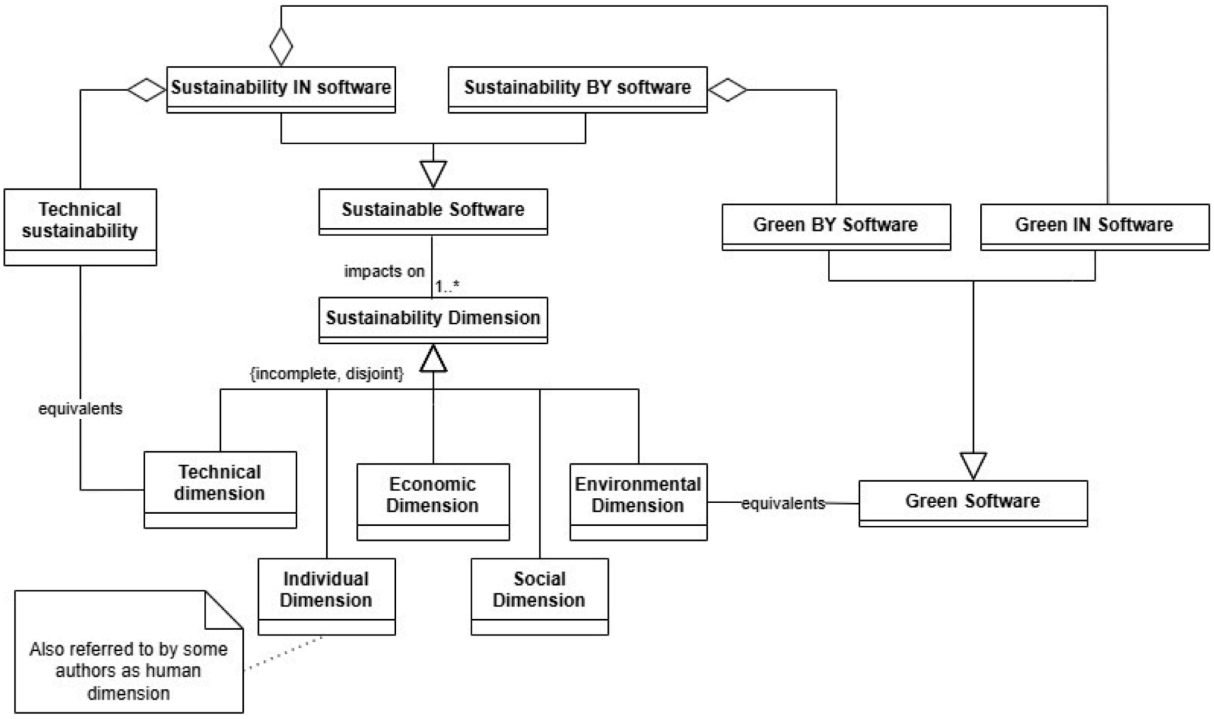


Figure 2: Conceptual map of sustainable software terminology. (Reprinted from [2])

This research investigates how the energy consumption of LLMs evolves relative to their accuracy. It is therefore concerned with *sustainability in software* and, more specifically, with *green in software*.

While it is difficult to pinpoint exactly when Green Software Engineering (GSE) gained mainstream traction, the evolution of the SSE landscape provides a clear indicator. A tertiary study by García-Mireles et al. [2], covering 80 Systematic Reviews (SRs), found that, among the various dimensions of software sustainability, the environmental dimension received the most attention, accounting for 55% of the research focus, with a "predominant emphasis on energy efficiency". The overall growth of SSE research can therefore be taken as a reasonable indication of the growth of GSE.

The number of SRs published per year in the SSE field since 2011, reported in Figure 3, shows a clear trend. Although the growth is not uniform, publications have increased from one or two per year until 2013 to a median of seven per year from 2016 onward, with a recent surge to more than 10 per year in 2023 and 2024.

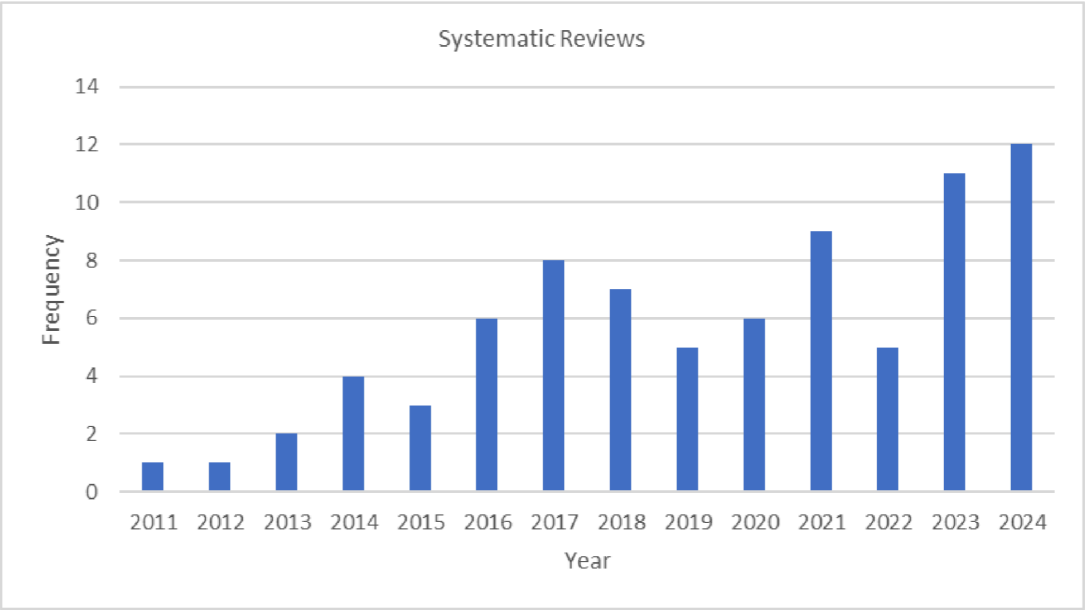


Figure 3: SRs published per year. (Reprinted from [2])

2.1.1 Green AI

The concept of Green AI is a more recent but rapidly evolving subfield. Green AI was defined in 2020 as "AI research that yields novel results while taking into account the computational cost, encouraging a reduction in resources spent" by Schwartz et al. [35], who coined the term. However, a systematic review of 98 primary studies on Green AI found that most of the literature addresses this "reduction in resources spent" at the level of energy efficiency (81 studies), while the higher-level definitions of "Carbon Footprint" and "Ecological Footprint" are addressed by only 20 and 9 studies, respectively [3].

This focus on energy efficiency as a proxy for environmental impact is well justified. As Schwartz et al. [35] observe, in just six years (2012 to 2018) there was an estimated 300,000-fold increase in the amount of compute used to train deep learning models. More computation requires more energy, which in turn drives carbon emissions. Schwartz et al. describe the research behind this trend as *Red AI*: work that seeks better accuracy through the use of massive computational power, with little attention to its cost. Their proposal is to make efficiency an evaluation criterion alongside accuracy, so that a result is judged not only by how good it is, but also by how much it costs to obtain. This is the idea behind the *intelligence efficiency* studied in this thesis, which relates the accuracy of a model to the energy it consumes [37].

The field of Green AI started gaining significant attention around 2020, with a sharp rise in research publications from that year onward. According to Verdecchia et al. [3], 76% of the papers examined were published since 2020, highlighting that year as a clear turning point in the field's growth (Figure 4). The research focuses primarily on *Monitoring*, *Hyperparameter Tuning*, *Model Benchmarking* and

Deployment, topics that together cover 61% of the papers. It also concentrates on training: 49 of the 98 studies focus on the training phase, and only 17 on inference.

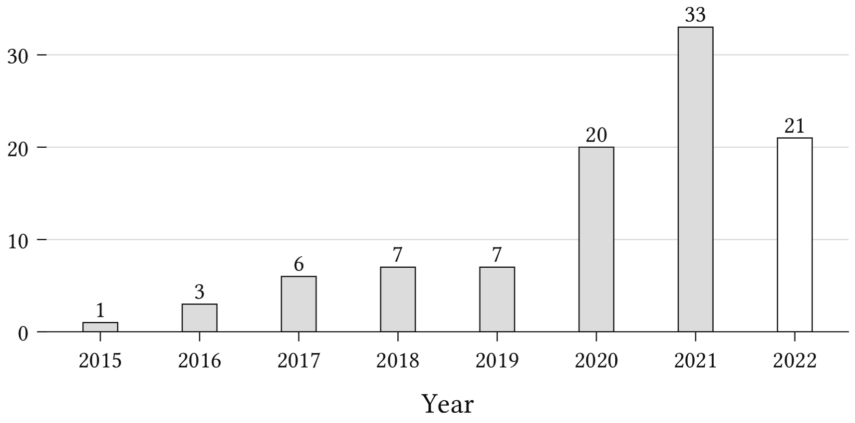


Figure 4: Number of publications per year on Green AI. (Reprinted from [3])

The research agenda of the field is still taking shape. In 2025, the "Greening AI with Software Engineering" workshop brought together 29 participants, from practitioners to academics, who identified and prioritized the key challenges of the field: energy assessment and standardization, benchmarking practices, sustainability-aware architectures, runtime adaptation, empirical methodologies, and education [48]. The first two are the ones most directly addressed by this thesis, which measures the energy of LLM inference across models with the same benchmark suite and experimental protocol.

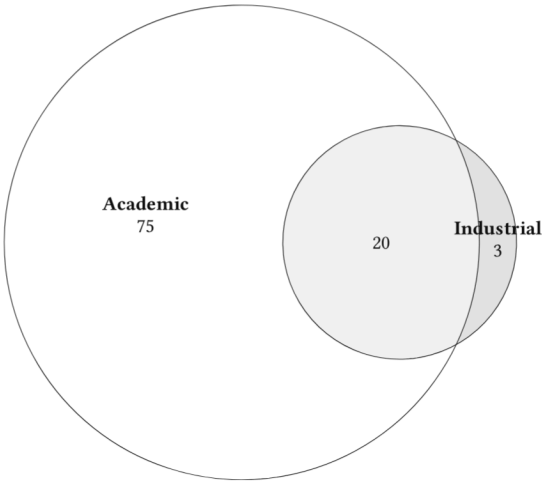


Figure 5: Industry involvement in Green AI scientific literature. (Reprinted from [3])

Despite this academic momentum, a critical gap persists in the form of limited industry involvement. Verdecchia et al. [3] found that a substantial majority of research on Green AI is conducted within academic settings, with only a small fraction of studies involving industrial partners or being authored by industry researchers (Figure 5). This lack of industry participation is a significant concern, as it can lead to a

disconnect between academic research and the practical challenges and needs of real-world software development.

The low level of industry involvement also means that the findings and proposed solutions often lack validation in industrial contexts, which is essential for ensuring their practical applicability and impact. This gap highlights a need to investigate whether the sustainable software practices identified in academic research are being actively adopted, adapted, or ignored in real-world AI development and deployment, a necessary step to determine if these practices are genuinely shaping industry behaviour or remain confined to scholarly discourse. This thesis approaches that question from the outside: by measuring the models that companies release, it examines whether efficiency is improving in what the industry actually ships.

2.2 Large Language Models: Evolution and Efficiency

Machines understanding and communicating using human language has long been a research challenge [49]. Natural Language Processing (NLP) [50] emerges as the AI field focused on this goal, relying fundamentally on Language Modelling, which aims to predict the probability of sequences of linguistic units (tokens) such as words or sentences. Over time, Language Models (LMs) have evolved from early Statistical Language Models (SLMs) [51] to Neural Language Models (NLMs) [52], then to Pre-trained Language Models (PLMs), and now to today's Large Language Models.

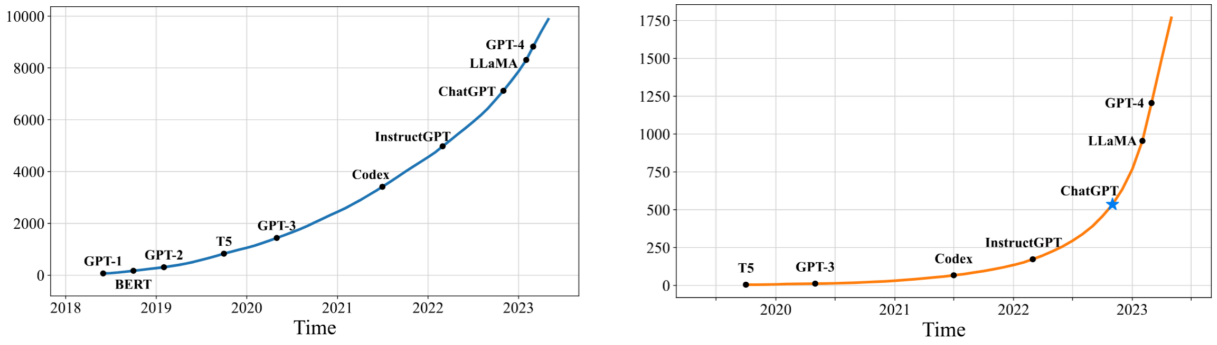
The term "Large Language Model" was coined by the research community as a way of describing large-sized PLMs, since an LLM is essentially the result of scaling the training dataset or model size of a regular PLM, often into tens to hundreds of billions of parameters [4] and from a few hundred gigabytes to several terabytes of data [53], [54]. This scale is associated with abilities that are not observed in smaller PLMs (known as *emergent abilities*) [55]. It is also related to a major reason for the swift advancement of the field in the past few years: data diversity.

In recent years, the continuous expansion of the World Wide Web (WWW) has led to a surge in diverse, internet-based data sources. From social media and blogs to forums and reviews, this proliferation of heterogeneous text prompted a shift towards training models on multi-domain corpora, a transition that yielded significant improvements in generalization and multitasking capabilities [56].

Supporting this data-driven expansion is an enormous increase in computational scale [57]. Between 2012 and 2018, the amount of compute used in the largest AI training runs doubled every 3.4 months, as documented by OpenAI[4]. This acceleration was made possible by breakthroughs in specialized hardware, including GPUs, TPUs, and NPUs [28], [58], [59], alongside algorithmic innovations that improved

4 https://openai.com/index/ai-and-compute

efficiency and scalability [60], [61], [62]. Together, these advances, combined with the explosion of accessible, diverse data, created the conditions for rapid progress. The release of ChatGPT marked a turning point, triggering an exponential surge in research activity. As illustrated in Figure 6, the number of arXiv papers on large language models and language models skyrocketed thereafter, reflecting the field's momentum.



(a) Query="Language Model" (b) Query="Large Language Model"

Figure 6: The trends of the cumulative numbers of arXiv papers that contain the keyphrases "*language model*" (since June 2018) and "*large language model*" (since October 2019), respectively. (Reprinted from [4])

Today, a plethora of state-of-the-art models are available. These include open models for local deployment and proprietary models offered by major providers such as OpenAI and Google, accessible to individuals and businesses through intuitive chat interfaces and dedicated APIs. Figure 7 showcases open and closed LLM releases from recent years.

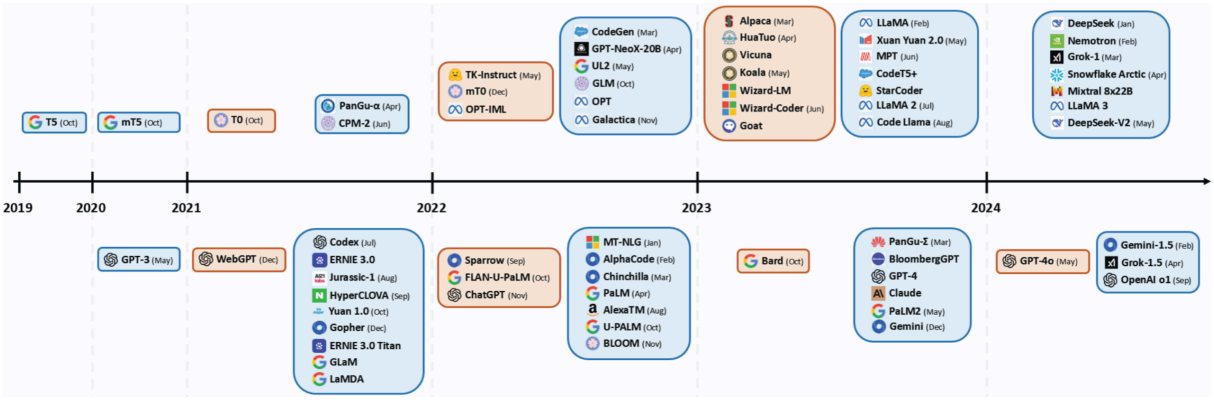


Figure 7: Chronological display of LLM releases: blue cards represent "pre-trained" models, while orange cards correspond to "instruction-tuned" models. Models on the upper half signify open-source availability, whereas those on the bottom are closed-source. (Reprinted from [5])

As the capabilities of these models increased, so did their sizes. Villalobos et al. [63] found that the growth in the size of notable Machine Learning (ML) models accelerated sharply around 2018, with language models as the main driver: after increasing by seven orders of magnitude between 1950 and 2018, their size increased by another five in just four years. Between 2018 and 2022, model size doubled every 4 to 8 months, depending on whether a group of exceptionally large models is included in the trend. OpenAI has not disclosed the size of GPT-4, released in 2023, but an unofficial industry analysis estimates it at over one trillion parameters [64].

This multiplication in model size has significantly increased the computational and energy demands for both training[5] and inference[6], leading to a corresponding rise in environmental impact through increased CO_2 emissions. It is estimated that training GPT-3 required 1,287 MWh [65]. According to Argerich and Patiño-Martínez [19], this amount is equivalent to the average annual energy consumption of 420 people.

While significant research has focused on reducing the energy consumption of large language models [66], [67], [68], the focus has largely been on the energy use of training, often overlooking the operational (inference) phase [3]. Yet, as these models are increasingly integrated into everyday applications and user bases grow rapidly, inference is becoming a dominant factor in their overall environmental impact. Patterson et al. [65] note that most companies already spend more energy on serving models than on training them, citing industry estimates that inference represents 80% to 90% of the ML workload. According to one industry estimate, adding generative AI to web search requires at least four to five times more computing per search [69]. Chien et al. [11] estimate that, for a "ChatGPT-like" service, one year of inference requires 25 times the compute, and produces 25 times the carbon emissions, of training GPT-3 (Table 1).

Workload Model	**Inference Cost** (GPU-hrs)	**Training Cost** (GPU-hrs)	**Inference/Training**
ChatGPT-RR	55,966,667	2,236,467	25×
Google-RR	3,099,154,167	2,236,467	1,386×

Table 1: Annual compute for inference and model training, various workload models (A100 GPU-hrs). (Data from [11])

Argerich and Patiño-Martínez [19] further elaborate on the importance of prioritizing energy efficiency during inference. Their argument rests on three points:

5 The computationally intensive process of developing a model by exposing it to vast datasets, allowing it to adjust internal parameters (weights) to recognize linguistic patterns and relationships.

6 The operational phase where a trained model processes a specific user prompt to generate a response, using its fixed parameters to predict the most likely output without further learning.

1. Models are trained once but used for inference millions of times, leading to cumulative energy use that can exceed training consumption.

2. Training can be scheduled during periods of higher renewable energy availability, whereas inference is often time-sensitive and its execution cannot be delayed.

3. Training can be run in regions where the electricity has a low carbon intensity, while inference is usually location-constrained by latency, network bandwidth, and data privacy regulations.

As discussed in Section 1.2, the evolution of LLMs is shaped by two forces that pull in opposite directions. Architectural advances such as Mixture-of-Experts [29] and Grouped-Query Attention [30], post-training quantization [31] and more efficient hardware [28] reduce the cost of processing each token, while capabilities such as reasoning, multimodality and large context windows increase the number of tokens that have to be processed [32], [33], [34]. Whether the balance is positive, that is, whether newer models extract more performance from each joule, can only be established by measuring it. The remainder of this section introduces two notions that are needed to understand both sides of this tension: how a model produces a response, and the different ways in which models are trained to respond.

2.2.1 From Prompt to Response

An LLM does not operate on text directly. The input is first divided into *tokens*, units such as words or pieces of words, and it is tokens that the model receives and produces [4]. Most current LLMs are based on the decoder of the Transformer architecture [4], [60] and generate text *autoregressively*: one token at a time, each computed from the prompt and from all the tokens generated before it, until the model produces a token that marks the end of the response [27].

This process has two distinct phases, known as *prefill* and *decode* [70]. In the prefill phase, the model processes all the tokens of the prompt, which can be done in parallel, and computes the first token of the response. In the decode phase, it produces the remaining tokens sequentially, with one pass through the model for each of them. To avoid processing the entire sequence again at every step, the intermediate results that the attention mechanism computed for the previous tokens, known as keys and values, are kept in memory in the *Key-Value (KV) cache* [27]. The two phases use the hardware very differently. Prefill is limited by computation and takes full advantage of the parallelism of the GPU. Decode handles a single token of each request per pass and is limited by memory access, which leaves much of the computing capacity of the GPU unused and makes this phase responsible for most of the time a request takes [27], [70], [71].

These mechanics explain what makes an inference more or less expensive:

- **The length of the response.** Every generated token requires a pass through the model, so the time and the energy of a request grow with the number of tokens it generates.
- **The length of the prompt.** The prompt determines the cost of the prefill phase and the size of the KV cache that every decode step has to read. The number of tokens that a model can handle in one request, counting both the prompt and the response, is limited by its *context window* [34].
- **The size and design of the model.** At each decode step, the weights of the model and all the cached keys and values have to be loaded from memory, and this memory traffic is the bottleneck of the phase [30]. The cost of a step therefore grows with the number of parameters of the model, and also depends on how its attention mechanism is designed. With GQA, each key and value is shared by a group of attention heads instead of being computed for every head, which reduces the size of the KV cache with little loss in quality [30]. This design has been gradually replacing the original multi-head attention in recent small models [72].
- **Batching.** Since the decode phase leaves computing capacity unused, an inference engine can process several requests in the same pass. The cost of loading the weights is then shared among them, which increases throughput [27] and reduces the energy spent on each request [73].

The empirical evidence on how much each of these factors weighs on the energy of an inference is reviewed in Section 2.7.

2.2.2 Instruction-tuned and Reasoning Models

The models of a family are usually released in more than one variant, and these variants differ in how they were trained to respond.

Instruct models. *Instruction tuning* consists of fine-tuning a pre-trained model on a collection of tasks written as instructions in natural language, together with the expected responses, which improves its ability to follow instructions and to generalize to tasks it has not seen [4]. The resulting *instruct* models, often named chat models, are the ones used in assistants, and answer a request directly. Their answers can be guided by including a few solved examples of the task in the prompt, a technique known as *few-shot* prompting, as opposed to *zero-shot* prompting, in which no examples are given [4].

Reasoning models. Models answer complex questions more accurately when they write down intermediate steps before the final answer. Wei et al. [74] showed that including a few examples of such a *chain*

of thought in the prompt significantly improves the performance of a model on arithmetic, commonsense and symbolic reasoning tasks. *Reasoning* models turn this behaviour into a trained capability: they learn, typically through reinforcement learning, to produce a long chain of thought before answering, without being asked to [75], [76]. OpenAI's o1 is an example among closed models [75], and DeepSeek-R1 among open ones [76]. DeepSeek-R1 writes its reasoning between `<think>` tags, followed by the final answer. Its developers also *distilled* it into smaller models: existing open models with 1.5 to 70 billion parameters were fine-tuned on 800,000 samples generated by DeepSeek-R1, so that they learn to reason in the same way [76].

Dual-mode models. More recently, both behaviours have been combined in a single model. Qwen3 integrates a *thinking mode* and a *non-thinking mode*, selected through a parameter of the request [77]. Other recent models offer a similar switch, or several levels of reasoning [78]. This thesis refers to models of this kind as *dual-mode* models.

The cost of reasoning. The chain of thought of a reasoning model is made of output tokens, generated one by one like any others (Section 2.2.1). Reasoning is therefore paid for in tokens, memory and time [32]. Chung et al. [73] found that reasoning models produce one to two orders of magnitude more output tokens per request than standard chat models. On multiple-choice questions, Dauner and Socher [79] measured an average of 37.7 tokens per answer for standard models, to which reasoning models added an average of 543.5 tokens of reasoning. In the AI Energy Score leaderboard, reasoning models use, on average, 30 times more energy than models without reasoning, or with reasoning turned off [78]. Whether the gain in accuracy compensates for this additional energy is one of the questions examined in Section 2.7.

Implications for evaluation. Reasoning models also react differently to the prompt. The developers of DeepSeek-R1 report that few-shot prompting consistently degrades its performance, and recommend describing the problem directly, without examples [76]. Since their responses are much longer, they also need a higher limit on the number of tokens they can generate. These differences make it difficult to evaluate instruct and reasoning models under exactly the same conditions, a problem discussed in Section 3.1.

2.3 The Landscape of Energy Measurement Tools

Precise and reliable energy monitoring is fundamental to the field of Green Software. Measuring the total power consumption of a physical device is relatively simple when using a power meter, provided that the researcher has physical access to the machine. However, quantifying the energy used by specific software

running on that device is a much more difficult task. Argerich and Patiño-Martínez [19] identify three primary reasons for this:

1. Software often runs on servers or machines that are not physically accessible, such as those hosted in the cloud. It may also operate within virtualized environments, sharing resources across one or more physical servers. In those scenarios, direct measurement with a power meter is not feasible.

2. Even with physical access to the machine running the software, it is not possible to isolate energy consumption by individual hardware components such as the Central Processing Unit (CPU), memory, GPU, or hard disk. This would require dedicated power monitors connected to each component and synchronized in order to capture consumption data only during software execution.

3. Modern computers feature multiple cores and execute multiple processes simultaneously or near-simultaneously. To accurately estimate the energy consumption of a specific software application, it is essential to precisely track when the software runs, typically with millisecond-level timing accuracy.

To address these limitations, a wide range of energy measurement tools have been developed. Ranging from low-level hardware counters to process-level monitors and high-level wrappers, these tools enable researchers to measure the energy usage of specific applications or processes without the need for additional hardware or physical access to the machine. While impressive, these tools do not come without their own limitations, as Argerich and Patiño-Martínez [19] point out:

1. These tools depend on sensors and low-level libraries that are usually constrained by the hardware they run on and not compatible with virtualization.

2. The tools can be limited by the platform or operating system in which they run, such as Kubernetes, Linux, Windows and macOS, where functionality and compatibility may vary.

3. Few of these tools are optimized specifically for the heterogeneous workloads of LLMs, where both the CPU and the GPU are used, in order to take advantage of hardware acceleration. Multiple tools are often needed for each device, such as Intel Running Average Power Limit (RAPL) [80] for CPU and NVIDIA SMI [81] for GPU.

The tools explored in this section can be organized in three layers, each building on the one below it. At the bottom are the **hardware interfaces** provided by the manufacturers, which expose the readings of sensors and counters built into the hardware: Intel RAPL for processors, and NVIDIA Management Library (NVML) and AMD System Management Interface (AMD SMI) for GPUs. Above them are **system-level**

monitors, such as Scaphandre and EnergiBridge, which collect these readings periodically and present them in a uniform way, in some cases attributing them to individual processes. At the top are **trackers aimed at ML workloads**, such as CodeCarbon, CarbonTracker and EnviroLLM, which add features like carbon emission estimates and integration with training or inference code. This layering has an important consequence: a tool can never be more accurate than the interface it reads from, and each additional layer introduces its own assumptions and sampling behaviour.

Table 2 summarizes the tools covered, which are described in detail in the following subsections.

Tool	Covers	How energy is obtained	Evidence on accuracy and main limitations
		Hardware interfaces	
Intel RAPL	CPU package and DRAM (Intel, and most AMD processors)	Cumulative energy counters in the processor, updated about every 1 ms	Correlation of 0.99 with wall power [82]. Memory values deviate up to 20% [83]. Counters overflow and cover only part of the system.
NVML and NVIDIA SMI	NVIDIA GPUs	On-board power sensor, read as power samples or as a cumulative energy counter	Power error of about ±5%. On some GPUs only 25% of the runtime is sampled, with energy errors of up to 70% when used naively [84].
AMD SMI	AMD GPUs and processors	On-board sensors, read as power samples	No independent validation found. Energy counter reported as unusable in its predecessor [85].
		System-level monitors	
Scaphandre	CPU, per host and per process	RAPL, shared between processes by CPU time	Correlation of 0.947 with an external meter, 0.85% energy overhead [86]. Does not cover GPUs.
EnergiBridge	CPU and NVIDIA GPU, on Linux, Windows and macOS	Processor registers and system sensors, sampled at a fixed interval	Not validated against external measurements. Coverage differs between platforms [87].
		Trackers for ML workloads	
CodeCarbon	CPU, memory and NVIDIA GPUs	RAPL and NVML when available, estimates otherwise, sampled every 15 s	Overhead of about 1% [88]. Errors of up to 40% against external measurements [89].
CarbonTracker	CPU and NVIDIA GPUs, during training	RAPL and NVML, with predictions from the first epochs	Prediction errors of 4.9% to 19.1% [90]. Designed for training, not inference.
EnviroLLM	CPU, memory and NVIDIA GPUs, for local LLMs	NVML power samples every 2 s, estimates otherwise	Tested on a single machine, with no external validation [91].

Table 2: Overview of the energy measurement tools explored in this section.

2.3.1 Intel RAPL

Intel RAPL is an interface built into Intel processors that reports the energy consumed by the processor and allows system software to limit its power. The concept was first proposed for estimating and capping the power of memory [80], and became available to software with the Sandy Bridge microarchitecture [82], [92]. It has since become the most widely used source of CPU energy data, and is also available on most AMD processors since 2017 [93].

How it works. RAPL divides the processor into *domains*, each with its own energy counter: the whole package, the cores, the integrated graphics, the DRAM attached to the processor, and, on some models, the entire platform. Which domains are available depends on the processor model [86], and AMD processors only provide the package and core domains [93]. Each counter accumulates the energy consumed by its domain since the processor was started, and is updated approximately every millisecond [82]. The energy consumed by a piece of software is obtained by reading a counter before and after its execution and subtracting the two values. On Linux, the counters can be read directly from the processor's registers or, more conveniently, through kernel interfaces such as the power capping framework (`powercap`) [94], which exposes them as files. Raffin and Trystram [93] compare these access mechanisms in detail.

Accuracy. The first implementation of RAPL, in Sandy Bridge processors, did not measure energy: it estimated it from a model of the processor's activity, which produced systematic errors. Starting with the Haswell generation, the values are based on actual measurements, which improved their accuracy considerably [95]. On a Haswell system, Khan et al. [82] found a correlation of 0.99 between the package power reported by RAPL and the power measured at the wall, and showed that the total power of a server can be predicted from RAPL readings with a mean error of 1.7%. They also found the cost of reading the counters to be negligible, with a performance overhead below 1%. The memory domain is less reliable: by instrumenting the hardware, Desrochers et al. [83] observed deviations of up to 20% between the reported and the actual DRAM power [82].

Limitations. Three limitations must be considered when using RAPL:

- **It only covers part of the machine.** The counters include the processor and its memory, but not the fans, storage, network interfaces or the losses of the power supply. Jay et al. [86] show that an external power meter always reports more than tools based on RAPL, and that the difference is not constant, growing with the load of the machine.

- **It measures components, not software.** The counters report the consumption of an entire domain, and cannot attribute it to a process. They also have no timestamps and are not all updated at the same instant, which limits the precision of measurements over very short intervals [82].

- **The counters overflow.** The counters are held in 32-bit registers and restart from zero when they reach their maximum. Since they count in energy units that depend on the processor model, of 15.3 µJ in Sandy Bridge processors and 61 µJ in Haswell and Skylake processors, the time until an overflow depends on both the model and its power [82], [96]: about 13 minutes for a Sandy Bridge processor drawing 80 W [96], and 52 minutes for a Haswell processor drawing 84 W [82]. The processor does not signal when this happens, so software must detect it and correct the difference. According to Raffin and Trystram [93], this is a frequent source of bugs in measurement tools, and one that is hard to notice, because how often it occurs depends on the power drawn by the machine. An error of this kind was found and corrected in this work, as described in Section 5.6.1.

Several libraries make RAPL easier to use from different programming languages, including RAPLCap for C [97], pyRAPL for Python [98], jRAPL for Java [99], and RAPL Energy for Rust [100]. Most of the higher-level tools described below also rely on RAPL for their CPU readings.

2.3.2 NVIDIA SMI and NVML

The NVML is a library provided by NVIDIA for monitoring and managing its GPUs [101]. It gives access to the state of a device, including its utilization, memory usage, temperature and power draw. NVIDIA System Management Interface (NVIDIA SMI) [81] is a command-line tool built on top of this library that provides the same information from a terminal [102]. It targets NVIDIA's data centre and professional GPUs, with more limited support for consumer models. Bindings for the library exist for other languages, such as the official Python package (pyNVML) [103] and a community-maintained wrapper for Rust [104]. Since NVIDIA GPUs offer no other software interface to their power sensors, practically every tool that reports the energy of an NVIDIA GPU does so through NVML.

How it works. The power values come from a sensor on the GPU board, which measures the current drawn by the device [84]. There are two ways of turning them into energy. The first is to read the **power draw** repeatedly, either with NVIDIA SMI or with the corresponding library function, and to add up the samples over time. The second is to read a **cumulative energy counter**, available on the Volta generation and newer, which returns the total energy consumed by the GPU, in millijoules, since its driver was loaded [101]. As with RAPL, the energy of a workload is then the difference between two readings.

Accuracy of the power readings. The most thorough evaluation of these readings was carried out by Yang et al. [84], who compared them against an external power meter on more than 70 GPUs from all 12 generations that support power measurement. Their findings concern two separate sources of error.

The first is the error of the sensor itself. NVIDIA's documentation states that the power draw is accurate to within 5 W, but the authors found the error to be **proportional** to the power, at around ±5%. For a 700 W GPU such as the H100 [28], this corresponds to 35 W. This error differs from one device to another, but is consistent for a given device. The authors therefore conclude that comparing the power of different workloads **on the same GPU** remains reliable, while comparing values obtained from different GPUs does not.

The second source of error is related to time. Yang et al. [84] show that the reported power is not instantaneous: it is updated at a fixed period, and each value is an average over a window whose length depends on the GPU generation. When the window is shorter than the update period, part of the activity of the GPU is never observed. This is the case for the A100 and H100, where each value covers only 25 ms out of every 100 ms, so **only 25% of the runtime is sampled**, and for the GH200, where this share drops to 20%. On most other recent GPUs, including the one Ada Lovelace model tested, the reported value is an average over the previous second, which covers all activity but responds slowly to changes [84].

Together, these effects can distort energy measurements considerably. In the experiments of Yang et al. [84], when the energy of a workload was measured by simply running it once and adding up the power samples, the error varied unpredictably between workloads and reached 70%, with an average of 39%. By following a set of good practices, namely running the workload repeatedly for at least a few seconds, adding random delays between trials and discarding the samples taken while the power is still rising, the authors reduced the error to about 5% in all cases, which is the error of the sensor itself [84].

The energy counter. The cumulative energy counter avoids the problems that arise from sampling, since it does not depend on how often it is read. Wassermann et al. [105] use it as their reference for GPU power, on the grounds that it is continuously integrated and does not suffer from sampling errors. For the same reason, tools such as CodeCarbon read this counter instead of adding up power samples [106]. Two caveats apply. First, the counter is computed from the same sensor, so it is still subject to its proportional error of about 5%. Second, how the GPU accumulates this counter internally is not publicly documented, and no study was found that validates it against an external power meter, as Yang et al. [84] did for the power readings.

Despite these limitations, NVML remains the only practical way of measuring the energy of an NVIDIA GPU without additional hardware, and is the basis of most research on the energy consumption of GPU workloads.

2.3.3 AMD SMI

AMD SMI is AMD's equivalent of NVIDIA SMI, described as a "unified tool for managing and monitoring GPUs" that allows applications to "control GPU operations, monitor performance, and retrieve information about the system" [107]. It is the successor of ROCm System Management Interface (ROCm SMI), and is available as a command-line tool and as a library for C++, Python and Go. It supports Linux, both on physical machines and inside virtual machines.

In terms of functionality, AMD's tools provide roughly the same information as NVML, including the current power draw of the GPU [85]. They have, however, received far less scrutiny. NVIDIA SMI has been available for more than a decade [108], [109] and has been evaluated in detail, while no comparable study of the accuracy of AMD's power readings was found. The available evidence also suggests that the tools are less mature. While developing a library for collecting hardware metrics from different vendors, Breyer et al. [85] found that the function of the ROCm SMI library that should return the cumulative energy counter of the GPU returned a constant value instead. The energy of AMD GPUs therefore had to be obtained by adding up power samples, the approach whose weaknesses were described above for NVIDIA.

2.3.4 Scaphandre

Scaphandre is an open-source monitoring agent dedicated to energy consumption. Released in late 2020 by tech workers and inspired by "a deep sense of duty" regarding the present environmental challenges faced by the planet, it aims to provide an "easy, lightweight, robust and well understood way to precisely measure energy consumption" [110].

How it works. Scaphandre obtains the energy of the processor from Intel RAPL [111], and its distinguishing feature is that it divides this energy between the processes running on the machine. The share attributed to each process is proportional to the CPU time it used during the measurement interval [112]. This allows the consumption of a single application, container or virtual machine to be estimated, which the underlying counters cannot do by themselves. The results can be exported to monitoring systems such as Prometheus[7], which makes Scaphandre well suited to continuously monitoring services in production.

7 https://prometheus.io/

Kp et al. [113], for instance, use it to attribute energy to the components of a distributed system running on Kubernetes, and Serrano-Gutierrez and Ayala [114] use it in the context of serverless functions.

Accuracy. Jay et al. [86] compared Scaphandre with other software power meters and with an external power meter. Its power profiles follow those of the external meter closely, with a correlation of 0.947. Its energy overhead was 0.85% on average, which is small but among the highest of the tools compared, and its CPU overhead reached about 6% when sampling once per second, since Scaphandre performs all of its calculations on the machine being measured.

Limitations. Scaphandre is designed for monitoring, not for precise measurements of short workloads: it samples at a low frequency, once every two seconds by default [86]. Its per-process values are estimates based on CPU time, not measurements, and it inherits the limitations of RAPL. Most importantly for LLM workloads, it does not measure the consumption of GPUs.

2.3.5 EnergiBridge

Created by researchers at the Delft University of Technology, EnergiBridge aims to "bridge" the gap between energy-conscious software development and the diversity of environments in which software runs [87]. Most energy measurement tools only work on one operating system or one brand of processor, which makes it difficult to use the same method everywhere. EnergiBridge addresses this by offering a single command-line tool, with the same interface and output format, on Linux, Windows and macOS, and on Intel, AMD and Apple processors.

How it works. EnergiBridge is used by placing it in front of the command to be measured. While the command runs, it records the energy and resource usage of the system at a fixed interval, and writes them to a file in a uniform format. It is written in Rust and reads the data directly from the system, without depending on other tools. On Intel and AMD processors, it reads the energy counters from the processor's registers, the same source used by RAPL, while on Apple computers it reads the power reported by the system's sensors. It also collects the usage and power of NVIDIA GPUs. The tool is explicitly aimed at education and research, where ease of use across the different machines of students and researchers matters more than access to every low-level detail.

Limitations. The data that EnergiBridge provides depends on the platform. On Apple computers with ARM processors, for example, only the power of the whole system is available, while on other systems the processor is reported separately. Its accuracy is therefore that of the underlying source on each platform, and its authors do not compare its results against external measurements. Like the other tools based on

the counters of the processor, it reports the consumption of the whole machine, and not of the measured command alone.

2.3.6 CodeCarbon

CodeCarbon is an open-source Python package that estimates the carbon footprint of running a program, by measuring the energy used by the hardware and combining it with the carbon intensity of the electricity consumed [106]. It is easy to add to existing code and is one of the most widely used tools for reporting the emissions of ML experiments.

How it works. CodeCarbon periodically records the consumption of the GPU, the CPU and the memory, every 15 seconds by default [106]. How each value is obtained depends on what the system offers:

- For NVIDIA GPUs, it uses the NVML energy counter, through the official Python bindings. Other GPUs are not measured [88].
- For the CPU, it reads Intel RAPL on Linux, and uses equivalent system interfaces on Windows and macOS. Older versions relied on the Intel Power Gadget, which has since been discontinued [115].
- When no such interface is available, it falls back to an **estimate**. It looks up the Thermal Design Power (TDP) of the processor in a list of more than 2,000 models and assumes that the processor draws 50% of that value on average.
- The power of the memory is also estimated, based on the number of memory modules installed.

The resulting energy is converted into emissions using the carbon intensity of the electricity of the cloud provider or country where the program runs. When this is unknown, a world average is used.

Accuracy. The cost of running CodeCarbon is small. Bouza et al. [88] measured the additional energy caused by several trackers with a power meter, and found it to be around 1% of the total for CodeCarbon, a result consistent with that of Jay et al. [86]. The accuracy of the values it reports is more questionable. Jay et al. observed that, on CPU benchmarks, CodeCarbon reported almost twice the power of other tools that read the same RAPL interface. Bouza et al. found its results to be closer to those of a power meter than those of CarbonTracker, although still below them. More recently, Fischer [89] compared CodeCarbon against external measurements over hundreds of experiments, and found that its estimates follow the general pattern of the real consumption but deviate from it by up to 40%.

Limitations. CodeCarbon is convenient for obtaining an approximate carbon footprint with little effort, but it is not a precise measurement instrument. Its long sampling interval and its fallback to estimates based on the TDP mean that the same code can produce quite different values on different machines, depending on which sources are available. Studies that need precise measurements should use the underlying interfaces directly, or validate its results against them.

2.3.7 CarbonTracker

CarbonTracker is an open-source Python tool for tracking and predicting the energy consumption and carbon footprint of training deep learning models [90]. It was created to make practitioners aware of the footprint of their experiments, in response to the rapid growth of the compute used in ML, and to encourage the reporting of energy alongside model performance.

How it works. CarbonTracker is added to a training loop and measures the consumption of each epoch. Like CodeCarbon, it reads Intel RAPL for the CPU and NVML for NVIDIA GPUs [88], and it collects these measurements, together with the carbon intensity of the local electricity, in separate threads, so as not to disturb the training. Its distinguishing feature is **prediction**: after monitoring one or a few epochs, it predicts the total duration, energy and emissions of the full training, so that an expensive experiment can be reconsidered before it is run to completion.

Accuracy. In the experiments of its authors, the energy predicted after monitoring a single epoch was within 4.9% to 19.1% of the energy measured over the full training [90]. It should be noted that this is the error of the prediction relative to the tool's own measurements, and not the error of those measurements relative to an external power meter. The predicted emissions depend further on the quality of the forecasted carbon intensity. Jay et al. [86] measured an energy overhead of 0.76% for the tool.

Limitations. CarbonTracker is built around the concept of training epochs, and offers no equivalent for inference. This makes it unsuitable for studying the consumption of deployed models, which is the focus of this work.

2.3.8 EnviroLLM

Developed by Troy Allen at the Georgia Institute of Technology, EnviroLLM is an open-source toolkit for measuring, benchmarking and optimizing the performance and energy consumption of LLMs running on personal devices [91]. Most existing tools were designed for training, or for inference at the scale of data centres. EnviroLLM is one of the few that focus specifically on the local execution of LLMs, which makes it the tool whose goals are closest to those of this work.

How it works. EnviroLLM detects the processes of common local LLM platforms, such as Ollama and LM Studio, and monitors their resource usage while a model answers a set of prompts. It records the usage of the CPU and memory every two seconds, and reads the power draw of NVIDIA GPUs through NVML. When the power of the GPU is not available, it is estimated from a baseline consumption of the system and the utilization of the CPU and GPU. The energy of an inference is obtained by adding up these power values over its duration. Besides energy and speed, the tool uses another LLM to judge the quality of the responses, so that users can compare models and quantization levels in terms of both quality and efficiency. The results are stored in a local database and displayed in a web dashboard.

Limitations. EnviroLLM is aimed at helping users choose a model for their own machine, and not at producing precise measurements. Its energy values come from power samples taken every two seconds, which is the approach that Yang et al. [84] showed to be the least reliable, and from estimates when no sensor is available. The CPU is monitored through its utilization, and not through RAPL. As its author acknowledges, the tool was only tested on a single hardware configuration, and its values were not compared against external measurements [91].

2.4 Designing Reliable Energy Experiments

The tools described in the previous section report how much energy a machine consumed during an interval, but a reliable comparison between two pieces of software requires more than a tool. Even on the same machine, repeated executions of the same program produce different energy values, because the measurements are affected by factors that have nothing to do with the software under study: the temperature of the hardware and of the room, the tasks running in the background, or the decisions of the scheduler [12]. Ournani et al. [116] quantified the problem with more than 1,000 experiments on over 100 machines, and showed that controllable factors, such as the experimental protocol and the power management features of the processor, can change the variation between identical runs by up to 30 times. On mobile devices, Bokhari et al. [117] observed that the state of the system keeps changing between successive experiments despite reasonable efforts to keep it uniform, to the point of producing misleading comparisons between variants of a program.

Cruz [12] summarizes the practices used in Green Software research to deal with this variability, in a guide to setting up energy efficiency experiments. They are listed in Table 3.

Practice	What it consists of	What it prevents
Minimize background activity	Closing applications, services and hardware that are not needed	Energy from other tasks being counted as that of the software under study

Practice	What it consists of	What it prevents
Freeze settings	Fixing, and reporting, the configuration of what cannot be turned off	Differences between runs caused by settings that change
Warm up	Running a dummy task before measuring, or discarding the first executions	The first measurements being taken on a cooler machine
Repeat	Repeating each measurement, ideally 30 times	Conclusions drawn from a single, possibly atypical, execution
Rest	Pausing between executions	The heat accumulated in one execution affecting the next
Shuffle	Executing the alternatives being compared in random order	Changes in the machine over time favouring one of the alternatives
Control the temperature	Keeping the room temperature stable, or recording it and discarding the runs that deviate	Differences between day and night in long experiments
Automate	Scripting every execution	Differences between repetitions introduced by manual steps

Table 3: Practices for reducing bias in energy measurements. (Adapted from [12])

Order of execution. If all the repetitions of one alternative are executed before those of another, any change in the state of the machine between the two groups is confused with the difference between the alternatives. Cruz recommends choosing each execution at random [12], and Bokhari et al. propose rotating the alternatives in a round-robin order, with the same purpose [117].

Number of repetitions. Cruz suggests 30 repetitions, the sample size generally accepted for statistical testing [12]. Studies of LLMs, in which a single evaluation can take hours [118], usually repeat their measurements fewer times. Among the studies reviewed in Section 2.7, Jin et al. repeat each evaluation 10 times [118], while Fernandez et al. and Alizadeh et al. repeat theirs 3 times [71], [119].

Idle consumption. A machine consumes energy even when it is idle, and this *static* consumption is part of every measurement. Subtracting it gives the *dynamic* energy of the software, defined as the total energy minus the idle power of the machine multiplied by the execution time [120]. Measuring the idle power of the machine is therefore part of an energy experiment, since it shows how much of the measured energy is actually caused by the workload.

Aspects specific to LLM inference. Measuring the inference of LLMs adds difficulties of its own. The inference engine has to be warmed up before the first measurement: Fernandez et al. [71] run up to 20 batches beforehand, to allow for memory allocation and for the compilation steps that take place during the first executions. When requests are processed in batches, the energy of an individual request cannot be observed directly, and has to be derived from that of the batch [73]. Finally, the power readings of a GPU have limitations of their own, described in Section 2.3.2.

Automation. Applying these practices by hand to hundreds of runs is impractical and prone to error, which motivated the development of frameworks that orchestrate experiments automatically. Experiment Runner [121] is a Python framework for the orchestration of measurement-based experiments. The experiment is described in code, in terms of its factors and of the number of repetitions, and the framework executes the runs, optionally in random order and with a cool-down period between them, records their data, and can pause and resume the experiment without losing results. According to its authors, it has been used in more than 50 empirical studies. It is the framework adopted in this work, as described in Section 4.3.

2.5 Local Execution of LLMs

The rapid acceleration of the LLM landscape and its integration across multiple industries has driven growing interest in deploying these models locally, primarily motivated by economic and privacy advantages [19], [20]. Multiple open models have since emerged, allowing anyone with the necessary hardware to operate their own local and private large language model. Some of the most popular and powerful open LLM families include:

- Llama, by Meta [23]
- Gemma, by Google [22]
- Phi, by Microsoft [122]
- Mistral, by Mistral AI [123]
- DeepSeek, by DeepSeek AI [76]
- Qwen, by Alibaba Cloud [77]

Running LLMs locally presents significant challenges due to their high computational and memory demands. To address this issue and enable the execution of LLMs on consumer-grade hardware, researchers adapted the long-standing concept of *quantization*. Broadly defined, quantization is the process of mapping a large set of input values to a smaller, discrete set of output levels [124]. When applied to an LLM, this technique reduces the precision of the model's numerical parameters, typically converting them from 32-bit floating point to lower bit-width representations such as 8-bit integers [125]. This reduction significantly

decreases memory usage and computational load while aiming to preserve model accuracy, making deployment on resource-constrained devices feasible [125].

Quantization is implemented through various file formats and frameworks, each optimized for specific hardware or use cases. Among the most widely adopted are GGUF and EXL2. GGUF, developed by the llama.cpp [126] team, is a flexible and portable format designed for efficient inference on CPUs and GPUs, supporting multiple quantization levels and enabling cross-platform deployment. EXL2, on the other hand, is the format of ExLlamaV2, a library for running LLMs on consumer GPUs [127]. It is based on the same optimization method as GPTQ and supports 2- to 8-bit quantization, mixing several levels within a model: the precision of each layer is chosen to minimize the quantization error for a target average number of bits per weight [127].

2.5.1 Obtaining Quantized Models

In practice, users rarely need to perform quantization themselves, since pre-quantized versions of many mainstream models are published and available for direct download. Platforms such as the Hugging Face Hub[8] host a vast number of quantized variants of popular open models, in formats including GGUF, AWQ and FP8, and such variants are frequently maintained by the community, while local execution platforms such as Ollama [128] and LM Studio[9] curate their own catalogues of quantized models. In these cases, users can deploy a quantized version without ever performing quantization themselves.

When a pre-quantized version is unavailable, or when full control over the process is desired, quantization can instead be performed locally. This is supported by dedicated compression libraries implementing different quantization methods, such as llm-compressor [129] and AutoAWQ [130], as well as by the conversion tooling distributed with llama.cpp [126] for producing GGUF files. Some serving frameworks further support online quantization, compressing a model at load time rather than ahead of time, as in the case of vLLM [27], [131]. Because so many methods exist, each targeting a specific precision, commonly INT8, FP8 and INT4, models quantized to the "same" precision can still behave differently, an additional complication for model comparison that is elaborated in Chapter 3.

2.5.2 Dense vs. Mixture-of-Experts Architectures

Where quantization controls a model's precision, its architecture determines how its parameters are used. Most open LLM families adopt a conventional *dense* architecture, in which every parameter of the model

8 https://huggingface.co/

9 https://lmstudio.ai/

is activated for each processed token. As a direct consequence, the computational cost of processing a token scales with the model's total number of parameters, so serving larger dense models demands proportionally more compute power. The *Mixture-of-Experts* (MoE) paradigm addresses this scaling problem by structurally decoupling the total number of parameters from the compute required per token [132], [133]. Instead of using one large layer shared by every token, MoE models employ a collection of specialized "expert" networks, and a routing mechanism activates only a sparse subset of them for each token, leaving the remaining experts idle [132], [134]. The compute required per token therefore scales with the number of active parameters rather than with the total parameter count, allowing substantially larger models to be served with only a modest increase in computational cost [133], [134].

This scaling advantage has made MoE architectures increasingly common [29], both among frontier proprietary systems, the largest of which are widely believed to rely on MoE layers [64], and within open families such as Mistral, DeepSeek, Qwen or Phi, which have released multiple MoE variants [134]. From the perspective of local execution, however, the architecture carries an important caveat: even though only a fraction of the experts is active for a given token, all of them must still be loaded into memory. An MoE model therefore exhibits a memory footprint proportional to its total parameter count, while its per-token computational cost resembles that of a much smaller dense model. As further discussed in Chapter 3, this distinction complicates the fair comparison of models built around different architectural paradigms.

Expert routing is not the only way of decoupling a model's total parameter count from the computation it performs. Google's Gemma 3n models, and the smaller models of the Gemma 4 release, are identified by an *effective* parameter count, indicated by an "E" in their names (for instance, "E2B" and "E4B"), which is lower than their total parameter count [135], [136]. These models use *Per-Layer Embeddings*, which give each layer of the model its own embedding for every token. The resulting embedding tables are large, but they are only used for lookups, which is why the effective parameter count is much smaller than the total [136]. The Gemma 4 E2B model, for example, has 2.3 billion effective parameters out of a total of 5.1 billion. In Gemma 3n, the parameters used to process image and audio inputs can additionally be left out when they are not needed [135]. Like MoE models, they behave computationally as a smaller model than the one that must be stored, even though they do not contain expert networks. In this document, models of both kinds are referred to as *sparsely activated*.

Once the architecture and precision of a model are settled, an execution framework, also called an *inference engine*, is still needed to run it. The engine loads the weights of the model, manages its memory, schedules the requests it receives and runs the computation on the available hardware, so its design has a large effect on both speed and energy consumption. Many such engines exist: a recent survey by Park et al. [137] examines 25 of them. They range from high-level tools that make local models easy to use,

to highly optimized systems designed to serve many users with maximum performance. The four engines explored below are among the most widely used for running open models, and cover this entire range. Table 4 compares their main characteristics.

Engine	Main goal	Models	Hardware	Scale
Llama.cpp	Portability and minimal setup	GGUF files, usually quantized	CPUs and GPUs of all major vendors, including hybrid execution	Single machine, few simultaneous users
Ollama	Ease of use	Library of ready-to-use models, based on GGUF	Same as llama.cpp, its backend	Single user, single GPU
vLLM	High-throughput serving	Original models from the Hugging Face Hub	NVIDIA, AMD and Intel GPUs	Many simultaneous users, several GPUs
TensorRT-LLM	Maximum performance on NVIDIA hardware	Models converted for the target GPU	NVIDIA GPUs only	Many simultaneous users, several GPUs and machines

Table 4: Comparison of the inference engines explored in this section.

2.5.3 Llama.cpp

Llama.cpp [126] is one of the foundations of the local LLM ecosystem. Created by Georgi Gerganov, it is a library written in plain C/C++ with no external dependencies, whose stated goal is to enable LLM inference "with minimal setup and state-of-the-art performance on a wide range of hardware". It was originally designed to run models on a CPU, without requiring a GPU [137], and treats Apple silicon as a first-class target. This focus on portability is what distinguishes it from the other engines: the same code runs on laptops, desktops, servers and embedded devices.

Hardware support. Over time, llama.cpp has gained backends for most accelerators, including NVIDIA GPUs through CUDA, AMD GPUs through HIP, Apple silicon through Metal, Intel GPUs through SYCL, and other devices through Vulkan [126]. On x86 and ARM processors, it uses the vector instructions available on each one. It also supports *hybrid* inference, in which some layers of a model run on the GPU and the rest on the CPU. This makes it possible to run models that do not fit in the available Video RAM (VRAM), at the cost of lower speed.

Model format and quantization. Llama.cpp introduced the GGUF format, which stores the parameters, structure and metadata of a model in a single file, replacing its earlier GGML format [137]. The format supports integer quantization at several levels, from 1.5 to 8 bits per parameter, and it is in quantized form that most GGUF models are distributed. Together with hybrid inference, this is what allows

large models to run on consumer hardware with limited memory. Besides the library, the project includes a command-line program and a server compatible with the OpenAI API.

Limitations. Llama.cpp is optimized for running on a single machine and for one or a few simultaneous users. It offers continuous batching and multithreading, but has no native support for clusters [137]. In a benchmark of inference engines on several accelerators, Chitty-Venkata et al. [138] observed that it obtains only marginal gains from additional GPUs, because it cannot fully exploit their parallelism. For a measurement study, its typical mode of use also introduces the two complications discussed in Chapter 3: models are usually quantized, and hybrid inference spreads the work between the CPU and the GPU.

2.5.4 Ollama

Ollama [128] is a tool designed to make running LLMs locally as simple as possible, for end users and developers without a technical background in the area. It hides the handling of model files, configuration and hardware behind a single program, with an interface similar to that of Docker for containers: a model is downloaded and started with a single command.

How it works. Ollama is written in Go and consists of a client and a server [137]. The server runs in the background, loads the models and executes the requests, using llama.cpp as its inference backend. The client is a command-line program that sends requests to the server. The same server exposes a REST API, on port 11434 by default, as well as an API compatible with that of OpenAI, so that local models can be used from other applications [128]. Models are obtained from a central library maintained by the project, in a way similar to a package registry. They can also be imported from GGUF or Safetensors files and customized through a *Modelfile*, which defines the base model, its parameters and its system prompt [137]. Ollama is available for Linux, macOS and Windows, detects the available hardware automatically, and supports NVIDIA and AMD GPUs as well as ARM platforms.

Limitations. This accessibility comes at a price. According to Park et al. [137], Ollama prioritizes ease of use over advanced inference optimizations: it targets a single GPU, lacks the batching strategies needed for sustained throughput, and cannot be extended to several machines. The decisions it makes automatically, such as which variant of a model to download, typically a quantized one, and how to configure its execution, are convenient for everyday use, but leave less control over the conditions of an experiment than the engine it is built on.

2.5.5 vLLM

vLLM [27], [131] is an engine designed for serving LLMs to many users with high throughput. It was originally developed at the Sky Computing Lab of UC Berkeley, and has since become a large open-source project with contributions from both academia and industry.

PagedAttention. The starting point of vLLM is the observation that the throughput of a server is limited by how many requests it can process together, and that this number is limited by memory. During generation, the model stores intermediate results for every token of every request in the KV cache, which grows and shrinks as requests progress. Earlier systems reserved a contiguous region of memory for each request, sized for the longest possible output, and much of this memory was wasted: Kwon et al. [27] measured that only 20% to 38% of it held useful data. Their solution, *PagedAttention*, is inspired by the virtual memory of operating systems. It divides the KV cache of each request into blocks that hold a fixed number of tokens and do not need to be stored contiguously, so memory is allocated only as it is needed, and blocks can be shared between requests. With memory used almost without waste, more requests fit in a batch, and the authors report throughputs 2 to 4 times higher than those of the previous state-of-the-art systems at the same latency.

Features. vLLM has since accumulated a broad set of optimizations [131]. It processes incoming requests with *continuous batching*, adding them to the running batch as they arrive. It can reuse the computation of prompt prefixes shared between requests (*prefix caching*), split long prompts into pieces so they do not delay other requests (*chunked prefill*), and execute the model through optimized CUDA graphs. It supports several quantization formats and loads models directly from the Hugging Face Hub in their original format. For larger models, it distributes the work over several GPUs. It runs on NVIDIA, AMD and Intel GPUs, and can be used either as a server compatible with the OpenAI API or as a Python library, for processing prompts directly from a program.

Architecture. Internally, vLLM separates the component that receives requests from the one that executes them. The latter, called the engine core, runs in its own process and is responsible for scheduling and for running the model, while tasks that use the CPU, such as tokenization, are handled outside it [137].

Position among the engines. Park et al. [137] rate vLLM highly for generality and ease of use, and Chitty-Venkata et al. [138] highlight its flexibility, since it can be accelerated on a wider range of hardware than vendor-specific engines. Most of its optimizations can also be configured or disabled individually, and the engine reports detailed metrics about each request. This combination of performance, control and transparency has made it a common choice for research as well as for production.

2.5.6 TensorRT-LLM

TensorRT-LLM [139] is an open-source library developed by NVIDIA to obtain the best possible inference performance from LLMs on its own GPUs, such as the H100 [28] and the A100 [58].

How it works. The library builds on TensorRT [140], NVIDIA's compiler for deep learning models. Instead of executing a model as it was published, the model is first converted into a form that the compiler can optimize. The compiler analyses the computation of the model, selects the most efficient implementation of each operation for the target GPU, and merges consecutive operations to reduce memory traffic [137]. The result takes full advantage of hardware features such as Tensor Cores and low-precision arithmetic, at the cost of an additional preparation step for every model.

Features. TensorRT-LLM includes the optimizations expected from a serving engine [139]. *In-flight batching*, NVIDIA's name for continuous batching, groups incoming requests dynamically, and the KV cache is managed in blocks, as in vLLM. It supports quantization formats tied to specific hardware generations, such as FP8 on the H100 and newer. Large models can be distributed over several GPUs and several machines. For deployment, it integrates with NVIDIA's Triton Inference Server, and it provides a Python API for defining and running models.

Position among the engines. The specialization of TensorRT-LLM is both its strength and its limitation. In the benchmark of Chitty-Venkata et al. [138], it provided the highest performance and the lowest power consumption on NVIDIA hardware, in line with their general observation that vendor-specific engines achieve the best throughput on each accelerator. On the other hand, it only runs on NVIDIA GPUs [137], and the conversion step makes it less convenient when many different models have to be evaluated.

Taken together, these four engines illustrate a trade-off that runs through the entire ecosystem. Engines that are easy to use and run everywhere give up performance and control, while the engines that extract the most from the hardware are tied to it and require more preparation. Since the engine determines how efficiently a model uses the hardware, it also influences the energy consumed by an inference, which makes its selection an important methodological decision, discussed in Section 3.5.

2.6 Evaluating the Performance of LLMs

A core objective of this thesis is to determine how the relationship between energy consumption and model accuracy evolves as new versions of state-of-the-art models are released. Measuring accuracy requires LLM benchmarks: collections of questions or tasks that assess the performance of a model in one or more domains.

The performance of an LLM can vary significantly depending on the task it is given. For example, a code-oriented model may excel in coding tasks but perform poorly on general knowledge tasks. This research aims to evaluate general-purpose LLM families, so selecting a comprehensive suite of benchmarks is fundamental to assess their performance across multiple domains and gain insight into their general intelligence.

Table 5 presents a selection of benchmarks commonly used to evaluate LLMs, along with the domains they assess.

Benchmark	Domain	Description
HLE [6]	Academic Reasoning	2,500 expert-written questions in over a hundred subjects, designed as a final closed-ended academic evaluation.
GPQA Diamond [141]	Scientific Reasoning	198 very hard multiple-choice questions in biology, chemistry, and physics, written and validated by domain experts.
MMLU [142]	General Knowledge	A foundational benchmark of 15,908 multiple-choice questions across 57 subjects spanning STEM, humanities, social sciences, and professional fields.
MMLU-Pro [7]	Reasoning & General Knowledge	12,032 questions in 14 domains, extending MMLU with reasoning-heavy questions and ten options instead of four.
MATH [143]	Advanced Mathematics	12,500 challenging problems from high school mathematics competitions, across seven subjects and five difficulty levels.
AIME 2025 [144]	Competition Mathematics	30 competition problems (15 from AIME I and 15 from AIME II), each answered with an integer from 0 to 999.
HumanEval [145]	Coding (Python)	A classic coding benchmark containing 164 handwritten programming problems, verified with unit tests.
MBPP [8]	Coding (Python)	974 crowd-sourced, entry-level programming problems.
LiveCodeBench [13]	Coding	Coding problems from LeetCode [146], AtCoder [147] and CodeForces [148], regularly updated to avoid data contamination.
LiveCodeBench Pro [9]	Coding	584 hard competitive programming problems from CodeForces [148], ICPC [149], and IOI [150], annotated by Olympiad medalists and regularly updated to avoid data contamination.
SWE-bench Verified [10], [151]	Agentic Coding	500 human-validated GitHub issues from 12 Python repositories, which must be resolved by editing the code.

Benchmark	Domain	Description
FACTS Benchmark Suite [152]	Factuality	Factual accuracy in four scenarios: grounding in documents, image-based questions, closed-book questions, and search.
Global PIQA [153]	General Cultural Knowledge	Physical commonsense questions written by native speakers in 116 language varieties.
SuperGLUE [154]	Language Understanding	Eight language understanding tasks, designed as a more challenging successor to GLUE.

Table 5: Overview of key LLM evaluation benchmarks across domains.

While useful for evaluating general-purpose model capabilities, these benchmarks cover many domains at once and lack specificity. As Anjum et al. [155] point out, general-purpose models frequently falter on tasks that require deep, domain-specific knowledge. Even a leading model like GPT-4, for instance, achieves low accuracy on specialized financial questions. Domain-specific benchmarks are therefore crucial for diagnosing such failures, and a wide range of them has been developed across disciplines including Engineering, Science, Technology, Mathematics, Humanities, Finance, Healthcare and Language Understanding, which are explored in detail in [155].

Despite their importance, conducting hundreds of specialized benchmarks is not a feasible approach for this work. Higher-level benchmarks, such as those in Table 5, still offer valuable insights into the capabilities of LLMs. Although their evaluations may be less detailed, they effectively illustrate a model's performance across multiple domains.

Each of these benchmarks is outlined below: what it measures, how it was built, the scores that models currently obtain, and whether it suits the smaller, local models studied in this research.

2.6.1 Humanity's Last Exam (HLE)

With the rapid advancement of LLMs in recent years, many popular benchmarks have been saturated by the latest state-of-the-art systems, with accuracy scores surpassing 90% on datasets such as MMLU. As the differences between models shrink, it becomes difficult to measure further progress, which creates a need for more challenging evaluations [6].

To address this gap, HLE [6] introduces a dataset of 2,500 "extremely challenging" questions, written by nearly 1,000 subject experts from over 500 institutions. The questions are closed-ended, so answers can be verified automatically: 24% are multiple-choice and the rest require an exact answer, and around 14% include an image. As illustrated in Figure 8, models that achieve high scores on older benchmarks

demonstrate significantly lower accuracy when evaluated with Humanity's Last Exam (HLE). At its release, the accuracy of the frontier models tested by Phan et al. ranged from 2.7% to 13.4%.

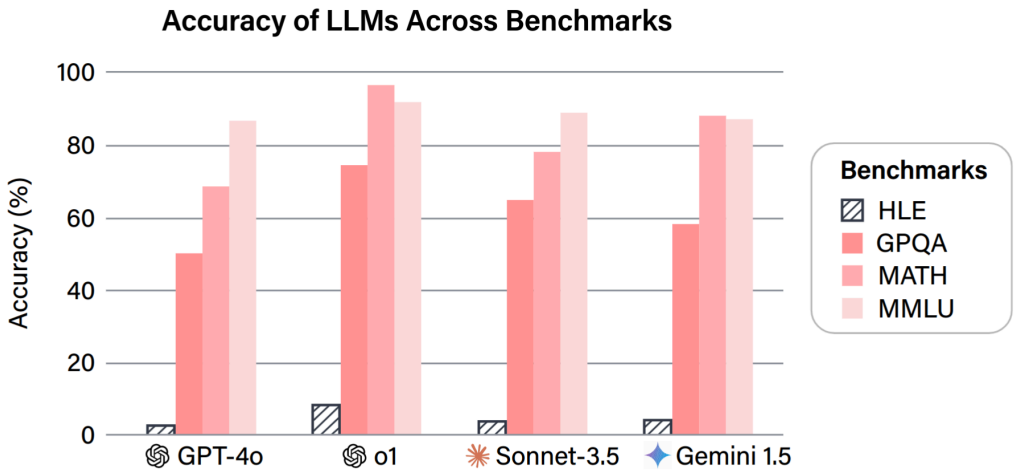


Figure 8: Accuracy of frontier models on Humanity's Last Exam compared with earlier benchmarks. (Reprinted from [6])

The questions span over a hundred subjects, which are grouped into eight high-level categories: Math, Biology/Medicine, Computer Science/Artificial Intelligence, Physics, Humanities/Social Sciences, Chemistry, Engineering, and 'Other'. The distribution of questions across these categories is illustrated in Figure 9.

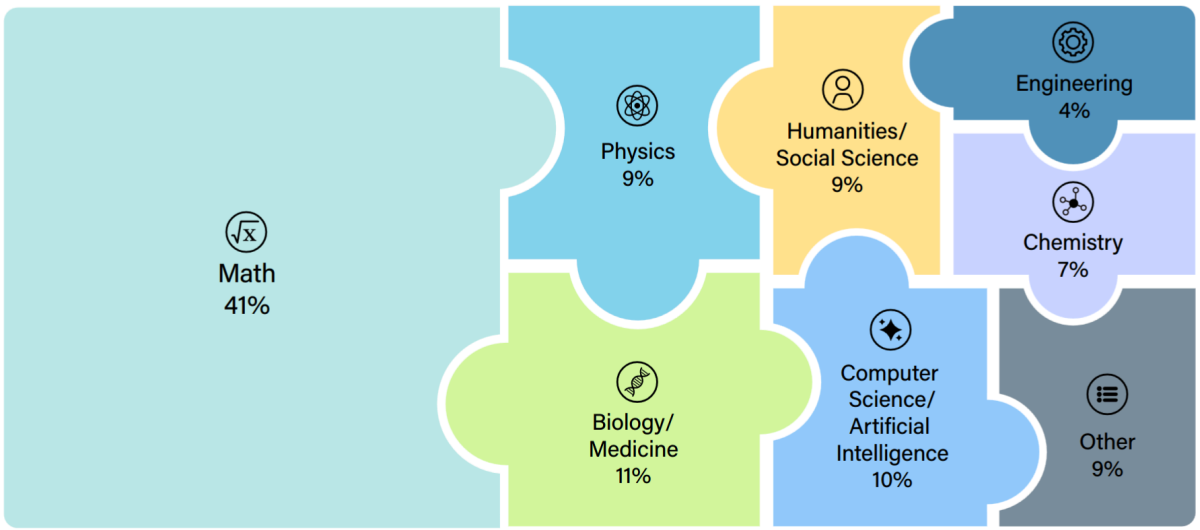


Figure 9: Distribution of HLE's questions across domain categories. (Reprinted from [6])

Scores have risen since then, but the benchmark is still far from solved: Google reports 44.4% for Gemini 3.1 Pro without the use of tools, the highest result among the frontier models it compares in that setting [156]. Designed for evaluating full-sized, state-of-the-art models, HLE may not be the best fit for smaller

local models capable of running on consumer hardware. Their accuracy on this highly complex benchmark would likely be too low to identify any meaningful performance differences.

2.6.2 GPQA Diamond

The rapid improvement of LLMs in recent years raises the possibility of developing superhuman AI models in the near future, systems whose knowledge and capabilities exceed those of even the best human experts. This calls into question whether humans would still be able to supervise, evaluate, and control such systems, a challenge referred to as *scalable oversight* [157].

Several scalable oversight techniques have been proposed, but testing whether they will still work once AI systems surpass human capabilities is difficult, since those systems do not exist yet. A potential solution is *sandwiching* [158], which reproduces that future scenario with current models. It relies on a model that outperforms non-experts in a specific domain, while human experts can still surpass it, so that the model is "sandwiched" between the two groups. The non-expert plays the role of the future human supervisor, and the model plays the role of the superhuman AI: the non-expert has to reach the correct answer to a question by interacting with a model whose answers they cannot verify on their own, and which may be wrong. The experts do not take part in this process. Their answers serve only as the reference against which the result is checked, which is what makes it possible to tell whether an oversight technique worked.

Graduate-Level Google-Proof Q&A (GPQA) [141] is a dataset developed to support sandwiching experiments. It consists of 448 multiple-choice questions in Physics, Chemistry, and Biology, each with four options and written by experts in the corresponding field. The questions are "Google-proof": highly skilled non-experts, who hold or are pursuing a PhD in another field, reached only 34% accuracy despite spending over 30 minutes per question on average with unrestricted access to the web, while experts in the field reached 65%. The dataset therefore provides what a sandwiching experiment needs: questions that non-experts cannot answer on their own, with correct answers that are known to experts. GPQA Diamond is the highest-quality subset of the dataset: 198 questions on which the expert validators agreed with the answer and the majority of non-experts answered incorrectly.

Given the difficulty of its questions, GPQA Diamond is also commonly used as a benchmark of scientific reasoning. At its release, GPT-4 answered 38.8% of the Diamond questions correctly [141], whereas Google now reports 94.3% for Gemini 3.1 Pro [156], which shows that frontier models are approaching its ceiling. Its main limitation for this work is scope: covering only three sciences, it would need to be paired with other datasets to evaluate a model's general intelligence.

2.6.3 MMLU

The original Massive Multitask Language Understanding (MMLU) [142] benchmark was introduced in 2021 to evaluate the performance of LLMs across a wide range of topics that were, at the time, largely unassessed by existing NLP benchmarks. Compiling a dataset of questions from various human academic subjects was key to matching the trend of expanding knowledge bases on which models were increasingly being trained.

The 15,908 multiple-choice questions of MMLU, each with four options, cover 57 subjects that range from traditional areas, such as mathematics and history, to specialized fields like law and ethics. The difficulty of the questions spans from an elementary level to an advanced professional level, testing both world knowledge and problem-solving abilities. The benchmark was far from solved when it was introduced: the largest GPT-3 model reached 43.9% accuracy and the best model evaluated reached 48.9%, against an estimated 89.8% for human experts. These characteristics made it ideal for evaluating a model's general intelligence across both common and uncommon domains, quickly establishing it as a leading LLM benchmark.

Although highly effective when introduced, MMLU has since been saturated. Wang et al. [7] note that, after GPT-4 reached 86.4% in March 2023, the frontier models released in early and mid-2024 all settled between 86% and 87%, and that part of its questions are unanswerable or wrongly annotated, which lowers the maximum score a model can reach. Paired with increasing data contamination [159], this once-pivotal dataset has become ineffective for evaluating continuously improving AI models.

2.6.4 MMLU-Pro

MMLU-Pro [7] was introduced in 2024 as an enhanced dataset designed to extend the original, performance-saturated MMLU benchmark. Wang et al. [7] identified three main problems with the original benchmark:

1. With only three wrong options per question, models can exploit shortcuts to find the answer without truly understanding the question. This leads to an overestimate of their real performance and to unstable results.

2. Most questions test knowledge and require little reasoning, especially in the Science, Technology, Engineering and Mathematics (STEM) subjects, which reduces their difficulty. In fact, most models perform better when answering directly than when reasoning step by step with chain-of-thought [74].

3. Part of the questions are unanswerable or wrongly annotated. This noise lowers the maximum score that can be reached, a ceiling which the frontier models hit.

MMLU-Pro sought to address these issues by:

1. Increasing the number of multiple-choice options from 4 to 10, containing 3 times more "distractors" and decreasing the probability of correct guesses by chance.
2. Increasing the share of challenging, college-level exam problems, which require an LLM to reason in different domains in order to obtain a final answer.
3. Implementing two rounds of expert reviews in the dataset creation pipeline, reducing dataset noise.

Questions from the original MMLU dataset were also subjected to a comprehensive review, which merged the previous 57 subject categories into 14 broader categories and eliminated overly simple questions that failed to challenge the models. Figure 10 illustrates the full pipeline involved in the construction of the final MMLU-Pro dataset.

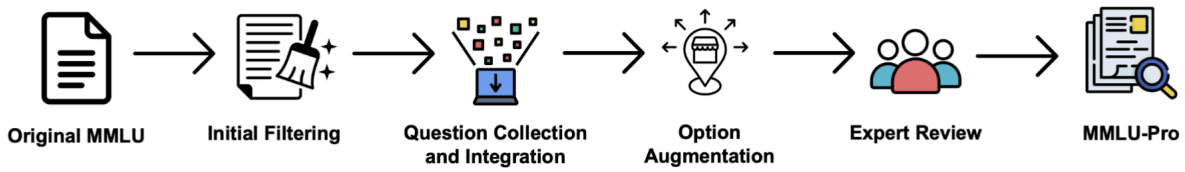


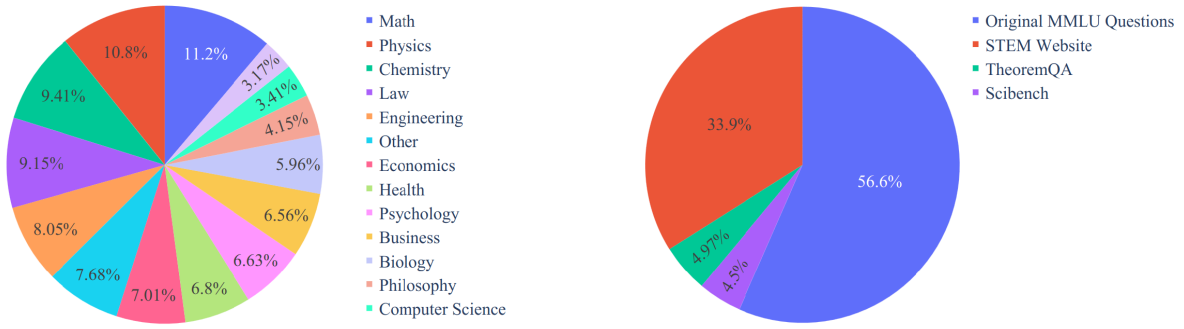
Figure 10: MMLU-Pro dataset construction pipeline. (Reprinted from [7])

Although a majority of the questions in the final dataset are adapted from MMLU, over 40% are newly sourced from the STEM Website[10], TheoremQA [160], and SciBench [161]. These sources provide high-quality STEM problems, human-annotated theorem-application questions, and advanced science problems derived from college exams, respectively. The full distribution of question disciplines and data sources is outlined in Figure 11.

The result is a dataset of 12,032 questions across 14 knowledge domains that is both harder and more reliable than its predecessor [7]. At its release, the best model, GPT-4o, reached 72.6% accuracy, and the gap between GPT-4o and GPT-4-Turbo grew from 1 percentage point on MMLU to 9 on MMLU-Pro, which shows that it separates models more clearly. It is also more stable: across 24 different prompt styles, the scores of a model varied by around 2%, compared to 4% to 5% on MMLU. Finally, in contrast to MMLU, models obtain better results on MMLU-Pro with chain-of-thought than with direct answers, which

10 https://stemez.com/subjects

confirms that its questions require reasoning. For this reason, the authors evaluate models with prompts that contain five examples answered step by step.



(a) Distribution of Disciplines in MMLU-Pro (b) Data Source Distribution in MMLU-Pro

Figure 11: Distribution of disciplines and question sources in the MMLU-Pro dataset. (Reprinted from [7])

MMLU-Pro was quickly adopted as a benchmark for the broad, multi-subject knowledge of LLMs. It is, for instance, one of the six benchmarks of the Open LLM Leaderboard, which evaluates open models [159], [162]. It is also effective at differentiating the smaller, open models that this study investigates: in the original evaluation, the scores of open models ranged from 15.9% for Gemma-2B to 56.2% for Llama-3-70B-Instruct, with models of 7 to 8 billion parameters spread between 23.5% and 41.0% [7].

2.6.5 MATH

The MATH [143] dataset consists of 12,500 problems from high school mathematics competitions. It aims to evaluate the problem-solving abilities of LLMs by assessing their effectiveness at analysing problems, selecting "good heuristics from a large set of possibilities," and combining them to produce a final answer, a process inherent to mathematical problem-solving.

The problems are categorized by difficulty levels from 1 to 5 across seven distinct subject areas. They have proven difficult even for humans: a "computer science PhD student who does not especially like mathematics" scored approximately 40%, while a "three-time International Mathematical Olympiad (IMO) gold medalist" achieved 90%. The models evaluated at the time of its release obtained accuracies between 3.0% and 6.9% [143].

MATH has since been saturated by frontier models. In 2024, OpenAI reported an accuracy of 94.8% for its o1 model and observed that frontier models do so well on MATH that it is no longer effective at differentiating them [75]. For open models, its hardest problems remain in use: the Open LLM Leaderboard keeps only the questions of level 5 [159]. The dataset has also proven highly valuable for training. It was used to generate extensive synthetic datasets, such as NVIDIA's OpenMathInstruct-2 [163], which helped

advance the mathematical reasoning capabilities of open LLMs. Using Supervised Fine-Tuning (SFT) on this dataset, Toshniwal et al. obtained a model based on Llama-3.1-8B that outperformed the official instruction-tuned version of the same model by 15.9 percentage points on MATH (from 51.9% to 67.8%).

2.6.6 AIME 2025

The American Invitational Mathematics Examination (AIME) [164] is a mathematics competition administered by the Mathematical Association of America (MAA) [11]. It is taken by the high scorers of the American Mathematics Contest 10 (AMC 10) [165], American Mathematics Contest 12 (AMC 12) [166], and United States of America Mathematical Talent Search (USAMTS) competitions, and is the second exam in the series used to select the team that represents the United States at the IMO.

The exam consists of 15 questions with a three-hour time limit, where each answer is an integer between 0 and 999, inclusive. Calculators are prohibited. Problems generally increase in difficulty as the exam progresses, and are rated from 3 to 6 on the difficulty scale of the Art of Problem Solving Wiki. In 2025, the exam took place on two distinct dates and therefore had two official versions: AIME I [167] and AIME II [168].

In the LLM benchmarking landscape, the combined set of questions from AIME I and AIME II is typically referred to as AIME 2025, a dataset including all 30 problems [144]. This dataset is used as a benchmark to evaluate the ability of LLMs to solve complex mathematical problems requiring multi-step logical deductions and structured symbolic reasoning.

According to the AIME 2025 leaderboard of LLM Stats [169], which aggregates scores reported by model developers, several frontier models have achieved 100% on the benchmark, including Grok-4 Heavy [170] and Gemini 3 Pro [171]. While the dataset remains valuable for assessing the capabilities of smaller LLMs, it represents a barrier that top state-of-the-art models have effectively overcome. With only 30 problems, it also offers a coarse measurement, since each problem is worth more than 3 percentage points.

2.6.7 HumanEval

Released in mid-2021 by engineers and researchers at OpenAI, HumanEval was one of the first LLM coding benchmarks. It was introduced alongside the landmark paper on Codex [145], a GPT-3-based model fine-tuned on publicly available code from 54 million GitHub repositories, of which a production version powered GitHub Copilot [172]. The benchmark was designed to evaluate the ability of Codex to synthesize functional Python code from *docstrings*, short descriptions embedded within functions explaining their

11 https://artofproblemsolving.com/wiki/index.php?title=Mathematical_Association_of_America

logic and usage. Crucially, HumanEval popularized the pass@k metric for functional correctness, which measures the probability that at least one of k generated code samples passes all accompanying unit tests [173]. A problem therefore counts as solved if any of the samples is correct, although in practice the correct sample would still have to be identified. Ribeiro et al. [174] showed that the hidden states of a model encode whether the code it generated is correct, more reliably than the probability it assigns to that code or the confidence it states, and that ranking the samples by this signal, without running any test, brings the pass@1 of a model on HumanEval close to its pass@10.

The original dataset consists of 164 handwritten Python programming problems paired with unit tests (averaging 7.7 tests per problem) that assess domains such as language comprehension, algorithms, and simple mathematics. Upon release, HumanEval quickly became a standard benchmark for code models. To address its restriction to Python, researchers later introduced HumanEval-X [175], an extension that translated the original 164 problems into four additional languages: C++, Java, JavaScript, and Go.

However, HumanEval showed its limits early on. Liu et al. [176] found that its tests are insufficient to fully assess the correctness of the generated code, and that extending them 80 times, in a variant called HumanEval+, reduces the scores of models considerably. Jain et al. [13] observed that some models perform well on HumanEval but not on newer problems, which suggests overfitting to the benchmark. The benchmark has also reached performance saturation: on the EvalPlus leaderboard [177], OpenAI's o1-preview achieves a pass@1 of 96.3% on the original tests, and open models such as Qwen2.5-Coder-32B-Instruct exceed 90%. HumanEval is therefore no longer considered a definitive test of modern model capability.

2.6.8 MBPP

Mostly Basic Python Problems (MBPP) was first introduced by Google Research in August 2021, just one month after the release of OpenAI's HumanEval (Section 2.6.7), as part of a study on the limits of LLMs for program synthesis [8]. This marked Google's initial contribution to the then-emerging field of LLM code generation.

The dataset consists of entry-level Python programming tasks totaling 974 distinct problems. Each problem contains a short natural language description and unit test cases that the synthesized code must pass, both of which are included in the prompt, as shown in Figure 12. The questions in MBPP were largely crowdsourced, though a subset of 426 problems was hand-verified by the paper's authors to resolve ambiguity and missing details. These problems span diverse tasks, ranging from simple numerical and string manipulations to list processing and number sequence generation.

prompt

```
Write a python function to check if a given number is
one less than twice its reverse. Your code should
satisfy these tests:

assert check(70) == False
assert check(23) == False
assert check(73) == True
```

model

```
def check(n) :
  if n == 2*int(str(n)[::-1])-1 :
    return True
  else :
    return False
```

Figure 12: Example prompt (few-shot examples excluded) and program synthesized by the largest evaluated model in the original MBPP paper. (Adapted from [8])

Later derivatives such as MBPP+ [178] and MBPP Pro [179] were created to address different limitations of the original benchmark: MBPP+ expands the test suite to reduce false positives from weak test coverage, while MBPP Pro reframes MBPP as a self-invoking code generation task, in which a model must solve a problem and then use its solution to solve a related, more complex one.

Upon its release, Austin et al. [8] found that the largest model tested solved 59.6% of the problems of MBPP when allowed 80 samples per problem, leaving substantial room for improvement. However, by September 2024, the EvalPlus leaderboard [177], which uses a hand-verified subset of 399 problems, showed OpenAI's o1-preview model achieving a pass@1 of 95.5% with a single sample per problem, with more than a dozen models scoring above 85%. Similar to HumanEval, these results demonstrate clear dataset saturation.

Five years after its initial release, MBPP has been largely conquered by frontier LLMs. However, in the small-model space, it can still meaningfully differentiate capabilities. On the MBPP leaderboard of LLM Stats [180], models in the 7B to 14B parameter range present distinct, spread-out performance tiers, with a 37-point margin between the lowest and highest-performing models ($\approx 52\%$ to $\approx 89\%$). These scores are reported by the developers of each model, so they are not necessarily obtained under the same conditions.

2.6.9 LiveCodeBench

LiveCodeBench emerged in early 2024 as a benchmark designed to address the shortcomings of the widely adopted coding benchmarks of the time, such as HumanEval and MBPP. Jain et al. [13] identified and sought to address three major problems:

1. By focusing exclusively on natural language to code generation tasks, previous benchmarks ignored the multi-faceted nature of coding and overlooked broader code-related capabilities.

2. With open-sourced, static datasets, previous benchmarks were exposed to data contamination and overfitting, as benchmark samples eventually got included in the training datasets of new models.

3. Prior works highlighted quality issues in previous code benchmarks: HumanEval suffered from ambiguous problems and insufficient tests [176], while MBPP required sanitization to resolve ambiguities and missing details from its crowdsourced questions [8].

To avoid data contamination and ensure problem quality, LiveCodeBench collects problems from three coding competition platforms: LeetCode [146], AtCoder [147], and CodeForces [148]. The quality of the problems is already validated by the users of these platforms, while contamination is avoided by evaluating models strictly on problems published after their training cutoffs. Table 6 outlines the number of problems collected for LiveCodeBench, along with their distribution across difficulty tiers, as of May 2024.

Platform	**Total Count**	**#Easy**	**#Medium**	**#Hard**
AtCoder	267	99	91	77
LeetCode	235	79	113	43
CodeForces	9	4	2	3
Total	**511**	**182**	**206**	**123**

Table 6: Statistics of problems collected in LiveCodeBench, as of May 2024. (Adapted from [13])

To ensure a holistic evaluation of the models' coding capabilities beyond code generation, LiveCodeBench introduces three additional scenarios: **Code Execution**, **Self-Repair**, and **Test Output Prediction**. Code Execution tests code comprehension by prompting the model to "execute" a program on a given input to predict its output. Self-Repair assesses debugging ability by requiring the model to fix an incorrect program using its execution feedback. Finally, Test Output Prediction evaluates problem understanding and logical reasoning by asking the model to determine the correct output for a problem description and input without generating or running any code.

Although LiveCodeBench (LCB) includes four distinct evaluation scenarios, code generation is the most frequently reported one. In the original evaluation, the model with the highest overall score in this scenario, GPT-4o, solved just 33.2% of the Medium problems and 4.2% of the Hard ones [13]. On the official LCB leaderboard [181], which covers problems released until April 2025, the leading model on the problems released from August 2024 onwards, o4-mini (high) [182], reaches 89.4% on Medium difficulty and 63.5% on Hard difficulty. The Easy problem set has therefore reached performance saturation, with the best

models above 99%, and the Medium set is approaching it. The Hard set, in contrast, is far from solved: no listed model exceeds 64% on it, and open models of 32 billion parameters, such as EXAONE-4.0-32B, stay below 47%. This makes LCB a relevant coding benchmark for models of very different sizes.

2.6.10 LiveCodeBench Pro

LiveCodeBench Pro was introduced just over one year after LCB, in a study by Zheng et al. [9]. The paper challenged reports that LLMs were outperforming elite humans in competitive programming, asking: "Are LLMs truly capable of reasoning at the level of elite human competitors?".

Category	Tag	Cognitive Focus	# Problems	% Problems	Difficulty (Elo)
Mathematics	Number Theory	Logic	77	13%	$1\,884 \pm 825$
	Combinatorics	Logic	62	11%	$2\,423 \pm 666$
	Game Theory	Observation	27	5%	$1\,900 \pm 827$
	Others	Logic	118	20%	$1\,608 \pm 775$
Greedy	—	Observation	163	28%	$1\,708 \pm 791$
Data Structures	Segment Tree	Knowledge	38	7%	$2\,629 \pm 502$
	Others	Knowledge	101	17%	$2\,108 \pm 689$
Dynamic Programming	—	Logic	134	23%	$2\,431 \pm 614$
Graph Theory	Tree	Knowledge	53	9%	$2\,308 \pm 528$
	Others	Knowledge	42	7%	$2\,331 \pm 680$
String	—	Logic	44	8%	$1\,595 \pm 867$
Algorithmic Paradigms	Constructive	Observation	120	21%	$1\,849 \pm 894$
	Implementation	Knowledge	108	18%	$2\,057 \pm 837$
	Ad-hoc	Observation	103	18%	$1\,578 \pm 814$
	Case Work	Observation	89	15%	$1\,713 \pm 830$
	Binary Search	Observation	80	14%	$2\,021 \pm 652$
	Bitmasking	Logic	46	8%	$2\,174 \pm 679$
	Two Pointers	Knowledge	22	4%	$1\,777 \pm 528$
	Interactive	Observation	21	4%	$2\,152 \pm 643$
Total			584	100%	$1\,827 \pm 822$

Figure 13: LiveCodeBench Pro dataset statistics. (Reprinted from [9])

Zheng et al. argued that existing coding benchmarks, including LCB, could not adequately answer this question due to limitations such as inconsistent execution environments, weak test cases, and unbalanced difficulty distributions. To address these shortcomings, LiveCodeBench Pro introduces a significantly more challenging evaluation suite. It draws problems exclusively from "top-tier" competitive programming contests, including CodeForces [148], International Collegiate Programming Contest (ICPC) [149], and International Olympiads in Informatics (IOI) [150], while omitting LeetCode [146] problems, which are considered easier and especially prone to training data contamination. Furthermore, LiveCodeBench Pro mitigates contamination through daily problem updates and ongoing human oversight. A team of

competitive coding experts and Olympiad medalists reviews every problem, annotating each entry with detailed metadata, such as required skills, to enable deeper analysis of model performance trends.

In its original version, LiveCodeBench Pro is composed of 584 distinct problems spanning 7 categories, such as Mathematics and Data Structures. Problems are assigned to the Easy, Medium, or Hard difficulty tiers according to their Elo rating, a system originally designed to calculate the skill level of chess players: up to 2,000 for Easy, up to 3,000 for Medium, and above 3,000 for Hard. Figure 13 details the distribution of these problems across categories, task tags, cognitive focus, and difficulty ratings.

Upon its release, the increased difficulty of LiveCodeBench Pro was immediately evident across all tiers. Even on the Easy tier, the best non-reasoning model solved only 32.4% of the problems. Among reasoning models, only o4-mini (high) solved more than half of the Medium problems (53.5%), with the next best model at 25.4%. Notably, none of the models evaluated by Zheng et al. was able to solve a single problem of the Hard tier. Figure 14 illustrates the complete evaluation results across model families.

Model	Hard	Medium	Easy	Rating	Pct.%	AvgTok	AvgCost
Reasoning Models							
o4-mini-high	0.0%	53.5%	83.1%	2 116	1.5%	23 819	$0.1048
Gemini 2.5 Pro	0.0%	25.4%	70.4%	1 992	2.3%	29 879	$0.2988
o3-mini	0.0%	16.9%	77.5%	1 777	4.9%	18 230	$0.0802
DeepSeek R1	0.0%	9.9%	56.3%	1 442	18.0%	16 716	$0.0366
Gemini 2.5 Flash	0.0%	12.7%	47.9%	1 334	30.3%	35 085	$0.0116
DeepSeek R1 Distill-Llama-70B	0.0%	2.8%	33.8%	999	56.0%	12 425	$0.0050
Claude 3.7 Sonnet (Max Reasoning)	0.0%	1.4%	36.6%	992	56.5%	19 075	$0.2861
Gemini 2.0 Flash Reasoning	0.0%	0.0%	29.6%	893	63.1%	11 143	$0.0390
Non-Reasoning Models							
GPT-4.1 mini	0.0%	5.6%	28.2%	1 006	55.5%	2 662	$0.0043
DeepSeek V3 0324	0.0%	5.6%	32.4%	984	57.1%	2 712	$0.0030
GPT-4.1	0.0%	0.0%	23.9%	889	64.2%	2 131	$0.0170
GPT-4.5	0.0%	0.0%	26.8%	881	64.8%	968	$0.1452
Qwen-Max	0.0%	0.0%	14.1%	821	69.4%	1 244	$0.0080
Claude 3.7 Sonnet (No Reasoning)	0.0%	1.4%	16.9%	804	70.7%	3 554	$0.0533
Llama 4 Maverick	0.0%	0.0%	15.5%	634	80.4%	1 160	$0.0007
Claude 3.5 Sonnet	0.0%	0.0%	14.1%	617	81.4%	810	$0.0122
Gemma 3 27B	0.0%	0.0%	8.5%	601	82.5%	668	$0.0001
GPT-4o	0.0%	0.0%	9.9%	592	83.1%	1 133	$0.0227
Meta Llama 3.1 405B Instruct	0.0%	0.0%	9.9%	574	84.3%	568	$0.0005
DeepSeek V3	0.0%	0.0%	12.7%	557	84.9%	1 020	$0.0011

Figure 14: Model performance on LiveCodeBench Pro. (Reprinted from [9])

Frontier models have progressed considerably since then. The benchmark also assigns each model an Elo rating, obtained by comparing it with human contestants. The best rating in the original evaluation was 2,116, for o4-mini (high) [9], whereas Google reports ratings of 2,439 for Gemini 3 Pro and 2,887 for Gemini 3.1 Pro [156]. LCB Pro therefore remains a benchmark aimed at the limits of frontier LLMs,

which makes it unsuitable for the smaller, local models under consideration in this research. In the original evaluation, Gemma 3 27B, an open model larger than any of those studied here, solved 8.5% of the Easy problems and none of the Medium or Hard ones [9]. Scores this close to zero would not allow the coding performance of such models to be compared.

2.6.11 SWE-bench Verified

SWE-bench, introduced by Jimenez et al. [10], evaluates the ability of LLMs to resolve real-world software engineering problems. Unlike code-generation benchmarks such as HumanEval and MBPP, which ask the model to write a short function from a written description, each SWE-bench instance presents a genuine GitHub issue alongside a snapshot of the repository in which it occurs. The model must edit the code to fix the reported problem, which often requires understanding and coordinating changes across several functions, classes, or files, bringing the task much closer to everyday software engineering work [10]. Figure 15 summarizes this process. The repair of existing code has also been studied on narrower tasks. Ribeiro et al. [183], for instance, used GPT-3 to fix type errors in OCaml programs, counting a repair as successful only when property-based tests showed that the patched program produces the same output as the version corrected by its programmer, which was the case for 39% of the 1,318 ill-typed programs evaluated.

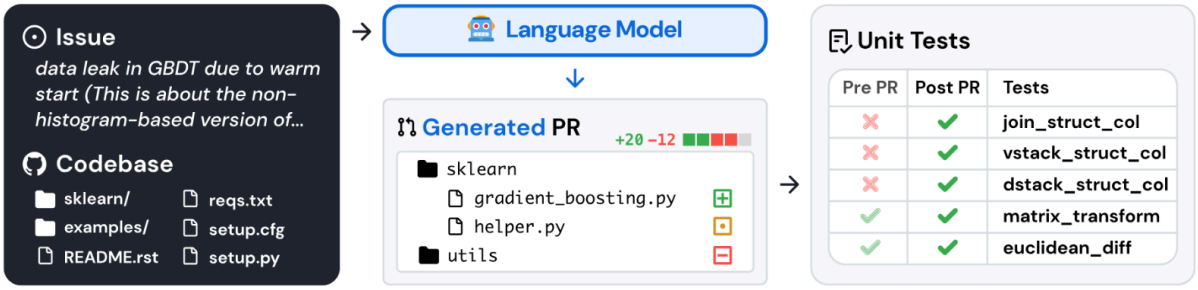


Figure 15: SWE-bench summary of the evaluation process. (Reprinted from [10])

The dataset contains 2,294 such instances collected from 12 popular Python repositories, including Flask [184], Django [185], Scikit-learn [186], and SymPy [187]. A model's proposed fix is evaluated by running the repository's tests: the solution must make the tests that previously failed due to the bug pass again, while keeping all existing tests working. The final score is the percentage of issues successfully resolved. On the original dataset, even the best model of the time, Claude 2, resolved less than 2% of the issues [10].

Because some of the original instances turned out to be ambiguous or impossible to solve, OpenAI later released, together with the authors of SWE-bench, SWE-bench Verified: a subset of 500 instances confirmed by human annotators to be clear and solvable [151]. At its introduction, GPT-4o resolved 33.2%

of this subset when paired with the best open-source agent available. SWE-bench Verified quickly became the preferred version of the benchmark for evaluating models, and results have risen sharply since. The official leaderboard includes a setting in which every model is evaluated with the same simple shell agent (mini-SWE-agent), a program that reads the issue and runs shell commands in the repository to propose and test fixes, without any further tools [188]. In this setting, GPT-4o resolves 21.6% of the issues, while the best score at the time of writing is 76.8%, for Claude Opus 4.5, closely followed by Gemini 3 Flash and MiniMax M2.5 at 75.8% and Claude Opus 4.6 at 75.6% [189]. When measured with more elaborate agent setups, results are higher, with Claude Opus 4.5 reaching 79.2%, which shows that reported scores depend considerably on how the model is employed and not only on the model itself.

SWE-bench is therefore a demanding and realistic software engineering benchmark, but it evaluates complete agent systems rather than the models in isolation, and computing scores requires running tests in full execution environments. Due to its complexity and difficulty, it might not be suitable for evaluating the smaller, local LLMs under consideration in this research.

2.6.12 FACTS Benchmark Suite

The FACTS Leaderboard, proposed by Google DeepMind in 2025 [152], is an evaluation suite that measures the factuality of LLMs, i.e. the ability to generate text that is factually accurate and well-grounded. It aggregates four sub-leaderboards covering complementary scenarios: Grounding v2, which evaluates long-form responses generated from provided documents; Multimodal, which assesses responses to approximately 1,500 image-based questions; Parametric, a closed-book factoid question-answering task with over 2,000 questions probing knowledge internalized during training; and Search, which tests information-seeking scenarios where the model must call a search Application Programming Interface (API) [152].

Automated judge models evaluate responses across each sub-leaderboard using task-specific criteria. The final "FACTS Score" is the average accuracy across the four components. To guard against contamination, every component is split into public and private test sets, and scores combine performance on both [152].

Factuality remains an open problem even for the strongest proprietary models: in the original study, the best model evaluated, Gemini 3 Pro, obtained a FACTS Score of 68.8, with the next best models scoring below 65 [152]. This highlights the importance of factuality as an evaluation dimension for modern LLMs. The suite is, however, impractical for this work. Its scores depend on judge models and on a search tool,

and the complete evaluation, which includes the private test sets, is conducted only by Kaggle [152], so it could not be reproduced on local hardware.

2.6.13 Global PIQA

Global Physical Interaction Question Answering (PIQA) [153] was introduced in 2025 as a culturally diverse, participatory benchmark for physical commonsense reasoning, extending the format of the original PIQA benchmark [190] to over 100 languages and cultures. The benchmark was authored by 335 researchers from 65 countries and covers 116 language varieties across five continents, 14 language families, and 23 writing systems. In the official non-parallel split, each language features exactly 100 examples reviewed by native speakers, with 96.5% being manually drafted without LLM assistance [153].

In the non-parallel split of the benchmark, over 50% of examples refer to culturally specific elements such as local foods, customs, and traditions, and questions are written directly in the target language rather than translated from English. Following the PIQA format, each example presents a physical goal along with two candidate solutions, only one of which is correct, probing physical commonsense in either completion or prompted evaluation settings [153].

Aggregate results mask considerable disparities. The best model evaluated, Gemini 2.5 Pro, achieved 91.7% overall, but scored 80.2% on Sub-Saharan African languages against 95.6% on Western European ones; overall, low-resource languages exhibit accuracy gaps of up to 37 percentage points, with open-weight models performing consistently worse [153]. The benchmark underscores that cultural and linguistic diversity in evaluation data is essential to fairly assess the worldwide capabilities of LLMs, a point that English-only benchmarks such as MMLU-Pro do not address. As a measure of general intelligence, however, it is limited: it evaluates physical commonsense rather than knowledge across domains, and with two options per question, random guessing already yields 50%.

2.6.14 SuperGLUE

SuperGLUE was introduced by Wang et al. [154] in 2019 as a more challenging successor to GLUE, a popular language understanding benchmark that had become saturated shortly after its release: by July 2019, the state-of-the-art models had already surpassed the estimated human performance on that benchmark [154]. SuperGLUE is designed to remain “stickier” and encompasses eight tasks, spanning question answering, natural language inference, coreference resolution, and word sense disambiguation.

While retaining the two hardest tasks from GLUE, SuperGLUE introduced six new tasks selected from an open call for proposals along with several design improvements: expanding task formats beyond sentence-

level classification to include question answering and coreference resolution, providing comprehensive human performance baselines for all tasks, releasing an improved software toolkit, and establishing refined usage rules, such as requiring the dataset versions distributed with the benchmark and prohibiting any use of the unlabeled test data [154].

Despite its original ambition, SuperGLUE was also solved within a few years. By 2021, a single fine-tuned model, DeBERTa, had surpassed the human baseline in the overall score of the benchmark (89.9 versus 89.8) [191], and the benchmark is now rarely reported in state-of-the-art model evaluations. In the current landscape it has effectively been superseded by far more challenging knowledge-oriented datasets such as HLE and MMLU-Pro, and it is therefore not viable for the evaluations conducted in this work.

2.6.15 Summary

From the perspective of this research, which evaluates small models running on local hardware, the benchmarks presented above fall into four groups:

- **Saturated benchmarks.** MMLU, MATH, HumanEval and SuperGLUE have been solved, or nearly solved, by current models, and no longer separate them clearly. MBPP and AIME 2025 are in the same situation for frontier models, although they can still differentiate smaller ones.
- **Benchmarks aimed at frontier models.** HLE and LiveCodeBench Pro were designed to challenge the largest models available. Smaller models would obtain scores too close to zero to be compared.
- **Benchmarks that evaluate more than the model.** SWE-bench Verified measures complete agent systems inside execution environments, and the FACTS suite depends on judge models, a search tool and private test sets. Neither can be reproduced by simply running a model locally.
- **Benchmarks with a narrow scope.** GPQA Diamond covers three sciences and Global PIQA covers physical commonsense, so neither is enough, on its own, to estimate general intelligence.

The benchmarks that remain can be run locally, are graded automatically and still spread the scores of small models: MMLU-Pro for knowledge and reasoning across many domains and, for coding, LCB. Ideally, several benchmarks would be combined into a suite that covers as many domains as possible. The benchmark selected for this research, and the constraints behind that decision, are discussed in Section 4.3.1.

2.7 Related Work

The previous sections presented the concepts, tools and benchmarks on which this thesis builds. This section reviews the studies that share its subject: the energy consumed by LLMs during inference. It is organized around three questions, each closer to this work than the one before. How much energy does inference consume, and what determines it? How does that energy relate to the quality of the answers? And how has it changed over time? The section ends by identifying what these studies leave unanswered, and how this thesis addresses it.

2.7.1 Measuring the Energy of Inference

Research on the energy consumption of AI models has concentrated on training [3], and the first measurements dedicated to LLM inference are recent. Samsi et al. [192] were among the first: they ran LLaMA models with 7, 13 and 65 billion parameters on NVIDIA V100 and A100 GPUs, and reported the energy consumed per second, per generated token and per response. For the largest model, each generated token cost about 3 to 4 joules. Luccioni et al. [193] broadened the scope to 88 models, 10 tasks and 30 datasets, and showed how much the task matters: generating text consumed, on average, over 10 times more energy than classifying it, and general-purpose generative models were far more expensive than models trained for a single task, even when the number of parameters was taken into account.

Subsequent studies examined what determines the energy of an inference. Three factors stand out:

- **The size of the model.** Larger models consume more energy per token, by a factor that depends on the setup. On a server with four H100 GPUs, Niu et al. [194] measured 7.3 times more energy per token for a Llama 3 model with 70 billion parameters than for one with 1 billion, while Husom et al. [195] report roughly two orders of magnitude between models with 70 billion parameters or more and smaller ones. The parameter count is nevertheless an incomplete predictor. In a study of 155 model architectures, Caravaca et al. [196] obtained an R^2 of only 0.49 when predicting energy from the number of parameters alone, and Argerich and Patiño-Martínez [19] observed that the number of layers and the size of the vocabulary also affect consumption.

- **The number of generated tokens.** Each output token requires a pass through the model (Section 2.2.1), which makes the length of the response a very strong predictor of the energy of a request. Husom et al. [195] obtained an R^2 above 0.95 in most settings when predicting energy from the length of the response, while the characteristics of the prompt showed minimal correlation with it. Poddar et al. [197] found a similarly strong correlation with the number of output tokens,

and Fernandez et al. [71] that decoding dominates the energy of a workload in all cases except those with the shortest responses. One consequence is that models of similar size can consume different amounts of energy for the same prompt, simply because some write longer answers than others [73].

- **The way the model is served.** Fernandez et al. [71] compared inference engines, batch sizes, decoding strategies and GPUs, and found that applying the appropriate optimizations reduces energy by up to 73% in relation to an unoptimized baseline. Batching several requests together and quantizing the model both reduce the energy of each request [197], and the batch size in particular has a large effect on both energy and response time [73].

Two public leaderboards apply such measurements systematically. The ML.ENERGY Benchmark [73] reports the energy per response of 40 generative models, served with vLLM on NVIDIA A100 and H100 GPUs under conditions that reproduce those of a service in production. The AI Energy Score [78], launched in February 2025, compares models in ten tasks according to the GPU energy they consume per 1,000 queries.

Two aspects of these studies are relevant to the design of this thesis. The first is how they measure. Most of them report the energy of the GPU alone [71], [73], [78], [192], [196]. Chung et al. [73] justify this choice: the GPU is the dominant consumer in a machine that serves these models, GPU models are more standardized across systems than the other components, and their energy can be measured in software. This leaves out part of the consumption: in the production environment of Google's Gemini assistant, the accelerators accounted for 58% of the energy of a prompt, and the processor and memory of the host machine for another 25% [24]. Several of these studies also serve the models with vLLM [71], [73], [196], which Caravaca et al. justify with its wide use in production. The second aspect is what these studies do not measure. Samsi et al. state that they did not control for the correctness of the outputs [192], and both leaderboards rank models by energy, without considering the quality of what they produce.

2.7.2 Energy and Accuracy

Energy alone says little about efficiency. A model that consumes half the energy of another but answers incorrectly is not more efficient, only cheaper. A smaller group of studies therefore measures energy and accuracy together.

Alizadeh et al. [119] evaluated 26 model families on software development tasks, running each model locally with and without quantization. They found that larger models with higher energy requirements do not

consistently yield proportional gains in accuracy, that large quantized models typically outperform medium-sized unquantized ones in both accuracy and efficiency, and that the number of active parameters, the length of the output and the level of quantization together explain over 90% of the variation in energy. Poddar et al. [197] evaluated ten models on a range of language tasks, and proposed a normalized accuracy metric to compare the trade-off between accuracy and energy across them. Cunha et al. [198] evaluated five quantized local models, with 6.7 to 9 billion parameters, on code completion and code generation benchmarks (HumanEval-X and MBPP+), executing them on the processor alone in order to reproduce deployment on devices without a GPU. Higher accuracy often came at the cost of more energy and longer execution times, but the prompt also affected both: prompts with three examples consumed less energy and time than prompts without examples, an effect that the authors attribute to a smaller number of generated tokens, and, according to them, without compromising the accuracy.

Reasoning is the design choice that has received the most attention, since it exchanges energy for accuracy directly (Section 2.2.2). Dauner and Socher [79] asked 14 models, with 7 to 72 billion parameters, 1,000 questions from MMLU. The models with reasoning were more accurate, but were responsible for far more emissions, mostly because of the additional tokens they generated. Jin et al. [118] compared Qwen2.5 models with 1.5 to 32 billion parameters, with and without reasoning, and concluded that reasoning offers a better trade-off between accuracy and energy than increasing the size of the model, especially in tasks that require complex reasoning instead of factual recall. The cost can nevertheless be extreme: in one case, reasoning multiplied the energy consumed by a model by 113.

The evidence on sparsely activated models is scarcer and less clear. Fernandez et al. [71] measured that an MoE model required more energy than dense models with a comparable number of active parameters, while Niu et al. [194] report that Mixtral 8x7B, another MoE model, consumes roughly the same energy per token as a dense model with 8 billion parameters.

These studies show that energy and accuracy do not grow in proportion, and that the relation between them depends on design choices such as size, quantization and reasoning. All of them, however, are snapshots. They compare the models available at a given moment, and none relates its results to when each model was released.

2.7.3 Efficiency Over Time

Whether AI is becoming more efficient is not a new question. Most of the existing answers, however, concern something other than the energy that LLMs consume during inference.

Training compute and prices. Hernandez and Brown [199] proposed measuring progress by holding performance constant: between 2012 and 2019, the compute needed to train an image classifier to the accuracy of AlexNet fell 44 times, which corresponds to efficiency doubling every 16 months. For language models, Ho et al. [200] estimate that the compute required to reach a given level of performance halved approximately every 8 months between 2012 and 2023. The same reasoning has been applied to prices: Gundlach et al. [201] found that the price of reaching a given benchmark score with frontier models decreased 5 to 10 times per year. These results show that the cost of a fixed level of capability falls quickly, but they refer to the compute spent on training or to commercial prices, not to the energy of running a model.

Inference energy before LLMs. Desislavov et al. [202] studied how the energy of one inference evolved between 2012 and 2021, in computer vision models and in language understanding models. They did not measure it, but estimated it from the number of operations of each model and the efficiency of the hardware available each year. Their conclusion is that the energy of the most advanced models keeps growing exponentially, while models in general use achieve steady gains in performance at an almost constant energy consumption. The study predates the generative LLMs considered here, whose energy depends on the number of tokens they produce.

Hardware and software. MLPerf Power [203] measures the power of complete systems running a fixed set of reference models. For its generative workloads, GPT-J and Llama 2, the energy efficiency of the submitted systems improved more than 100 times in about a year. Since the reference models are the same in every version of the benchmark, these gains come from the systems that run them: their hardware, their software and optimizations such as quantization.

Commercial models. For closed models, the only measurements are those of the providers themselves. Google reports that the median text prompt of its Gemini assistant consumed 0.24 Wh in May 2025, 33 times less than a year earlier, and attributes a 23-fold reduction to improvements in the models and a 1.4-fold reduction to better use of the machines [24]. These figures are measured in production, which makes them realistic, but the models are not available, so the result can neither be reproduced nor broken down further. Outside the providers, the energy of commercial models can only be estimated from indirect information, such as their response times and assumptions about the hardware that runs them [25], [204].

Local, open models. Three studies address the kind of model that this thesis is concerned with. Lu et al. [72] surveyed open models with up to 5 billion parameters and showed that their capabilities improved markedly between 2022 and 2024, by 10.4% to 13.5% depending on the type of task. Their analysis of the cost of running these models focuses on latency and memory use on edge devices.

The study closest to this thesis is that of Saad-Falcon et al. [37], who propose *intelligence per watt*, the accuracy of a model divided by the power it draws, as the metric to follow the progress of local inference. They evaluate more than 20 local models, with up to 20 billion active parameters, on 8 accelerators, using about one million queries that include real conversations, reasoning problems and benchmarks such as MMLU-Pro. As in this thesis, they measure the energy of the accelerator alone and process one query at a time. Their longitudinal analysis pairs the best local model of each year with the best local accelerator of that year: Mixtral 8x7B on an NVIDIA Quadro RTX 6000 for 2023, Llama 3.1 8B on an RTX 6000 Ada for 2024, and GPT-OSS 120B on an Apple M4 Max for 2025. Between the first and the last, intelligence per watt improved 5.3 times, of which the authors attribute 3.1 times to the models and 1.7 times to the hardware. The factor attributed to the models matches the increase in accuracy between the two, from 23.2% to 71.3% of the queries. The main analysis therefore rests on three combinations in which the size of the model (47, 8 and 120 billion parameters in total) and the accelerator change together. An appendix extends it to nine model families between April 2024 and August 2025, again across several GPUs.

A more recent comparison points in a different direction. When the AI Energy Score leaderboard was updated in December 2025, Luccioni and Gamazaychikov [78] compared 14 new text generation models, without reasoning and without MoE architectures, with models of similar size evaluated nine months earlier. Eight of the 14 consumed as much or more energy than their counterparts, with individual results ranging from 3% of the energy to almost twice as much. In the words of the authors, this “runs counter to the prevailing narrative that AI is becoming more efficient”. The comparison is restricted to energy: it does not tell whether the newer models are also more accurate.

These sources give very different answers. Per prompt, the energy of a commercial assistant fell 33 times in one year. The intelligence per watt of the best local configuration improved 5.3 times in two years. And most of the new models added to a public leaderboard consumed as much energy as older ones of the same size, or more. The answers differ because each source measures a different quantity and holds different things constant, which is the starting point of the gap identified next.

2.7.4 Research Gap

Table 7 compares the studies discussed in this section according to the two properties that matter most for this thesis: whether they evaluate accuracy together with energy, and whether they follow efficiency over time.

Study	Models and setting	Accuracy	Over time
	Energy of inference		
Samsi et al. [192]	LLaMA with 7 to 65 billion parameters, on V100 and A100 GPUs	No	No
Luccioni et al. [193]	88 models in 10 tasks, on A100 GPUs	For some tasks	No
Chung et al. [73]	40 generative models, on A100 and H100 GPUs	No	No
	Energy and accuracy		
Poddar et al. [197]	10 models with up to 13 billion parameters, in several language tasks	Yes	No
Alizadeh et al. [119]	26 model families, quantized and unquantized, in coding tasks	Yes	No
Cunha et al. [198]	5 quantized local models with 6.7 to 9 billion parameters, in coding tasks, on a CPU	Yes	No
Dauner and Socher [79]	14 models with 7 to 72 billion parameters, with and without reasoning	Yes	No
Jin et al. [118]	Qwen2.5 with 1.5 to 32 billion parameters, with and without reasoning	Yes	No
	Efficiency over time		
Desislavov et al. [202]	Vision and language understanding models, with estimated energy	Yes	2012 to 2021, before generative LLMs
Tschand et al. [203]	Fixed reference models, on 60 systems	Fixed targets	Systems only, with the models unchanged
Elsworth et al. [24]	Gemini, a closed model, in production	No	12 months, reported by the provider
Saad-Falcon et al. [37]	More than 20 local models, on 8 accelerators	Yes	2023 to 2025, best model and best accelerator of each year
Luccioni and Gamazaychikov [78]	Models of similar size, from two editions of a leaderboard	No	Two groups of models, nine months apart
This thesis	47 open models from 20 releases of 6 families, on one GPU and one inference engine	Yes	2023 to 2026, release by release, on fixed hardware

Table 7: Comparison of the studies most closely related to this thesis.

Three observations follow from this comparison.

First, the studies that measure energy and accuracy together are snapshots of the models available at one moment. They show which design choices are efficient, but not whether the field is moving towards them.

Second, the studies that follow efficiency over time each hold something different constant. MLPerf Power fixes the models and lets the systems change. Saad-Falcon et al. let models and hardware change

together, and follow only the best combination of each year. Google reports on a product in which the models, the software and the use of the machines all changed. The AI Energy Score compares models of the same size, but leaves accuracy out. It is therefore not surprising that their conclusions differ so much.

Third, and as a consequence, none of these studies isolates the contribution of the models. None measures, on the same hardware and with the same software, both the energy and the accuracy of the successive releases of the same model families. Without this, it is not possible to tell whether each new generation of a model is more efficient than the one it replaces, nor whether its gains come from answering better, from consuming less, or from both.

This is the gap that this thesis addresses. It evaluates 47 open models from 20 releases of six families, published between July 2023 and March 2026, on a single GPU, with a single inference engine and the same benchmark. Since the hardware, the software and the benchmark are the same for every model, the differences observed between two releases can be attributed to the models themselves. In relation to the existing work, this design adds three things:

- **Resolution.** The releases of each family are measured one by one, in three size categories, instead of one model per year. This makes it possible to follow the trajectory of the families with several releases, and to see whether they progress together or diverge.
- **Decomposition.** Energy and accuracy are recorded separately for every inference, so a change in intelligence efficiency can be traced to a change in accuracy, in energy, or in both.
- **Design choices.** The set of models includes instruct and reasoning models, as well as dense and sparsely activated ones. Models that support both operating modes are evaluated in each of them, so that the two modes can be compared on exactly the same weights.

This design also has a cost, which delimits the conclusions that can be drawn. The results refer to one GPU model, one benchmark, unquantized models with up to 16 billion parameters, and the energy of the GPU. The thesis is therefore complementary to the work of Saad-Falcon et al., and not a replacement for it. Their study shows how far local inference has advanced when models and hardware improve together, across many accelerators and types of query. This work shows what the models contributed to that advance, and how evenly.

Chapter 3

The Problem and Its Challenges

Despite the growing body of research on Green AI and on the energy efficiency of LLMs discussed in the previous chapter, little is known about how this efficiency has actually evolved over time, particularly across the wide range of model sizes and design choices that have emerged (Section 2.7.4). Identifying the trends in energy consumption and performance, and what drives them, is essential to understand where the field is heading, yet this kind of longitudinal, cross-model perspective is largely absent from existing work. Addressing this gap calls for a controlled empirical study of a curated and representative set of local, openly available LLMs. Such a study must measure the energy these models consume during inference while evaluating them on a benchmark suite that reflects general intelligence across a broad range of domains, so that efficiency and capability are assessed together and not in isolation.

Carrying out a study of this scale, however, is not straightforward. The diversity of the models, the period they cover, and the need for an evaluation that remains fair and consistent across all of them introduce challenges that must be understood before a methodology can be defined. Building on the state of the art, this chapter examines these challenges in six areas: the characteristics of the models themselves, the settings used to run them, the limitations of benchmarks, the constraints of the hardware, the choice of inference engine, and the measurement of energy.

3.1 Model Characteristics

The inherent, model-level characteristics of LLMs directly impact their behaviour and complicate direct comparisons. Models differ not only in scale, but also in numerical precision, quantization strategy, architectural design, and mode of operation, all of which can affect both performance and energy efficiency. Each of the five characteristics below is followed by the challenge it poses to this study.

1. **Parameter Count**: A measure of a model's size. Larger models require more compute power and memory, thus influencing their energy consumption.

Model sizes vary greatly across families and releases, from under one billion to hundreds of billions of parameters. There is no single size that every family offers and at which all of them could be compared, and comparing models of very different sizes directly would not be fair. The study therefore needs an approach that takes these differences into account.

Grouping models of similar size, or calculating metrics relative to the parameter count, are two possible strategies, but neither is without limitations. For example, the relationship between model size and performance is not linear, and larger models may behave differently from smaller ones even after normalizing for parameter count.

2. **Numerical Precision Level**: The data type used to store and compute the model's values (e.g., BF16, FP8, INT4). Higher-precision formats require greater memory footprint, higher memory bandwidth, and more compute capacity, which reduces inference speed.

3. **Quantization Method**: The technique or algorithm (e.g., AWQ, GPTQ, SmoothQuant) used to reduce the precision of model weights and/or activations. Different quantization methods can target different numerical precisions, or even target the same precision while yielding significantly different memory usage, inference speed, and output quality.

Using quantized models instead of their unquantized versions (typically stored in 16-bit BF16 precision) can significantly reduce memory usage and improve throughput, with only a small impact on accuracy. This makes it possible to run more capable models on constrained hardware. However, it also introduces a new variable: the quantization algorithm. Even when models are quantized to the same numerical precision, different algorithms can affect performance differently. In other words, a quantized model's behaviour depends not only on its architecture and features, but also on the specific quantization procedure used.

Using the same quantization process across all models would be a reasonable approach, but it comes with its own challenges. Although there are trusted providers of quantized model versions, finding a quantized version for every model under consideration is nearly impossible. Providers offer only a limited set of models, and as quantization algorithms have evolved over time, so have the models they distribute. A model from 2023 may not use the same quantization algorithm or process as a model from 2026, even if both come from the same provider. In addition, quantization is a fragile process that depends heavily on a model's specific characteristics: not all layers can always be quantized, so each model requires special care.

Quantizing models locally, either dynamically at runtime (online) or ahead of time (offline), would allow full control over the quantization process and provide a reliable, reproducible environment for the models

under consideration. However, the available tools are limited and do not support all model families and architectures, especially older releases, which greatly restricts the scope of models that could be quantized.

4. **Architectural Paradigm (Dense vs. Sparsely Activated)**: The model's structural design. Dense models activate all parameters for each token, whereas sparsely activated models, of which Mixture-of-Experts (MoE) models are the most common example, activate only a subset, which affects compute requirements, efficiency, and behaviour.

MoE architectures are among the most significant recent advances in LLMs, and should therefore be included in this research. However, their differences from dense models complicate a direct comparison.

As explained in Section 2.5.2, an MoE model only activates some of its "expert" networks for each token, which reduces computation and can speed up inference. It is therefore not obvious how such a model should be compared with a dense one: should its size be defined by its active parameters or by its total parameters? In other words, should it be matched against dense models with similar computational demands, or with similar memory requirements?

With inference running entirely on the GPU and no offloading, an MoE model still needs all of its weights loaded into memory, just like a dense model. A dense model and an MoE model with a similar total parameter count therefore have comparable memory requirements, and differ mainly in how much computation they perform. This distinction matters because a lack of compute only makes a model slower, whereas a lack of GPU memory can prevent it from being loaded at all. In local, single-machine settings, where memory is often the main bottleneck, memory requirements may thus be a more meaningful basis for comparison than the active parameter count. A practical way to address this issue is to group MoE and dense models by total parameter count, and compare them within those groups.

MoE is not the only design in which a model uses fewer parameters than it stores. As described in Section 2.5.2, some recent models are identified by an *effective* parameter count that is lower than their total, without relying on expert networks. Although the mechanism is different, these models raise exactly the same questions as MoE models: the computation per token corresponds to a smaller model than the one that must be loaded into memory. For this reason, this document uses the term **sparsely activated** to refer to any model whose active parameter count is lower than its total parameter count, and the reasoning above applies to all of them.

5. **Operating Mode (Instruct vs. Reasoning/Thinking)**: The model's operating mode during generation. Instruct models are optimized for direct answers, while reasoning-oriented models may

produce longer internal deliberation or more structured outputs, which can affect both performance and measurement.

As described in Section 2.2.2, reasoning models improve performance on complex tasks such as logic, mathematics, and planning by internally generating intermediate steps before producing a final answer. The introduction of reasoning or "thinking" techniques has been one of the most important developments in the LLM landscape, and it is now widely used by leading frontier models to achieve strong performance without requiring larger model sizes. This is especially advantageous in the local LLM space, where reasoning capabilities are now common, even among the smallest models. However, fairly evaluating and comparing reasoning models with standard instruct models remains challenging.

The internal "thinking" process of reasoning models often consumes a large number of tokens, sometimes exceeding the length of the final response itself. As a result, these models typically require larger context windows and output limits to accommodate their reasoning process. Otherwise, responses may be truncated, leading to incomplete answers and reducing overall performance. In addition, few-shot prompting, a technique commonly used in benchmarking in which a model is given example question-answer pairs, is often less effective for reasoning models [76]. In some cases, it can interfere with their internal reasoning process and produce outputs that are weaker or even malformed compared with zero-shot prompting, where the model is given no examples at all. This creates a methodological challenge for benchmarking, since applying the same evaluation setup to both reasoning and instruct models may give instruct models an unfair advantage.

3.2 Inference Settings

Configuring the execution of an LLM involves setting multiple parameters that significantly influence model behaviour. The supported values can vary across models, and the impact of these parameters on performance and energy efficiency is not always straightforward. The following parameters are particularly relevant.

1. **Context Window Length**: The maximum amount of input text the model can consider at once. Larger context windows allow more information to be processed, but they also increase memory usage and computational cost.

Although context window length can be set as an inference parameter, every model has a hard limit on how much text it can process at once, and different models support different context lengths. As LLMs

have advanced, context windows have grown significantly, with recent models supporting lengths several times larger than those of older ones. This creates a challenge for fair comparison, since models with shorter context window limits may not be able to process the same amount of information as those with longer windows. It is therefore crucial to ensure that the evaluation process accounts for this limitation and that all tasks fit entirely within the maximum context window of every model considered.

2. **Output Length**: The maximum number of tokens generated by the model for each output. Longer outputs require more decode steps and therefore increase runtime and energy use.

The maximum output length is important because it helps prevent abnormal generation behaviour, such as loops, by cutting off overly long responses. As noted in Section 3.1, reasoning models often produce much longer outputs because they generate intermediate thinking steps. Therefore, using the same maximum output length for both instruct and reasoning models could unfairly disadvantage reasoning models by truncating their answers. For a fair evaluation, the output length should account not only for the tokens needed for the final answer, but also for the additional thinking tokens a reasoning model may generate. This may require different maximum output lengths for instruct and reasoning models to reflect their different needs.

3. **Batch Size**: The number of inputs processed simultaneously in one pass. Larger batch sizes can improve throughput, but limit the ability to precisely measure the energy consumption of individual inferences, as the energy is averaged across multiple inputs.

Most LLM serving engines, such as vLLM, support batching, in which multiple inputs are processed simultaneously. Although larger batch sizes can improve throughput and hardware utilization, potentially increasing energy efficiency, they complicate energy measurement. Because power profiling tools record the aggregate power draw of the entire batch, it becomes difficult to attribute energy consumption to individual inferences. As a result, this limits the reliability of metrics such as energy per token or energy per inference, and therefore makes it harder to assess how model design and architecture affect efficiency. This issue is especially relevant in the context of this research, where it is crucial to examine the relationship between model characteristics and energy efficiency. For this reason, the batch size used in the evaluation must balance throughput against measurement accuracy.

4. **Sampling Parameters**: The decoding settings used to control token generation, such as *temperature*, *top-p*, *top-k*, and *repetition penalty*. These parameters can influence output diversity, determinism, and other aspects of generation behaviour, with varying effects across models.

Sampling parameters control how an LLM selects the next token during generation, shaping how conservative, creative, long, or repetitive the output is. However, using the same sampling settings across different models can be misleading, because those settings do not behave identically from one model to the next. The main issue is that sampling parameters act on each model's *raw probability distribution*, and these distributions can differ significantly. Some models are naturally more confident, others flatter, and some show different tendencies in output length or verbosity. As a result, applying identical settings can benefit one model while unfairly disadvantaging another, even when their underlying capabilities are similar. Rather than using the same values across all models, a fairer comparison may come from tuning these settings for each model, either manually or by following official recommendations.

3.3 Benchmark Limitations

Assessing large language model intelligence is inherently challenging due to fundamental evaluation limitations. While numerous benchmarks exist, as detailed in Section 2.6, no single benchmark can fully capture real-world usage complexity. Benchmarks typically target specific capabilities or tasks, which creates a gap between benchmark performance and actual performance in practical applications.

Selecting appropriate benchmarks requires navigating multiple competing dimensions. Generalized benchmarks spanning multiple domains provide breadth but often lack the specificity to detect nuanced capabilities, while domain-specific benchmarks offer depth at the cost of breadth. Beyond this trade-off, practical concerns further complicate selection: benchmark contamination can artificially inflate performance scores, and benchmarks can become saturated as models improve, losing their ability to discriminate between different levels of knowledge. Task design introduces additional bias, as benchmarks may inadvertently emphasize certain types of reasoning, knowledge formats, or cultural contexts. Language bias is yet another concern: models typically perform better when instructed in English, since most training data and benchmarks are English-centric, which can put them at a disadvantage when evaluated in other languages.

The choice of benchmark for this study must therefore be made carefully. A suitable benchmark needs sufficient domain coverage and technical rigor, a low risk of contamination, enough sensitivity to differentiate between models and, ideally, coverage of more than one language, while remaining feasible within the available time and resources.

3.4 Hardware Constraints

LLM deployment places significant demands on hardware, with compute and memory requirements that scale with model size. While insufficient computational power merely reduces performance, insufficient memory prevents a model from loading entirely. This creates a fundamental constraint: the models available for study are limited by the hardware's available memory.

LLMs can be deployed on a CPU using system RAM, on a GPU using GPU VRAM, or via hybrid offloading strategies that split the model between both when hardware is constrained. Because GPUs are optimized for the parallel operations and matrix calculations that LLM inference requires, GPU-only inference is significantly faster and more efficient than alternatives, making it the industry standard for LLM serving.

For this research, hybrid approaches introduce complications for energy measurement. Data transfers between GPU and system memory consume considerable time and energy, causing GPU idle periods while waiting for data. This creates entangled power consumption patterns across multiple components, making accurate energy attribution difficult and introducing uncontrolled variability. In contrast, CPU-only and GPU-only approaches have localized, uninterrupted execution with minimal overhead, enabling unambiguous energy attribution and far more reliable measurements.

Since CPU-only and GPU-only setups offer comparable measurement reliability, the choice between them comes down to a trade-off between model size and runtime. CPU-only inference uses system RAM, which is inexpensive and readily available, so larger models can be deployed. However, CPU computation has a much lower throughput, which greatly extends runtimes and limits the number of models that can be evaluated in a fixed amount of time. GPU-only inference is much faster, which reduces runtime and allows more models to be studied. However, VRAM is part of the GPU itself: it is expensive, limited, and cannot be expanded without replacing the GPU, which limits the size of the models to the memory of the GPU.

In the context of this research, where the primary goal is to analyse the evolution of LLM efficiency, it is crucial to examine a diverse set of models across a broad range of release dates, sizes, and providers. This variety is essential to identifying potential trends and should be considered when trying to minimize the impact of hardware constraints on the study.

3.5 Inference Engine Selection

As previously explored in Section 2.5, multiple frameworks have been developed to enable the deployment of LLMs. These frameworks, also commonly referred to as inference engines, play a vital role in determining

the performance and energy efficiency of model inference. However, the choice of inference engine is not trivial, as different engines may implement optimizations, batching strategies, and memory management techniques that can significantly influence both runtime and energy consumption.

Given the research requirements of accurately measuring inference energy consumption with reproducible conditions, several factors become critical in selecting an appropriate inference engine:

1. **Mature support and optimization for the hardware in use, namely the CPU and GPU.**

An inference engine that is properly optimized for the hardware in use is essential for the measurements to reflect the models and not the engine. If an engine fails to exploit the capabilities of the available CPU and GPU, the measurements reflect the engine's limitations rather than the model's intrinsic efficiency. This would conceal the true relationship between model characteristics and energy consumption, undermining the goal of assessing how model design affects efficiency. Choosing an engine whose optimized kernels take full advantage of the target hardware therefore ensures that recorded inefficiencies can be attributed to the model itself, rather than to suboptimal execution.

2. **Fine-grained control over batching and caching mechanisms.**

As discussed in Section 3.2, batching already introduces significant complications for energy measurement, since processing multiple inputs simultaneously obscures the energy footprint of any individual inference. Caching mechanisms, particularly prefix caching, present a similar but distinct challenge. In engines such as vLLM, when a request shares the same beginning (prefix) as a previous one, calculations are cached and reused, avoiding recomputation on subsequent requests. While this dramatically improves throughput, it means the computational work performed, and therefore the energy consumed, becomes dependent on how much of the prompt was previously cached rather than on the model's intrinsic behaviour. Consequently, identical prompts can yield very different energy measurements depending on cache state, introducing uncontrolled variability that can compromise the accuracy and reproducibility of the results. For precise and consistent energy measurement, the engine must allow caching and batching to be disabled or otherwise tightly controlled.

3. **Industry-standard adoption for credibility and reproducibility.**

Adopting an inference engine that is the de facto standard in published energy and performance studies is important for a defensible and reproducible methodology. Using the same engine as comparable work allows the results of this study to be positioned directly against existing literature, since differences in

efficiency can then be attributed to the models themselves rather than to execution tooling. This alignment strengthens the credibility of the findings, facilitates validation and replication by other researchers, and helps ensure that the methodology is perceived as sound and its measurements as trustworthy.

4. **Detailed internal logging of execution metrics.**

An inference engine that exposes detailed internal metrics, and offers convenient ways to access them, is essential to ensuring accurate measurements. Instead of relying on external approximations, such an engine directly reports execution data that would otherwise be difficult to obtain reliably, such as the number of tokens processed and generated. This makes it possible to calculate derived metrics that are central to the analysis, such as accuracy per token or energy per token, by combining these metrics with the energy readings and the benchmark results. A simple way of collecting these metrics also reduces the risk of measurement error and makes it easier to automate a large-scale experiment across the many models under consideration.

3.6 Energy Measurement

Accurately measuring the energy consumed during inference is a fundamental requirement of this research, since the reliability of every efficiency metric depends on the correctness of the underlying energy measurements. As explored in Section 2.3, a wide range of tools exists for this purpose, each with its own trade-offs, so selecting the right one requires considering several criteria:

1. **Accuracy**: How faithfully the reported values reflect the actual energy consumed. Errors in the underlying readings propagate directly into every derived efficiency metric, so the accuracy of the tool ultimately bounds the reliability of the study's results.

2. **Coverage**: Which hardware components the tool can measure, such as the CPU, GPU, and Random-Access Memory (RAM). Since LLM inference is a heterogeneous workload that exercises several components at once, the tool must cover all of them, or multiple tools must be combined to obtain a complete picture.

3. **Granularity**: Whether the tool reports only aggregate, system-wide consumption or can attribute energy to individual processes. Per-process attribution is the ideal way to isolate the energy of a single inference from other concurrent activity on the machine. When it is not available, strategies

to minimize interference must be employed, such as reducing background activity or measuring the idle energy consumption of the system.

4. **Platform support**: Whether the tool runs on the hardware and operating system in use, since compatibility constraints can otherwise force the adoption of a less suitable tool or an indirect measurement approach.

Beyond selecting an appropriate tool, precisely measuring inference energy consumption in practice presents its own challenges. Inferencing an LLM is rarely a clean, isolated operation. As discussed in Section 3.4, inference should be localized to a single component to allow unambiguous energy attribution, and, as discussed in Section 3.2, batching and caching can cause energy to depend on execution context rather than on the model itself. In addition, the overhead introduced by the measurement tool itself, though generally small, must be accounted for so that the act of monitoring does not materially distort the energy it aims to record.

Measuring the prefill and decode phases of inference separately presents a further challenge. As seen in Section 2.2.1, LLM inference is composed of two distinct stages: prefill, in which the model processes the entire input prompt in order to generate the first output token, and decode, in which output tokens are produced one at a time sequentially. These phases differ fundamentally in their computational profile: prefill is computationally intensive but brief, whereas decode is a long sequence of smaller, repeated steps. Since the two phases have very different energy characteristics, measuring them separately would show the energy cost of processing input compared with that of generating output. However, telling the phases apart automatically is difficult, since from the point of view of the energy measurement tool they form a continuous stream of computation on the same hardware. The prefill phase is also very short, often lasting only a fraction of a second, so even a small error in locating where it ends, or a measurement tool that only updates its readings at fixed intervals, can distort the energy attributed to it. Distinguishing them therefore requires synchronizing the energy readings with the execution timeline of the inference engine, so that the energy attributed to each phase can be delimited accurately, adding complexity to the measurement procedure.

These considerations together should inform the energy measurement strategy adopted in this research, balancing accuracy, coverage, and granularity to obtain results that are both reliable and interpretable.

Chapter 4

Methodology

This chapter describes the methodology followed in this research. It covers the environment in which the experiment was conducted (Section 4.1), the selection of the models under study (Section 4.2), the design of the experiment (Section 4.3), the way in which each run is executed and measured (Section 4.4), the metrics that are computed from its results (Section 4.5) and the procedures with which the collected data is analysed (Section 4.6). Each of these is described in its final form, as a decision and the reasons behind it. The software that realizes these decisions is the subject of Chapter 5, which also brings together what was corrected after the main execution of the experiment (Section 5.6).

4.1 Experimental Environment

This section describes the hardware and the software that make up the environment in which the experiment was conducted: the server, the inference engine, the tools with which energy is measured, and the framework that orchestrates the runs.

4.1.1 Hardware Configuration

The experiment was executed on a dedicated server provided and managed by the Software Technologies Laboratory (STLab)[12] of the University of Florence, Italy. The server combines two data centre CPUs, Intel Xeon Gold 5418Y[13], and a data centre GPU, the NVIDIA L40S[14], with a large amount of DDR5 memory, and its specifications are listed in Table 8. For this study, its most important component is the GPU and its 48 GB of VRAM, since it performs the inference and its memory determines the largest model that can be evaluated.

12 https://stlab.dinfo.unifi.it/

13 https://www.intel.com/content/www/us/en/products/sku/232379/intel-xeon-gold-5418y-processor-45m-cache-2-00-ghz/specifications.html

14 https://www.nvidia.com/en-us/data-center/l40s/

Server	
Operating System	Ubuntu 22.04.5 LTS
Kernel	Linux 5.15.0-171-generic
Architecture	x86_64 (64-bit)
Memory	16 × SK Hynix 32 GB DDR5
Disk	2 × Micron_7450_MTFDKCC1T9TFR 2 TB
Network	2 × 10 GB Intel Ethernet Controller
CPU	
Processors	2 × Intel Xeon Gold 5418Y, 2.0 GHz
Total Cores	24
Total Threads	48
Max Turbo Frequency	3.80 GHz
Processor Base Frequency	2.00 GHz
Cache	45 MB
TDP	185 W
GPU	
Model	NVIDIA L40S
GPU Architecture	NVIDIA Ada Lovelace architecture
GPU Memory	48 GB GDDR6 with ECC
Memory Bandwidth	864 GB/s
Interconnect Interface	PCIe Gen4 x16: 64 GB/s bidirectional
NVIDIA Ada Lovelace Architecture-Based CUDA Cores	18,176
NVIDIA Fourth-Generation Tensor Cores	568
FP32 (TFLOPS)	91.6
TF32 Tensor Core (TFLOPS)	366
FP16 (TFLOPS)	733
FP8 (TFLOPS)	1,466
Max Power Consumption	350 W

Table 8: Specifications of the STLab server, of each of its two CPUs and of its GPU.

4.1.2 Inference Engine

The models are executed **entirely on the GPU**. Section 3.4 discusses the trade-offs between running models on the CPU and on the GPU, and the need to balance the diversity of the models against the total runtime of the experiment, and this decision follows from those criteria and from the hardware described above. The 48 GB of VRAM of the L40S limit the size of the models that can be evaluated, but CPU inference would be far too slow for the time available, which would restrict the scope of the study much more. Inference on a GPU is also how these models are deployed in practice, which makes the measured energy more representative.

Following the selection criteria established in Section 3.5, **vLLM** (Section 2.5.5) [131] was chosen as the inference engine, for the following reasons:

- Support for a wide variety of model architectures
- Mature optimization for NVIDIA hardware, including the L40S GPU
- Wide adoption in both research and production deployments, including several of the energy studies reviewed in Section 2.7
- Fine-grained control over inference parameters, batching and caching
- Access to detailed internal metrics
- A complete Python library for scripting and automation

The control and transparency that vLLM offers were essential for running the same experiment reproducibly across very different models.

4.1.3 Energy Measurement Tools

vLLM only reports application-level metrics, such as token counts and processing times. Energy is therefore measured with two additional tools: **Intel RAPL** (Section 2.3.1) [94] for the CPU and Dynamic Random-Access Memory (DRAM), read through the interface exposed by the Linux kernel, and **NVIDIA Management Library** (Section 2.3.2) [101] for the GPU, accessed through the pyNVML Python library [103]. This combination was chosen in light of the criteria and challenges laid out in Section 3.6:

- Their accuracy and reliability have been shown in prior studies
- They directly support the Intel Xeon Gold CPU and the NVIDIA L40S GPU
- They are well documented and widely adopted in the research community
- They give precise, low-level control over when and how energy is measured, which higher-level tools hide behind additional layers

Both tools report the consumption of an entire hardware component, and neither can attribute energy to an individual process. This is a known limitation, since the energy of an inference cannot be separated from that of other activity on the machine. For this study, however, it is not critical. The server is dedicated

to the experiment, and the runs consist of long periods of sustained inference that keep the GPU under near-constant load, so that the energy of background processes is small in comparison.

Two additional measures reduce its influence further. Background processes were kept to a minimum during the measurements, and the *idle energy consumption* of the system was measured in a separate experiment, to serve as a baseline. This experiment recorded the energy consumed by the idle system over ten 30-second periods, using the same tools. On average, the idle system draws 35 W on the GPU, 122 W on its two processors and 24 W on their memory.

4.1.4 Orchestration and Software Versions

To execute the large number of runs required by this study in a reproducible way, the experiment is orchestrated with **Experiment Runner** [121], an open-source Python framework for the automatic execution of measurement-based experiments on software systems, published in Science of Computer Programming. With it, an experiment is defined in code in terms of its factors and treatments, and the framework takes care of the execution schedule and of resuming interrupted experiments without losing data. Because it separates the design of an experiment from the details of its execution, it allowed the focus to remain on the design of the study and not on the logistics of the runs.

Small differences in the execution environment can change how inference behaves or how much energy it consumes, which would compromise the comparison between runs and between models. For this reason, the versions of all the software components were **fixed for the entire experiment**. The main components and their versions are listed in Table 9, and the way in which the environment is reproduced is described in Section 5.5.

Component	Version
Python	3.12
CUDA	11.8
vLLM	0.19.1
Transformers	5.8.0
Hugging Face Hub (`huggingface_hub`)	1.21.0
Datasets	5.0.0
Accelerate	1.14.0
pandas	3.0.3
jenkspy	0.4.1

Table 9: Versions of the main software components used in the experiment.

4.2 Model Selection

The validity of this study depends on the models that it evaluates. The selected pool must be diverse and representative, covering multiple sizes, providers and release dates, while respecting the hardware and time constraints of the experiment. This section describes the procedure through which this pool was built. The procedure comprises the six stages summarized in Figure 16, each of which is described in one of the following subsections, and the last subsection characterizes the set of models that results from it. The entire procedure is automated, and its implementation is described in Section 5.2.

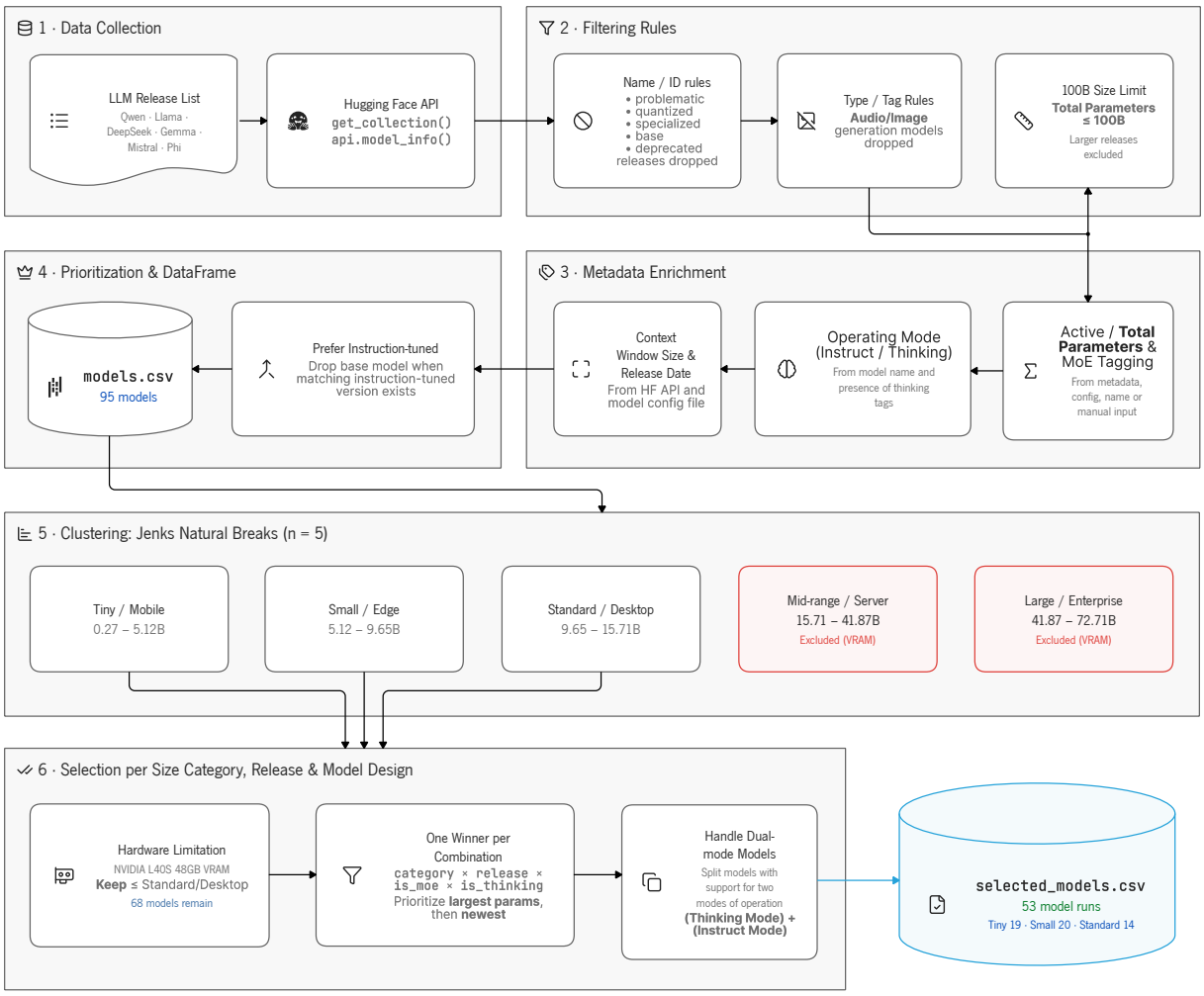


Figure 16: Complete pipeline of LLM data collection & selection.

4.2.1 Families and Releases

The first decision concerned the LLM families to be studied. Given the nature of this research, this choice was driven by the following requirements:

- Each family should be widely used and recognized

- Each family should have a history of successive releases
- Each family should offer compact variants suitable for local deployment on consumer hardware
- Combined, the set of families should offer structurally diverse models
- The set of families should be geographically diverse

Based on these criteria, six families were chosen, covering Europe, America and Asia (Table 10).

Name	Developer / Company	Geographic Origin
Mistral	Mistral AI	France
Gemma	Google	USA
Llama	Meta	USA
Phi	Microsoft	USA
DeepSeek	DeepSeek AI	China
Qwen	Alibaba Cloud	China

Table 10: Selected LLM families with corresponding developers and geographic origins.

The models of each family are collected from the Hugging Face (HF) Hub, which, as mentioned in Chapter 2, is the central platform of the open-source machine learning community, where developers and researchers publish and download pre-trained models. On the Hub, each release of a provider is grouped, together with all of its model variants, under a *collection* with a unique identifier, so that the full catalogue of a release can be collected automatically from the identifier of its collection. Table 11 lists the releases considered for each family and the identifiers of their collections.

A collection is maintained by its provider, and does not always contain every variant of a release. For this reason, models can also be added to a collection explicitly, and are then treated like any other model in it. This was required in a single case: the 16-bit versions of the three Ministral 3 instruct models, which Mistral publishes in separate repositories that are not part of the collection of that release. The versions listed in the collection are quantized, and are excluded for that reason (Section 4.2.2).

4.2.2 Filtering Rules

A collection contains many variants that are not relevant to this study. As the models of each collection are collected, an initial filter removes these variants, using three kinds of rules: keywords in the name of the model, the type and the tags that HF assigns to it, and an explicit list of models. Table 12 summarizes what is removed and how it is identified.

Family	Release	Hugging Face Collection ID
Mistral	Ministral 3	mistralai/ministral-3
	Mistral Large 3	mistralai/mistral-large-3
	Mistral Medium 3.5	mistralai/mistral-medium-35
	Mistral Small 4	mistralai/mistral-small-4
Gemma	Gemma	google/gemma-release
	Gemma 2	google/gemma-2-release
	Gemma 3	google/gemma-3-release
	Gemma 3n	google/gemma-3n
	Gemma 4	google/gemma-4
Llama	Llama 2	meta-llama/llama-2-family
	Llama 3	meta-llama/meta-llama-3
	Llama 3.1	meta-llama/llama-31
	Llama 3.2	meta-llama/llama-32
	Llama 3.3	meta-llama/llama-33
	Llama 4	meta-llama/llama-4
Phi	Phi 1	microsoft/phi-1
	Phi 3	microsoft/phi-3
	Phi 4	microsoft/phi-4
DeepSeek	DeepSeek	deepseek-ai/deepseek-llm
	DeepSeek V2	deepseek-ai/deepseek-v2
	DeepSeek V2.5	deepseek-ai/deepseek-v25
	DeepSeek R1	deepseek-ai/deepseek-r1
	DeepSeek V3	deepseek-ai/deepseek-v3
	DeepSeek V3.1	deepseek-ai/deepseek-v31
	DeepSeek V3.2	deepseek-ai/deepseek-v32
	DeepSeek V4	deepseek-ai/deepseek-v4
Qwen[15]	Qwen 1.5	Qwen/qwen15
	Qwen 2	Qwen/qwen2
	Qwen 2.5	Qwen/qwen25
	Qwen 3	Qwen/qwen3
	Qwen 3.5	Qwen/qwen35
	Qwen 3.6	Qwen/qwen36

Table 11: Selected LLM releases with corresponding HF collection IDs.

15 The original "Qwen" collection is ignored since its models are incompatible with current package versions and not natively supported. The Qwen 1.5 collection contains re-released model weights from the original collection, but in a format that is natively supported.

Removed Models	Identified by	Examples
Models that generate images or audio	Type and tags	`text-to-image`, `text-to-audio`
Quantized variants	Name	`gguf`, `awq`, `fp8`, `4bit`
Specialized variants	Name	`code`, `math`, `vision`, `ocr`, `guard`
Models that generate text by diffusion	Name	`diffusion`
Variants for specific frameworks or runtime formats	Name	`pytorch`, `keras`, `mlx`, `onnx`
Base models, not tuned for instructions	Name	`base`, `-pt`
Experimental or deprecated builds	Name	`exp`, `deprecated`
Individual models that the rules above do not cover	Explicit list	See Table B.1, in the appendix

Table 12: Rules of the initial filtering, with examples of the keywords and types used to identify each kind of model.

Two of these rules define the scope of the study. Intelligence is evaluated here through questions that are answered in text, so that only models that generate text can be assessed, and the rule on models that generate images or audio is the safeguard that enforces this condition. Models that also accept images or audio as input, but generate text, remain eligible and are evaluated on text alone. The rule on diffusion models has the same purpose. Instead of producing one token at a time, as described in Section 2.2.1, a model such as DiffusionGemma generates its response by refining whole blocks of tokens in parallel [205], which is a different inference process from the one studied here.

The rule on quantized variants, in turn, follows from the challenge of quantization raised in Section 3.1. Since no quantization procedure could be applied in the same way to every model, quantized models would introduce a variable that differs from one model to another, the algorithm with which they were quantized. All the models are therefore evaluated with their unquantized weights, a decision detailed in Section 4.2.6, and their quantized variants are removed at this stage.

The explicit list contains the few models that could not be handled by a general rule, which are listed, with the reason for the exclusion of each, in Table B.1 (in the appendix). Most of them are models that the inference engine could not run in the experimental environment (Section 4.1.4). The others are cases in which a rule based on names fails. In particular, the rule on quantized variants cannot identify a quantized model whose name does not say so, which is the case of the three Ministral 3 instruct models: they are distributed with 8-bit weights under their plain names.

Finally, models with more than **100 billion parameters** are excluded, for two reasons. First, this research focuses on models that can be deployed locally, on consumer hardware or on a single machine, which is not practical for models of this size. Second, the limit protects the classification into size categories

(Section 4.2.5). Below 100 billion parameters, the sizes of the models increase gradually, whereas above it they jump abruptly to several hundred billion parameters, and such extreme values would distort the categories. Since this rule depends on the parameter count of each model, it is only applied after that information has been collected, as described in the next subsection.

4.2.3 Model Metadata

A collection only provides the identifier of each model. To support the selection process, to design a fair experiment and to draw meaningful conclusions from its results, more information about each model is needed. Five characteristics are therefore collected for every model, each of which is defined below, together with the reason why it is needed. The sources from which they are obtained, and the way in which sparsely activated, reasoning and dual-mode models are identified, are described in Section 5.2.2 and Section 5.2.3.

- **Release date.** The release date of a model is the date on which its repository was created on the HF Hub. Open LLMs are sometimes announced before being uploaded to the Hub, but the real-world impact of the efficiency of a model depends on when it actually became available, and not on earlier announcements or publications. The date on which a model became publicly accessible on the Hub gives a reference point that can be verified and that is consistent across all models.

- **Number of parameters.** As explained in Section 3.1, not every model uses all of its parameters to process each token. Two counts are therefore recorded for each model: its *total* number of parameters and its number of *active* parameters.

- **Architectural paradigm.** For the purposes of this study, models are divided into two groups: *dense* models, which use all of their parameters, and *sparsely activated* models, which only use part of them. For dense models, the number of active parameters is simply the total. For sparsely activated models, the two values differ. This second group contains two kinds of models:

  - MoE models, in which only some of the expert networks are active for each token;

  - the Gemma 3n and Gemma 4 models whose names contain an "E" (such as `gemma-3n-E4B-it`), which are identified by Google through their *effective* parameter count [135], [136]. These are not MoE models: as described in Section 2.5.2, their effective size is smaller than their total size because a large share of their parameters are embeddings, which are looked

up instead of being computed, and, in Gemma 3n, because part of the model is only needed for image and audio inputs.

Both kinds are grouped together because they share the property that matters for this study, which is that the amount of computation per token corresponds to a model smaller than the one that must be loaded into memory. For this reason, they raise the same question about the relationship between the size of a model and its energy consumption.

- **Operating mode.** Unlike the previous characteristics, which describe the structure of a model, the operating mode describes how it behaves during inference: a reasoning model does not answer directly, and instead uses part of its output to reason before giving the final response. Not every model, however, is either an instruct model or a reasoning model. The *dual-mode* models introduced in Section 2.2.2 offer both an instruct mode and a reasoning mode, selected at inference time through a parameter of their *chat template*, the text template that defines how a conversation is converted into the text that the model reads, including where each message starts and ends and who wrote it. This distinction is essential for the experiment, because dual-mode models must be evaluated in both modes, while the other models are only evaluated in the one that they support.

- **Maximum context window.** The maximum size of the context window of each model is needed for a fair experiment design, since the prompt and the response must both fit in the context window and, as Section 3.1 explains, reasoning models need additional room for their reasoning. Knowing this limit makes it possible to define prompt lengths and output limits that every model can handle.

4.2.4 Prioritization of Instruction-tuned Variants

Once the metadata of all the models has been collected, one last filter is applied to the list. When it contains both a base model and a tuned variant of it, identified by a suffix such as `-it`, `-instruct`, `-chat` or `-hf`, the base model is removed and only the variant is kept. This complements the initial filtering, which already removes the base models that can be recognized by their name alone.

Base models are trained only to predict the next token, and are not prepared to follow instructions or to hold a conversation in the way that users expect from an LLM. The models that are actually deployed in practice are almost always the tuned variants, and most benchmarks are designed with them in mind. Evaluating a base model would therefore not reflect realistic usage, and keeping both versions would count the same model twice. After this step, the pool contains **95 models**.

4.2.5 Size Categories

As explored in Section 3.1, a direct comparison between models of very different sizes is not fair. To address this, the models are grouped into size categories, so that comparisons can be made between models of similar size.

The categories are based on the *total* parameter count of each model, and not on its active parameter count. As justified in Section 3.1, this groups the models by their *memory requirements*, shifting the focus from computational cost to *deployability*. A 14B-parameter MoE model may only activate 3B parameters for each token, but all 14B parameters must still be loaded into memory, so that it needs more capable hardware than a dense 3B-parameter model. The two cannot always be deployed on the same machine, and grouping by total parameters makes the categories reflect this practical constraint.

The limits of the categories were not chosen by hand. They were computed with **Jenks Natural Breaks** optimization [206], a method that divides a set of values into groups so that the values within each group are as similar as possible, and the groups are as different from each other as possible. This method suits the data, because open models are not evenly distributed across sizes: developers tend to release models around a few common sizes, with gaps between them. Jenks places the limits in these gaps, where fixed thresholds could easily split a group of similar models in two.

The limits were computed over the total parameter counts of the 95 models. The number of categories was set to **five**, since tests with other values showed that this number gives a balanced distribution of models while keeping the categories meaningfully different. Table 13 shows the resulting categories, named after the typical deployment scenario of the models in each range. These scenarios assume quantized models, as is frequent in local execution (Section 2.5), so that the unquantized models of this study require more memory than these names suggest (Section 4.2.6).

Size Category	**Range** (Billion Parameters)	**Models**
Tiny/Mobile	$[0.27, 5.12]$	33
Small/Edge	$(5.12, 9.65]$	21
Standard/Desktop	$(9.65, 15.71]$	14
Mid-range/Server	$(15.71, 41.87]$	16
Large/Enterprise	$(41.87, 72.71]$	11

Table 13: Size categories and corresponding total parameter count ranges.

To confirm that every category contains models from the whole period under study, and not only recent or only older ones, the distribution of the models over time was analysed (Figure 17). All five categories contain models spread across the timeline, with enough models in each to support meaningful conclusions.

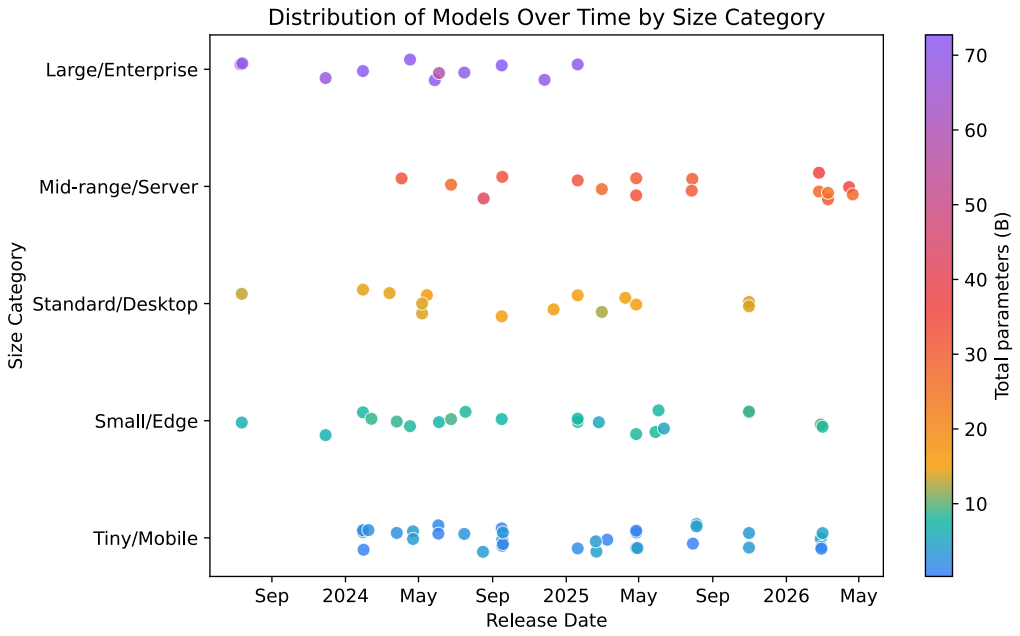


Figure 17: Distribution of models over time by size category.

4.2.6 Final Selection

Three decisions reduce the pool of 95 models to the set that is evaluated: the numerical precision at which the models are executed, which determines the size categories that the hardware can hold, the selection of a single representative among similar models, and the treatment of the dual-mode models.

The first decision is the numerical precision. As explored in Section 3.1, quantization reduces memory usage and improves throughput, but it also adds a variable that is difficult to keep consistent: quantized versions are not available for every model, quantization techniques have changed over time, and the tools for quantizing models locally do not support every architecture. To avoid these problems, all models are executed in **unquantized precision**, using the 16-bit BF16 format.

This choice limits which models can be studied, since unquantized weights need considerably more memory. The 48 GB of VRAM of the NVIDIA L40S GPU can hold the models of the three smallest categories, while leaving room for the context window and the caches of the inference engine. The models of the two largest categories, *Mid-range/Server* and *Large/Enterprise*, do not fit: the smallest of them has 25.8B parameters, which need about 52 GB in this precision. These two categories were therefore excluded, reducing the pool from 95 to **68 models**.

The second decision brings the pool down to a manageable size. A single model is selected for each combination of *size category*, *collection*, *architectural paradigm* and *operating mode*. When several models share the same combination, the one with the most parameters is chosen, since it is expected to be the most capable of its category, followed by the most recently released if there is still a tie. This rule

does not reduce the time span covered by the study: since each collection corresponds to one release of a family, keeping at least one model per collection keeps the pool spread over the entire timeline of releases.

The third decision concerns the dual-mode models, which can operate in instruct or in reasoning mode (Section 4.2.3). To evaluate both, each of them is split into **two separate runs**, one per mode, identified by an "(Instruct Mode)" or "(Thinking Mode)" suffix added to the name of the model. This makes it possible to compare instruct and reasoning models, and also the two modes of the same model. The implementation uses the term "thinking" for what this document may call "reasoning", and the two are equivalent.

4.2.7 Composition of the Final Set

The selection resulted in **47 models**: 32 instruct models, 9 reasoning models and 6 dual-mode models. With each dual-mode model evaluated twice, this gives a total of **53 model runs**. The final set covers 20 of the 32 releases listed in Table 11. The remaining releases only contain models that are larger than 100 billion parameters, belong to the two excluded size categories, or are incompatible with the experimental environment.

Figure 18 shows how the final models are distributed by operating mode and architectural paradigm. The final set is dominated by instruct (68.1%) and dense (89.4%) models, while reasoning, dual-mode and sparsely activated models are much less common. Of the five sparsely activated models, two are MoE models and three are Gemma models. This imbalance is expected, since instruction-tuned dense models

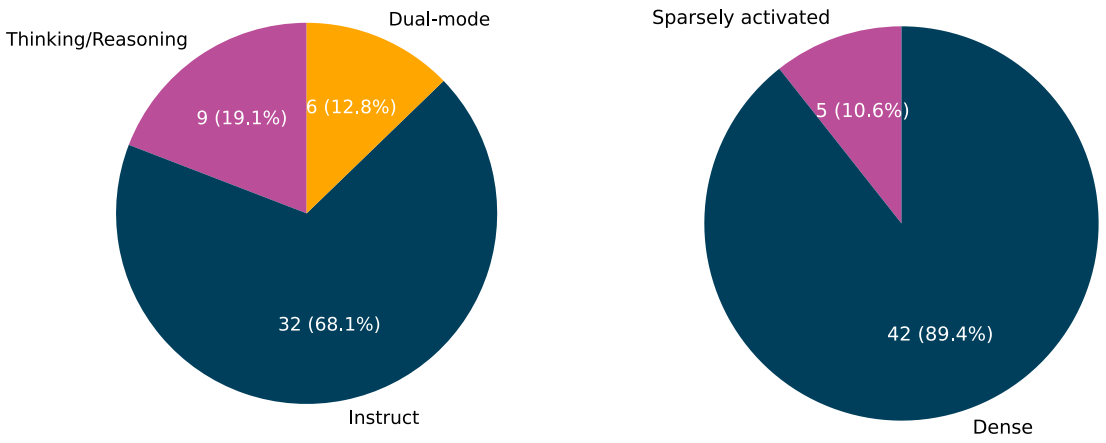


Figure 18: Distribution of models by operating mode (left) and architectural paradigm (right).

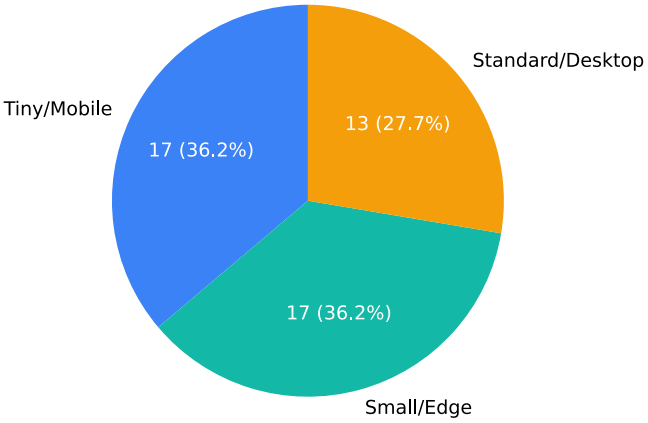


Figure 19: Distribution of models by size category.

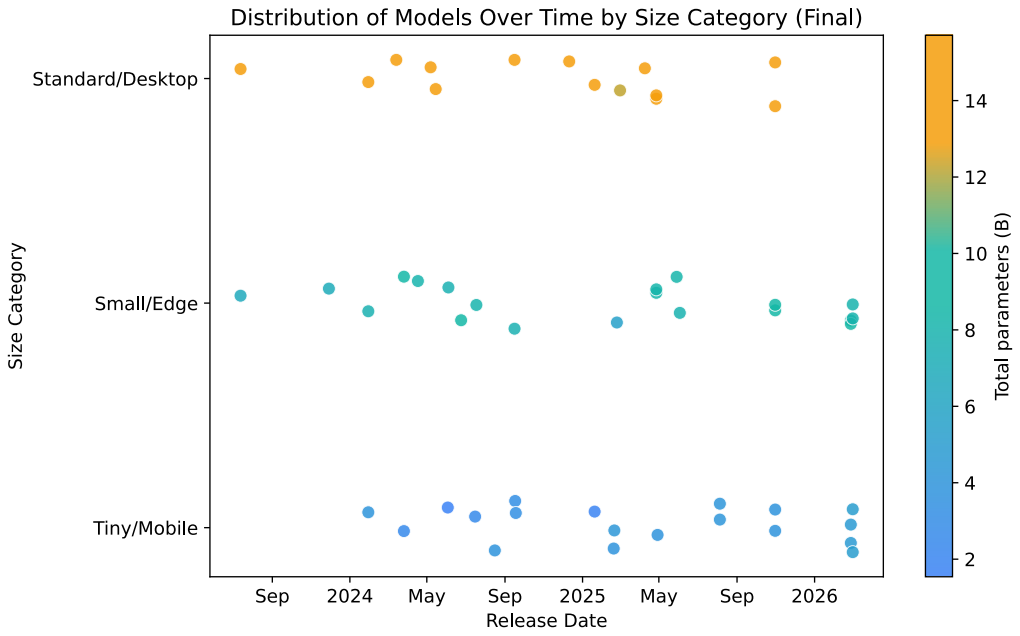


Figure 20: Distribution of models over time by size category (final set).

are by far the most common releases among open LLMs. It reflects the current landscape, and not a bias introduced by the selection process.

It is, nonetheless, a limitation of the study. The less common types of models are underrepresented, so that the conclusions that can be drawn about them are weaker. The main findings on the evolution of efficiency over time are therefore based mostly on instruct, dense models, while the remaining types provide additional insight into how these characteristics affect efficiency. These proportions should also not be extended to all open models, since the two excluded size categories may have a different distribution of model types.

Regarding size, Figure 19 shows that the models are evenly distributed across the three categories: Tiny/Mobile and Small/Edge have 17 models each (36.2%), and Standard/Desktop has 13 (27.7%). This balance also holds over time, as Figure 20 confirms: after all the selection steps, every category still contains models from the whole period under study.

4.3 Experimental Design

The design of the experiment results from four decisions: which models are evaluated, on which benchmark, how many times, and in which order. This section presents these decisions, and Section 4.4 describes the conditions under which each run is executed. The experiment is orchestrated with the Experiment Runner framework [121], introduced in Section 4.1.4, in the manner described in Section 5.3.

4.3.1 Model Runs and Benchmark

The experiment evaluates the models selected by the pipeline described in Section 4.2, in a total of **53 model runs**. These correspond to 47 models, plus 6 additional runs for the dual-mode models, which are evaluated in both their instruct and their reasoning mode so that the two can be compared.

All the models are evaluated on a **single standardized benchmark**, MMLU-Pro (Section 2.6.4) [7], which comprises approximately 12,000 high-quality multiple-choice questions across fourteen domains of knowledge. In relation to the original MMLU, it raises the number of options per question from four to ten, which makes it considerably harder for a model to reach the correct answer by guessing. As discussed in Chapter 2, MMLU-Pro also continues to discriminate well between the smaller open models, whereas many older benchmarks have become saturated.

No single benchmark can capture everything that a language model is able to do (Section 3.3), and executing several benchmarks was not viable at this scale. MMLU-Pro was selected because it balances breadth of topics with depth of reasoning, although it only covers English and therefore carries a language bias. A coding benchmark was also considered, since programming is one of the most common uses of language models and is not well covered by MMLU-Pro. However, with more than forty models evaluated ten times each, a second benchmark would have made the total duration of the experiment impractical.

4.3.2 Repetitions and Randomization

Each model run is **repeated 10 times**. A single measurement can be affected by factors that are unrelated to the model, such as background processes, thermal throttling, jitter of the operating system

and noise in the energy measurements. Repeating each run and aggregating its results reduces the effect of these fluctuations, makes outliers easy to identify, and increases the confidence in the results. As seen in Section 2.4, 30 repetitions are often recommended for experiments of this kind, but that number was not feasible here: with 10 repetitions, the 530 runs already required about 1,189 hours of execution, almost 50 days.

The order in which the runs are executed is **fully randomized**. If the repetitions of each model were executed consecutively, any temporary condition of the machine, such as accumulated heat or a peak of background activity, would affect all the measurements of the same model and bias its results. Randomization distributes these conditions evenly across the models, as recommended for energy experiments (Section 2.4). No fixed waiting period is observed between runs. Instead, the hardware is brought to the same temperature at the start of each run, as described in Section 4.4.4.

Together, these decisions result in a total of **530 individual runs**:

$$53 \text{ model runs} \times 1 \text{ benchmark} \times 10 \text{ repetitions} = 530 \text{ total runs}$$

Figure 21: Calculation of the total number of experimental runs.

These 530 runs constitute the main execution of the experiment. After it, the three Ministral 3 instruct runs were replaced by runs of the same models with 16-bit weights instead of 8-bit weights, obtained in a supplementary execution with only 5 repetitions (Section 5.6), which brings the total to 545 runs.

4.4 Execution of a Run

Every run follows the same procedure, in which one model is loaded, evaluated on the benchmark, and measured in each of its responses. The procedure builds on the official evaluation script of MMLU-Pro, provided by TIGER-AI-Lab [207], which was substantially extended for this study. Since the original only measured accuracy, most of the additions concern measurement and fairness:

- stratified sampling of the benchmark, which reduces the duration of the experiment to a manageable one while keeping the evaluation representative of the full dataset;
- prompt construction through the chat template of each model, which supports both instruct and reasoning models;
- warmup and thermal control, so that every run starts with the hardware in the same state;

- measurement of the energy and of the performance of each prompt, by means of Intel RAPL, NVML and the internal metrics of the inference engine.

A run comprises five phases, summarized in Figure 22 and described in the following subsections. Their implementation is presented in Section 5.4. One principle governs the order of these phases: **only the inference itself takes place inside the measurement window**. Any other operation that uses the hardware, such as loading the model, building and tokenizing the prompts, or extracting the answers, is performed either before the measurements start or after they end.

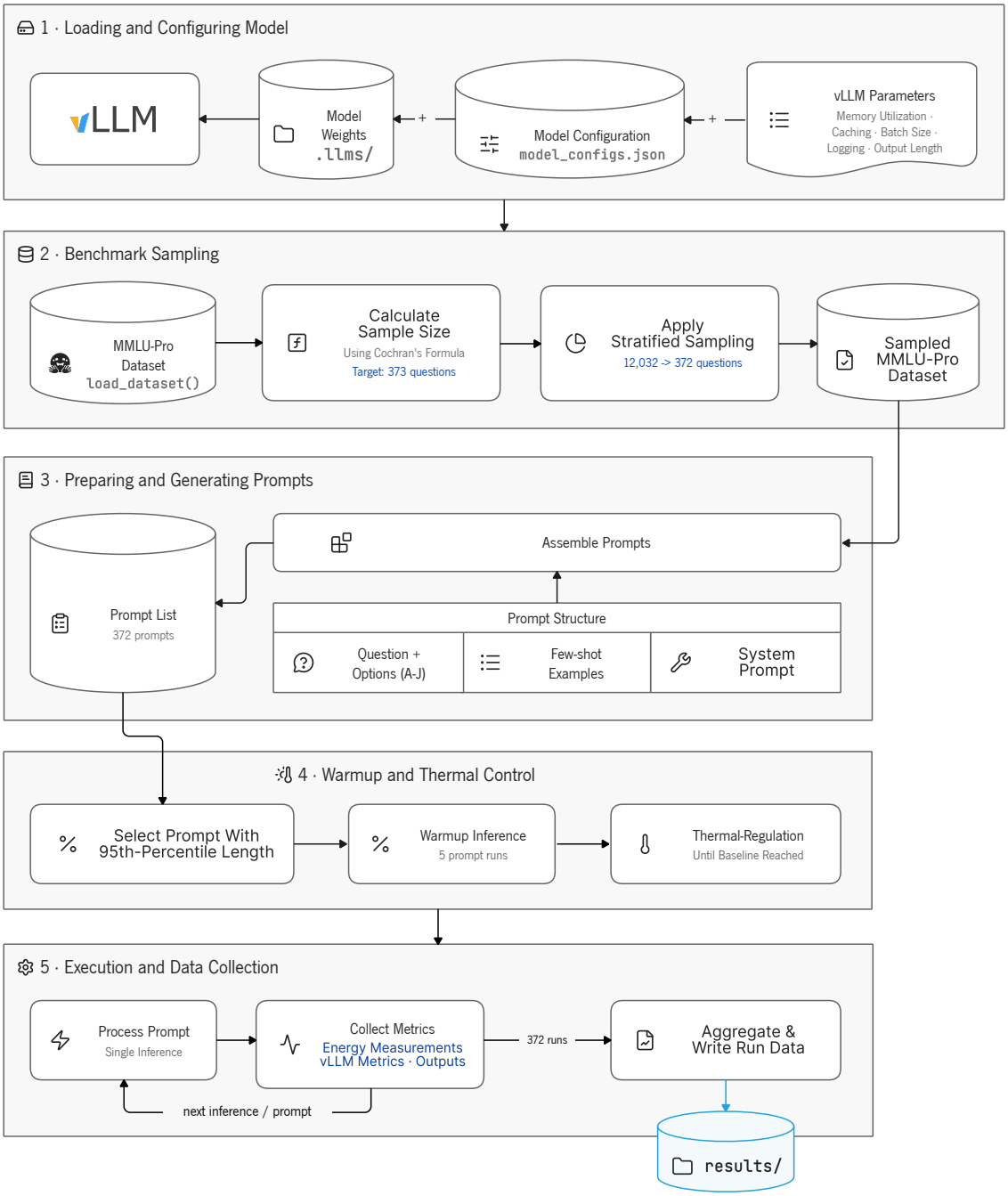


Figure 22: Overview of the model evaluation and execution workflow.

Each run is executed in isolation, so that it always starts from a clean state: the model is loaded from scratch, and all the memory is released when the run ends, which ensures that no run is affected by the one that preceded it. The total duration of a run, referred to as its *run time*, is also recorded. It covers the entire run, including the loading of the model, the preparation of the prompts and the warmup of the hardware, and therefore differs from the inference time measured within the run (Section 4.4.5), which only covers the generation of the responses.

4.4.1 Model Loading and Inference Configuration

The first phase creates the inference engine and establishes how the model generates text. Four aspects of this configuration bear on the fairness and on the reliability of the measurements: the sampling parameters, the context window, the output limit and the configuration of the engine.

As explored in Section 3.2, the **sampling parameters** of a model, such as its temperature, have a significant effect on its output and on its performance. For a fair comparison, each model must be evaluated with the settings **recommended by its creators**, and not with a single configuration shared by all the models. These settings are obtained from four sources, in order of preference:

1. explicit recommendations stated on the Hugging Face model card;
2. implicit recommendations found on the model card, such as the values used in its code examples;
3. the configuration files of the repository of the model, which often contain default generation parameters;
4. the default values of the inference engine, used as a last resort when none of the above provides a value.

The process comprises two steps. A baseline is first created automatically from the third and fourth sources, and is then reviewed manually against the model card, so that explicit or implicit recommendations replace the automatic values. Dual-mode models have separate settings for each operating mode, since their creators often recommend different values for each of them.

The **context window** addresses the differences in context length between models discussed in Section 3.2. Each model runs with the largest context window that it supports, so that it is evaluated at its full capability, up to a **limit of 10,240 tokens**. This limit prevents excessive memory usage and Out-Of-Memory (OOM) errors, and corresponds to the largest context window that the available memory could accommodate with the largest models of the study.

The **output limit** determines how much a model can generate. Its base value is **2,048 tokens**, the same that the official MMLU-Pro script uses. This is sufficient for the long, step-by-step answers that the harder questions require, even from the more verbose models. It also interrupts the models that become trapped in repetition, which would otherwise keep generating, and consuming energy, until the context window is full. Reasoning models generate their reasoning before the final answer, and both count towards the output limit. As identified in Section 3.2, giving instruct and reasoning models the same limit would therefore place reasoning models at a disadvantage, since a limit that is reasonable for an instruct model may be exhausted by the reasoning alone, leaving the answer truncated. Reasoning models are therefore given a **"thinking budget" of 4,096 tokens**, which is added to the base limit, so that a long reasoning trace cannot consume the tokens intended for the final answer:

$$\text{output limit} = \underbrace{2{,}048}_{\text{answer}} + \underbrace{4{,}096}_{\text{thinking budget}} = 6{,}144 \text{ tokens}$$

The thinking budget is twice the base limit, so that a reasoning run can generate three times as many tokens as an instruct run. A larger budget would have further prolonged the reasoning runs, which already account for 920 of the 1,125 hours of inference of the experiment (Section 6.7), and would have made its total duration difficult to accommodate in the time available. This budget nevertheless proved insufficient for part of the reasoning runs: in 13 of the 15, between 8.3% and 29.6% of the responses reached the output limit, most of them while the model was still reasoning (Section 6.4.4). This constitutes one of the limitations of the study (Section 7.5), and its consequences for the results are examined in Section 6.7.

Dual-mode models can run in either mode, and each mode is evaluated as a separate run (Section 4.2). The mode of a run determines both its thinking budget and the number of examples included in its prompts (Section 4.4.3).

Generation also ends when the model produces a **stop condition**, namely one of its end-of-sequence or end-of-turn tokens. The tokens related to reasoning are deliberately not used as stop conditions, so that reasoning models are not interrupted in the middle of their reasoning. For instruct models, the beginning of a new question is also treated as a stop condition. Having just seen five examples in a question-and-answer format, these models sometimes continue after their answer by writing a new question of their own, and stopping at that point keeps the measured energy limited to the answer itself.

The **configuration of the engine** was chosen to make the energy measurements reliable, in response to the challenges discussed in Section 3.2 and Section 3.5:

- **One request at a time.** The engine processes a single prompt at a time. Prompts are never processed simultaneously, so that each energy measurement corresponds to exactly one inference.

- **No reuse of computation between prompts.** The reuse of the computation of prefixes that are shared between prompts, such as the few-shot examples, is disabled. Every prompt is computed from scratch, so that its energy does not depend on the prompts that preceded it.
- **Internal metrics.** The internal metrics of the engine are enabled, which provides exact token counts and the durations of the prefill and decode phases of every request.
- **Optimized execution.** The models run in the optimized execution mode of the engine, as they would in a real deployment.
- **Memory.** The engine may use 90% of the VRAM, its default, for the weights of the model and for the KV cache, the remainder being left as headroom to avoid OOM errors.

Finally, the engine uses a **fixed random seed**, vLLM's default value of 0, and no seed is set for individual requests [208]. A model that samples its responses therefore draws the same sequence of random numbers every time its engine is started, which makes its responses reproducible. The effect of this on the repetitions of a model run is analysed in Section 6.1.

4.4.2 Benchmark Sampling

Even a single benchmark is too large to be executed in full, since answering all 12,032 questions in each of the 530 runs would take far too long. A statistically representative subset is therefore selected by **stratified sampling**. The size of the sample is calculated with Cochran's formula [209], for a confidence level of 95% ($z = 1.96$), a margin of error of 5% ($e = 0.05$) and an estimated proportion of 50% ($p = 0.5$), which is the most conservative choice. A finite population correction is then applied for the $N = 12{,}032$ questions:

$$n_0 = \frac{z^2 p(1-p)}{e^2} \approx 384.16 \qquad n = \left\lceil \frac{n_0}{1 + \frac{n_0 - 1}{N}} \right\rceil = 373$$

These 373 questions are then distributed across the fourteen categories in proportion to their size in the full dataset, so that the subset preserves the mix of domains of the original. The number of questions of a category h is $n_h = \mathrm{round}\left(\frac{N_h}{N} \cdot n\right)$, where N_h is the size of that category. As a result of the rounding, the final subset contains **372 questions**, one fewer than the target. Table 14 shows the result.

Category	Original Population Size	Sampled Size
Math	1,351	42

Category	Original Population Size	Sampled Size
Physics	1,299	40
Chemistry	1,132	35
Law	1,101	34
Engineering	969	30
Other	924	29
Economics	844	26
Health	818	25
Psychology	798	25
Business	789	24
Biology	717	22
Philosophy	499	15
Computer Science	410	13
History	381	12
Total	**12,032**	**372**

Table 14: Distribution of MMLU-Pro questions by category before and after stratified sampling.

Within each category, the questions are shuffled and the first n_h are selected. The shuffle uses a fixed random seed, so that every model and every repetition answers **exactly the same 372 questions**. Differences in accuracy or in energy between runs therefore originate in the models, and not in the questions that they happened to receive. The few-shot examples are taken from the separate validation split of MMLU-Pro, so that they never coincide with the evaluated questions.

4.4.3 Prompt Construction

Before any inference takes place, each question has to be converted into a prompt that the model can interpret. A prompt has three parts: the task instructions, a number of few-shot examples, and the question under evaluation. The way in which these parts are assembled depends on the model, and most of the effort of this phase lies in supporting the different chat templates and operating modes introduced in Section 3.1.

The chat template of a model (Section 4.2.3) converts a list of messages, each with a role such as *system*, *user* or *assistant*, into the exact string of special tokens on which the model was trained. Its use matters for fairness, since a prompt in the wrong format can lower the score of a model for reasons that are unrelated to its actual ability. The official MMLU-Pro script was written before chat templates were widely adopted, and builds every prompt by joining raw strings in a fixed format. That logic is retained only

as a fallback for older models without a chat template, all the other models being prompted through their own template.

Figure 23 shows the final structure of a prompt, which the remainder of this subsection follows from top to bottom.

Every prompt begins with the task instruction written by the MMLU-Pro team (Listing 1), in which `{$}` is replaced by the subject of the question:

```
The following are multiple choice questions (with answers) about {$}. Think step by step
and then finish your answer with "the answer is (X)" where X is the correct letter
choice.
```

Listing 1: Task instruction template written by the MMLU-Pro team.

Not every model supports a *system* role. Where it is supported, the instruction is placed in a system message. Otherwise, it is added to the beginning of the first user message.

The few-shot examples follow, each consisting of a user message with a question and its options, and of an assistant message with the reference step-by-step answer. Their number addresses the differences in prompt design between instruct and reasoning models discussed in Section 3.1. Few-shot examples help instruct models by showing them the expected format of the answer, but they often interfere with reasoning models, which are designed to work through a problem on their own [76]. Each type of model is therefore prompted in the manner for which it was designed: **5 examples for instruct models and 0 for reasoning models**. Using no examples for any model would be simpler, but it would place instruct models at a disadvantage, since MMLU-Pro expects answers in a strict format and is designed to be used with few-shot prompting. The last user message contains the question under evaluation, with its options labelled from A to J.

The chat template of the model then converts the list of messages into the final prompt, adding the role tags, the special tokens and the marker that signals to the model that it is its turn to answer. For dual-mode models, it is at this point that the operating mode of the run is selected. Finally, for instruct models, the text `Answer: Let's think step by step.` is appended to the prompt. This matches the format of the answers of the examples, and asks the model to reason before answering. Reasoning models do not receive this text, since they already reason on their own and it could interfere with their reasoning.

The reasoning models of the Ministral 3 release required two further adjustments. First, their documentation prescribes a specific system prompt, which describes how the reasoning and the answer are to be structured (Listing B.7, in the appendix), and which is added after the instruction of MMLU-Pro. Second, the opening delimiter of the reasoning block is appended to the very end of the prompt, so that the model

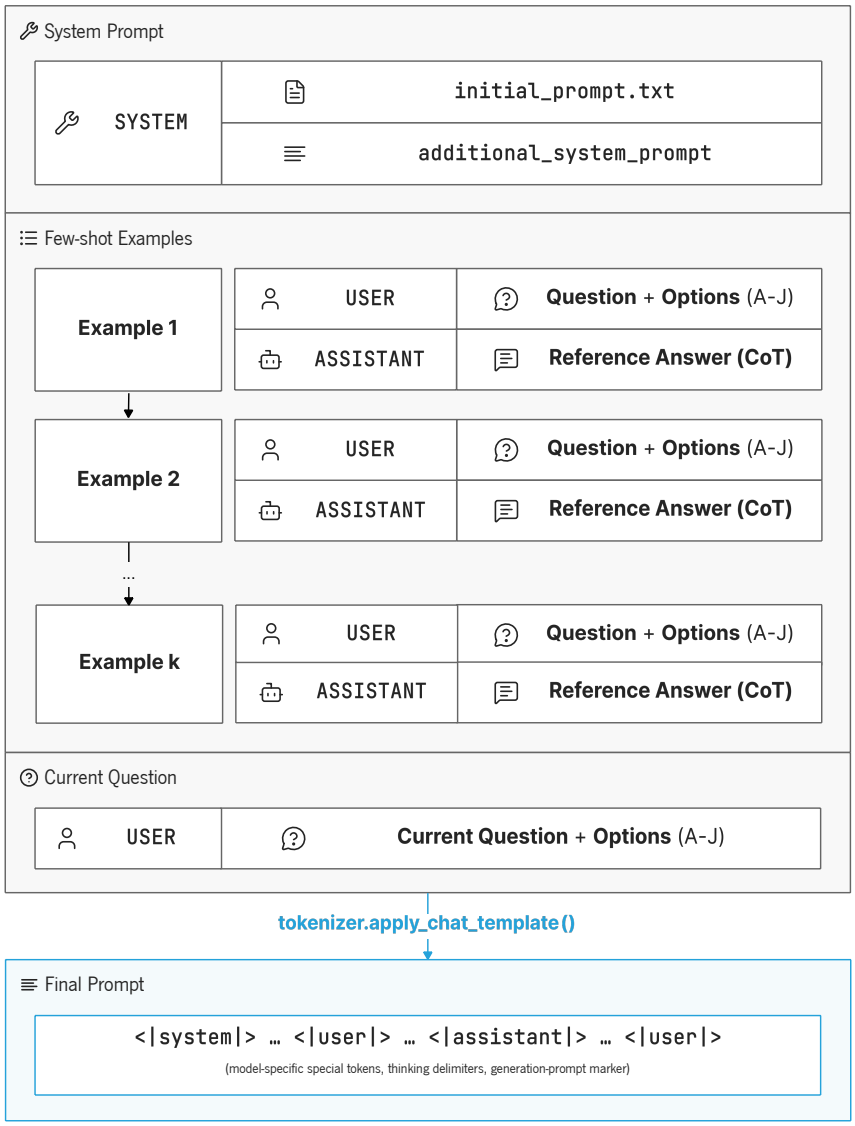


Figure 23: Structure of the final prompt sent to each model.

begins its response inside a properly opened block. Without it, these models sometimes failed to mark where their reasoning ended, which prevents the count of the thinking tokens described in Section 4.5.1. This adjustment resolved most of these cases, but the smallest model, with 3 billion parameters, still produced malformed outputs occasionally, so that **its counts of thinking tokens should not be considered fully reliable**.

Each finished prompt is tokenized and compared with the context window, leaving room for the answer of 2,048 tokens. When a prompt is too long, one few-shot example is removed and the prompt is rebuilt, until it fits. This addresses the challenges of the context window raised in Section 3.2: models with shorter context windows are never silently truncated, and still receive as many examples as they can accommodate. In this experiment, examples were removed only for the four instruct models with a context window of 4,096 tokens (Llama 2 7B and 13B, DeepSeek-LLM 7B and Phi-3-medium-4k-instruct), in 12 to 20 of their 372 prompts, which received two to four examples instead of five. Every other instruct prompt contained the

five examples. All 372 prompts are built and tokenized before the inferences start, since doing so on the CPU during the measurements would add energy and time that are unrelated to the model.

4.4.4 Warmup and Thermal Control

Between the preparation of the prompts and the first measurement, the hardware is brought into a controlled and repeatable state. This phase has two purposes. First, it warms up the inference engine, so that the first measured inference behaves like all the others. Second, it brings the GPU and the CPU to the same temperature at the start of every run. As discussed in Section 3.6, this is required for the recorded energy to reflect the model, and not the state in which the machine happened to be when the run started.

The first inferences that a model performs are slower than the subsequent ones, because the engine still has one-time initialization to complete, such as allocating memory and initializing the GPU kernels that it uses [71]. To keep this initialization out of the measurements, one prompt is processed **five times** before anything is measured. The prompt is the one whose length lies at the **95th percentile** of all the prepared prompts. It is close to the longest inputs that the model will receive, so that the engine is exercised under near-worst-case conditions, while remaining a real prompt of the benchmark. The number of inferences and the choice of the prompt are the same for every model, so that all the models undergo identical initialization. Larger models take longer to complete this step, which is nevertheless necessary to ensure that no model starts being measured with an engine that is not fully initialized. These five inferences also serve as a calibration: the temperature of the GPU before and after them provides an estimate of the number of degrees that each prompt adds, which is later used to determine how much heating is required.

The power drawn by a processor depends on its temperature, because the *leakage current*, the power that is lost even when the processor is idle, increases as the chip heats up [210]. A processor close to its thermal limit also lowers its clock speed to protect itself, which is the *thermal throttling* already identified as a source of variability in Section 4.3. Two runs of the same model can only be compared fairly if they start at the same temperature [12]. Otherwise, part of the difference in their energy originates in the environment, and not in the model.

Each run brings the GPU to **55 ± 1 °C** and the CPU to **35 ± 2 °C**. These are not idle temperatures. Since the hardware heats up considerably during inference, an idle target would require long cooling periods between runs. The targets were instead taken from the temperatures that the system naturally reached between consecutive runs and after the preparation of the prompts. As a result, only the first run

of the experiment requires a long warmup, and each subsequent run leaves the system close to the target of the next one.

The tolerances differ between the two components. The GPU heats and cools slowly and predictably, so that it can be kept close to its target, and it is responsible for most of the measured energy. The temperature of the CPU changes quickly and, since the CPU is mostly idle during inference, it has little effect on the measurements. A wider tolerance for the CPU avoids spending time on the control of a temperature that barely affects the results.

To reach both targets, each component is heated or cooled until both are within their tolerance. The GPU is heated by processing the warmup prompt again and the CPU by an arithmetic load on all its threads, while either of them is cooled by resting. The two components are regulated independently, so that one can be heated while the other cools down. This prevents the regulation from oscillating indefinitely, which could happen if heating one component to its target pushed the other out of its range (Section 5.4.3). Once the five warmup inferences are complete, both targets are reached in a median of **17.2 seconds**.

The direct control of the fan speed of the server was also considered. It would allow a more precise control of the temperature and reduce the influence of the room temperature, but it requires low-level access to the hardware of the server, which was not available, and adds considerable complexity. Since the regulation already reaches a stable baseline within seconds, it was not justified.

4.4.5 Inference and Data Collection

Once the hardware has reached its baseline temperature, the prepared prompts are processed one at a time, and each inference is measured individually. Every value of energy and of performance used in the analysis originates in this phase. Its design therefore aims at keeping each measurement free of any activity other than the inference itself, and comparable with all the others. The way in which these decisions are implemented is described in Section 5.4.4.

Energy is not measured by sampling power at regular intervals, but through the **cumulative energy counters** that the hardware updates continuously. The GPU exposes such a counter through NVML, in millijoules, and Intel RAPL provides equivalent counters, in microjoules, for each of the two processors and for the memory attached to each of them. The energy consumed by an inference is the difference between the readings taken immediately before and immediately after it.

This approach is more reliable than sampling. A power sample only captures the consumption at a single instant, so that short peaks between two samples are missed, and the energy has to be estimated under the assumption that the power remained constant in between. As discussed in Section 2.3.2, Yang

et al. [84] showed that estimating the energy of a GPU in this manner can lead to errors of up to 70%. A counter, by contrast, is accumulated by the hardware itself and does not depend on how often it is read, so it is not subject to these sampling errors [105].

The counters remain limited by the sensors on which they rely. For NVIDIA GPUs, Yang et al. [84] found the power sensor to have a proportional error of approximately 5%, which differs between devices but is consistent for a given device. Since every model of this study runs on the same GPU, this error affects all of them equally: the absolute values of energy carry this uncertainty, but the comparisons between models, which are the object of the study, do not.

The **measurement window** encloses a single inference. It opens immediately before the generation of one response is requested, and closes as soon as that response is complete and no work remains pending on the GPU. The same window provides the duration of the inference. Three further measures keep everything other than the inference outside it:

- the prompts are built and tokenized before the first measurement, so that the construction of a prompt never falls within the window;
- the engine processes one prompt at a time (Section 4.4.1), so that each measurement corresponds to a single inference, and not to a batch of unknown composition;
- every operation that does not need to be measured, such as the extraction of the answers and the count of the thinking tokens, is performed after all the prompts have been processed.

The RAPL counters have a limited range, of approximately 262 kJ on this server, after which they restart from zero [96]. This overflow is a known limitation of RAPL, and a frequent source of errors in measurement tools (Section 2.3.1) [93]. At the roughly 87 W that a processor draws during inference, a counter takes about 50 minutes to cover its range, whereas the longest inference of the experiment lasted about four minutes. A counter can therefore restart at most once during an inference, which makes the correct difference recoverable in every case, in the manner described in Section 5.4.4. The counter of the GPU does not restart in practice. In the main execution of the experiment, the RAPL counters were read differently, which affected the CPU and DRAM energy of the collected data, but not the GPU energy, as described in Section 5.6.

In addition to energy, the inference engine reports, for each prompt, the number of prompt tokens and of generated tokens, together with the time spent in the *prefill* phase, in which the prompt is processed, and in the *decode* phase, in which the response is generated. The prompts of the warmup are not included in these values.

Group	Metric	Unit	Source
Temperature	GPU temperature	°C	nvidia-smi
	CPU temperature	°C	psutil
Timing	Prefill time	s	vLLM
	Decode time	s	vLLM
	Inference time	s	Wall clock
Energy	GPU energy	J	NVML
	CPU energy	J	Intel RAPL
	DRAM energy	J	Intel RAPL
	Total measured energy	J	Sum of the above
Tokens	Prompt tokens	tokens	vLLM
	Generated tokens	tokens	vLLM
	Total tokens	tokens	Sum of the above
	Thinking tokens	tokens	Tokenizer

Table 15: The values recorded for every prompt, grouped by what they describe, with their unit and source.

Energy is, however, not measured separately for each phase. As discussed in Section 3.6, such a separation would reveal how much of an inference is spent on processing the prompt and how much on generating the response, but it could not be obtained reliably in this setup, for two reasons. First, the engine only returns control once the response is complete, and it runs in a separate process, so the counters cannot be read at the moment in which the prefill phase ends. The duration of each phase is known only because the engine records it internally and reports it afterwards. Second, the prefill phase is very short: its median duration is 92 ms for instruct models and 47 ms for reasoning models, whose prompts are considerably shorter, since they contain no few-shot examples (Section 4.4.3). These durations correspond to 3.8% and 0.12% of the inference time, respectively. A measurement taken over such a short interval would be dominated by the delay in reading the counters and by the frequency with which they are updated. Energy is therefore measured for each inference as a whole, and only the duration of each phase is recorded.

Table 15 lists the values recorded for each prompt. The temperatures are read immediately after each inference, in order to observe how they vary during a run.

4.5 Answers and Efficiency Metrics

Once all the prompts of a run have been processed, the values recorded during the inferences are complemented by the answer extracted from each response. They are then combined into a summary of

the run, from which the efficiency metrics are computed. This section describes these three steps, whose implementation is presented in Section 5.4.5 and Section 5.4.6.

4.5.1 Answer Extraction

Each response is first separated into its reasoning and its final answer. The responses are generated with their special tokens preserved, so that the delimiters of a reasoning block remain visible in the text. The block is identified by its closing delimiter, which marks the end of the reasoning, and its length is recorded as the number of **thinking tokens** of the response. The block is then removed, so that the answer is extracted from the final response, and not from the options that the model considered while reasoning.

The predicted option is given by the first of three patterns that matches the response, from the most to the least strict:

1. the format requested in the task instruction, `the answer is (X)`;
2. a line of the form `Answer: X`;
3. the last standalone letter between A and J in the response.

When none of the patterns matches, the original behaviour of MMLU-Pro is preserved, and a **random option** is selected, using a fixed random seed. A response that cannot be parsed therefore obtains the expected accuracy of a random guess, instead of being systematically counted as incorrect. The **accuracy** of a run is the fraction of the 372 questions answered correctly.

Both steps depend on the closing delimiter of the reasoning block. When the response of a reasoning model does not contain it, there is no block to count or to remove: the response is recorded with **zero thinking tokens**, and the three patterns are applied to its entire text. This occurs in two situations. In the first, the model reasoned in plain text, without the delimiters, and gave its answer at the end. The entire text is then the response of the model, and the answer is extracted from it as for an instruct model, although its reasoning is not counted as thinking tokens. In the second, the response reached the output limit (Section 4.4.1) while the model was still reasoning. No final response exists in this case, and the prediction is extracted from the unfinished reasoning, most often through the last of the three patterns. The second situation is the more frequent one: it occurred in 7,959 of the 55,800 responses of the reasoning runs (14%), against 1,460 for the first. It affects the accuracy and the thinking tokens recorded for reasoning models, and its consequences are analysed in Chapter 6.

4.5.2 Summary of a Run

The records of the prompts of a run are combined into a summary of that run. Energy, tokens and time are summed, temperatures are averaged, and the average power and throughput are obtained by dividing the corresponding totals. The energy, the generated tokens and the duration of an inference are reported both as a mean and as a median over the questions of the run, because a small number of very long responses can raise the mean well above the value of a typical question. The summary also counts two kinds of responses that matter for the interpretation of the results: those that were truncated upon reaching the limit of generated tokens (Section 4.4.1), and those from which no answer could be extracted.

The total measured energy is the sum of the three components that were measured. In the data of the main execution, the CPU and DRAM components cover one of the two processors (Section 5.6). Since the summary of a run depends exclusively on the records of its prompts, it can be recomputed from them at any time, which was done once after the experiment (Section 5.6). The fields of the summary are listed in Table 22, in the description of its implementation (Section 5.4.6).

4.5.3 Efficiency Metrics

The efficiency metrics are the main indicators used in the analysis. They are computed on the **GPU energy** of a run, and are defined as follows, where e_i is the GPU energy of the inference of question i, $E_{\text{GPU}} = \sum_{i=1}^{Q} e_i$ the total GPU energy of the run, G the total number of generated tokens, A the accuracy of the run and $Q = 372$ the number of questions:

$$\text{energy per question} = \frac{E_{\text{GPU}}}{Q}$$

$$\text{median energy per question} = \text{median}(e_1, ..., e_Q)$$

$$\text{energy per token} = \frac{E_{\text{GPU}}}{G}$$

$$\text{tokens per joule} = \frac{G}{E_{\text{GPU}}}$$

$$\text{accuracy per joule} = \frac{A}{E_{\text{GPU}}/Q}$$

$$\text{energy per correct answer} = \frac{E_{\text{GPU}}}{A \times Q}$$

The last two metrics carry the same information. Since $A \times Q$ is the number of questions answered correctly, the accuracy per joule is the number of correct answers obtained per joule, and the energy per correct answer, its reciprocal, is the energy spent on each correct answer.

The median is the only one of these metrics that is not based on the total energy of the run. It describes the energy of a typical question, and is reported for that purpose alone. All the other metrics, including the accuracy per joule, use the total energy: since the accuracy of a run is measured over all of its questions, it must be related to the energy spent on all of them. A ratio based on the median would disregard the energy of the longest responses, which is part of the cost of obtaining that accuracy.

The metrics per token divide the energy of a run by the tokens that the model generated, and leave out the tokens of the prompts, because the two kinds of tokens have very different costs. The prompt is processed in the prefill phase, which accounts for a very small part of the duration of an inference (Section 4.4.5), whereas every generated token requires its own pass through the model (Section 2.2.1). Prompt tokens nevertheless constitute the large majority of the tokens of an instruct run: with five few-shot examples in every prompt, they represent 83% of all the tokens processed in the instruct runs, against 9% in the reasoning runs, which receive no examples. A metric computed over all tokens would therefore reflect mostly the length of the prompts, and would not be comparable between the two types of models. Related studies make the same choice, and report the energy per output token [119], [192], [195]. The energy in these metrics remains that of the whole inference, and therefore includes the small part that is spent on processing the prompt.

No efficiency metric is computed on the total measured energy. The analysis relies on the GPU energy alone, for four reasons:

1. **It is where the model runs.** With GPU-only inference, the computation of the model takes place on the GPU. Its energy is the part of the consumption of the system that depends on the model and on how efficiently it converts energy into correct answers, which is what this study sets out to measure.

2. **The rest of the system does not distinguish between models.** The processor that was measured draws practically the same power for every model: between 86 W and 89 W on average, plus 12 W for its memory. Its energy is therefore little more than a constant multiplied by the inference time, which is already measured. Including it would add a penalty proportional to time, but no information about the model.

3. **It is the only complete measurement.** The GPU energy was measured directly for every inference and required no correction, whereas the CPU and DRAM energy covers only one of the two processors and had to be partly reconstructed (Section 5.6).

4. **It is what most related studies measure.** As seen in Section 2.7, most studies of the energy of LLM inference report the energy of the GPU alone, so that results based on it share the measurement boundary of the existing literature.

This choice has a consequence that must be borne in mind when these results are compared with those of other studies: the energy reported here is that of the GPU, and not of the whole system. The GPU accounts for 71% to 76% of the energy that was measured, depending on the model, and for a smaller share of that of the whole system, since the second processor is not included. As a reference, the two processors and their memory draw 146 W when the system is idle (Section 4.1.4).

4.6 Data Analysis

This section describes the procedures used to analyse the data collected in the experiment, whose results are presented in Chapter 6. The analysis is implemented in a set of Jupyter Notebooks, one for each section of that chapter, which are available in the experiment's repository [39] (Section 5.1). Every value reported in Chapter 6 derives from the output of one of these notebooks.

4.6.1 Values of a Model Run

Each combination of a model and an operating mode constitutes a *model run*, which was repeated 10 times, or 5 times for the three Ministral 3 instruct runs of the supplementary execution (Section 5.6). Unless stated otherwise, the value reported for a model run is the **median of its repetitions**. Five quantities are used throughout the analysis, in which the energy is always that of the GPU and the tokens are the generated ones:

- the *accuracy* (Section 4.5.1), expressed as a percentage, with differences in percentage points. The accuracy that random answers would obtain on this sample is referred to as the chance level. It amounts to 11.4%, the mean of the inverse of the number of options of each question, and not to 10%, because only 293 of the 372 questions have ten options, the remaining 79 having between three and nine;
- the *energy per question* and the *energy per token*, as defined in Section 4.5.3;
- the *tokens per question* and the *time per question*, which are the means, over the 372 questions of a run, of the generated tokens and of the duration of the inferences (Section 4.5.2). Since a small

number of long responses can raise the mean of the tokens, the evolution of the response length is also examined through the median over the questions. The responses are, in addition, compared with the answers of the examples of the prompt (Figure 23), with both lengths counted in words.

The intelligence efficiency is reported as the number of **correct answers per kJ**, which corresponds to the accuracy per joule multiplied by 1,000. It can be expressed as the product of three factors:

$$\text{intelligence efficiency} = \frac{\text{accuracy}}{\text{energy per question}} = \frac{\text{accuracy}}{\text{tokens per question} \times \text{energy per token}}$$

A model is therefore more efficient if it answers more questions correctly, if it generates fewer tokens in doing so, or if each token requires less energy. When two model runs are compared, the change in intelligence efficiency equals the product of the changes in the three factors, so that, on a logarithmic scale, it can be partitioned exactly into the contribution of each factor. This decomposition is applied to successive releases, to the two operating modes of a model, and to the first and last release of a family.

The temperatures of the GPU and of the CPU, read after each inference, are averaged over each run and followed from one question to the next, to verify the thermal state of the hardware. The results are also broken down by the 14 subjects of the benchmark. Since a large model consumes more energy than a small one on every question, the energy of a subject is not compared in joules, but as a proportion of the energy per question of the whole run: a value of 1.5 means that the questions of that subject consumed 50% more energy than the average question of the same run. Each subject is then summarized by the median of this proportion among the instruct runs and among the reasoning runs.

A model that answers at random still attains the chance level, which the ratio counts as correct answers. The results that depend on the intelligence efficiency are therefore verified with the **accuracy above the chance level**, in which the chance level is subtracted from the accuracy of every model run.

4.6.2 Trends over Time

The models are compared within their size category, and the trends are estimated on the instruct runs, which are available throughout the period, whereas reasoning runs only appear from 2025 onwards. The yearly change of a metric is estimated by fitting a straight line to the values of the model runs against their release date, and is given by the slope of that line. For the accuracy, the slope is a number of percentage points gained or lost per year. The other metrics, such as the energy per question, change in proportion to their value and span a much wider range, so they are fitted on a logarithmic scale, on which a constant slope corresponds to a constant proportional change. Their yearly change is therefore expressed as a

factor or a percentage: a factor of 2.6 per year means that the metric is multiplied by 2.6 each year, and a change of −38% per year that it falls to 62% of its value each year.

Since each size category contains only 10 to 15 instruct runs, a fitted line can show a noticeable slope by chance alone. A change is therefore only described as an increase or a decrease when the metric is significantly associated with the release date, according to **Spearman's rank correlation** ρ [211]. This coefficient ignores the values themselves and considers only the order of the model runs, comparing their ranking by release date with their ranking by the metric. It equals 1 when every model run has a higher value than all those released before it, −1 when every model run has a lower value, and is close to 0 when the two rankings are unrelated. The association is significant when its p-value is below 0.05, that is, when a correlation at least as strong would arise by chance in fewer than 5% of the cases if the metric did not change over time. The test and the line therefore have separate roles: the test establishes whether a change occurred, and the slope of the line expresses its magnitude.

The releases are also compared in pairs. Each instruct run is paired with the preceding instruct release of the same family, size category and architecture, and the change from the first to the second is expressed in accuracy, in energy per question and, where the second is more accurate and consumes more energy, in the additional energy per additional point of accuracy.

4.6.3 Accuracy at a Fixed Energy and Energy at a Fixed Accuracy

A model run lies on the **Pareto front** when no other model run is simultaneously more accurate and less costly in energy per question. The front associated with a given date considers only the model runs released up to that date.

To separate the evolution of accuracy from that of energy, one of the two quantities is held constant. For a given energy budget per question, the analysis tracks the best accuracy among the model runs released so far that remain within it. For a given accuracy target, it tracks the lowest energy per question among the model runs that attain it. Four budgets (0.5, 1, 2 and 5 kJ per question) and four targets (40%, 50%, 60% and 70%) are considered, corresponding to round values that span the range of the instruct runs.

The front of all the model runs is also used to assess how the cost of accuracy varies with its level. Between each model run of the front and the next, the *marginal cost* of accuracy is the additional energy per question divided by the additional points of accuracy. The *point of greatest curvature* of the front is the model run that lies farthest from the straight line joining its two ends, with the energy on a logarithmic scale and both axes scaled to the range of the front.

4.6.4 Comparison of Two Model Runs

Since every model run answers the same questions, the accuracy of two model runs is compared question by question, using the **exact McNemar test** [212]. The questions that both runs answer correctly, or that both answer incorrectly, reveal nothing about which of the two is more accurate, so the test considers only the questions that one run answers correctly and the other does not. If the two runs were equally accurate, each of them would be expected to win about half of these questions. The p-value of the test is the probability that a split at least as uneven as the one observed would arise by chance between two equally accurate runs. For example, if each of two runs answers correctly some questions that the other misses, 30 for the first and 15 for the second, this probability is 3.6%. As each run comprises several repetitions, a question is considered to be answered correctly by a run when it is correct in more than half of them. A difference is described as significant when the p-value is below 0.05.

When an instruct run is compared with a reasoning run, a portion of the responses reached the output limit (Section 4.4.1), and the outcome depends on how these responses are treated. The difference in accuracy is therefore reported under three views:

- *as recorded*, using the accuracy of each run over the 372 questions, in which a response that reached the limit is scored according to the answer that could be extracted from it (Section 4.5.1);
- *strict*, in which every response that reached the output limit is counted as incorrect, in both runs;
- on the *finished questions*, that is, the questions whose reasoning response did not reach the limit in any repetition.

4.6.5 Relation with the Characteristics of the Models

Four analyses relate the results to the characteristics of the models:

- The association between any two quantities across the model runs is described with the same coefficient, Spearman's rank correlation, referred to simply as the rank correlation.
- The energy per token is related to the size of a model, measured by its number of active parameters, by means of a power law, fitted as a straight line on a logarithmic scale. To verify that it generalizes to models not used in the fit, each model is predicted from a fit that excludes it, and subsequently from a fit that excludes its entire family. The energy per token *at equal size* is the measured value divided by the value that the power law gives for a model of that size.

- The share of the variation of a metric that is explained by the size and by the operating mode corresponds to the R^2 of a linear model of that metric on a logarithmic scale, with the logarithm of the active parameters and the operating mode as predictors.

- The families did not release their models at the same dates or in the same sizes, so that a direct comparison of their averages would confuse the family with the period and the size of its models. They are therefore compared by means of a linear model of each metric, fitted to the instruct runs, which predicts the metric from the release date, the size category and the family together. The *effect* of a family is then the difference between its model runs and the average of the families, once the release date and the size category are taken into account. As for the trends, the accuracy is modelled directly and the remaining metrics on a logarithmic scale. Whether the families differ at all is decided by an F-test, which compares this model with the same model without the family. Any additional predictor improves the fit slightly, and the p-value of the test is the probability of an improvement at least as large as the one observed if the families made no difference.

Chapter 5

Implementation

This chapter describes how the experimental pipeline implements the methodology defined in Chapter 4. Whereas the previous chapter states each decision and the reasons behind it, this chapter shows how that decision is realized in software: the components involved, the code whose logic is not evident from its description, and the problems that had to be solved along the way. Each section therefore refers to the part of the methodology that it implements. The complete implementation is publicly available in the repository of the experiment [39], and the material that is consulted rather than read, such as the listings that show a configuration or a call to a library, is gathered in Appendix B.

The chapter begins with the architecture of the pipeline (Section 5.1) and then follows its stages: the notebook that selects the models (Section 5.2), the orchestration of the experiment (Section 5.3) and the script that executes and measures each run (Section 5.4). It ends with the means by which the experiment can be reproduced and extended (Section 5.5), and with the corrections that were made after its main execution (Section 5.6).

5.1 Architecture of the Pipeline

The pipeline consists of four stages, each of which reads what the previous one produces (Figure 24):

1. **Model selection** (`model-selection.ipynb`, Section 5.2). The notebook collects the models of the selected collections from the HF Hub, together with their metadata, and writes the pool of 95 models to `models.csv` and the 53 model runs to `selected_models.csv`.

2. **Orchestration** (Experiment Runner, configured by `RunnerConfig.py`, Section 5.3). From the selected models, it creates the run table, downloads the models, collects their sampling parameters in `model_configs.json`, and then launches the runs, one at a time.

3. **Evaluation** (`evaluate_from_local.py`, Section 5.4). Each run loads one model with its configuration and answers the sample of MMLU-Pro with vLLM, while its energy is measured through Intel RAPL and NVML, and writes its results to a directory of its own inside `results/`.

4. **Post-processing and analysis** (`post-processing.ipynb`, `00-dataset.ipynb` and the analysis notebooks `01` to `06`). The recorded results are corrected (Section 5.6), and the tables and the charts of the analysis are built from them.

Each stage realizes one part of the methodology: the selection of the models (Section 4.2), the design of the experiment (Section 4.3), the execution of a run (Section 4.4) and the analysis (Section 4.6). The second and the third stages are executed on the server (Section 4.1.1), since they require its GPU, whereas the first and the fourth do not depend on it. The two stages that are executed on the server are implemented as Python scripts, and the other two as Jupyter notebooks, in which the code is kept together with its outputs.

Two further executions complement the main one, and are defined as separate configurations of the same framework. The supplementary execution of the three Ministral 3 instruct models with 16-bit

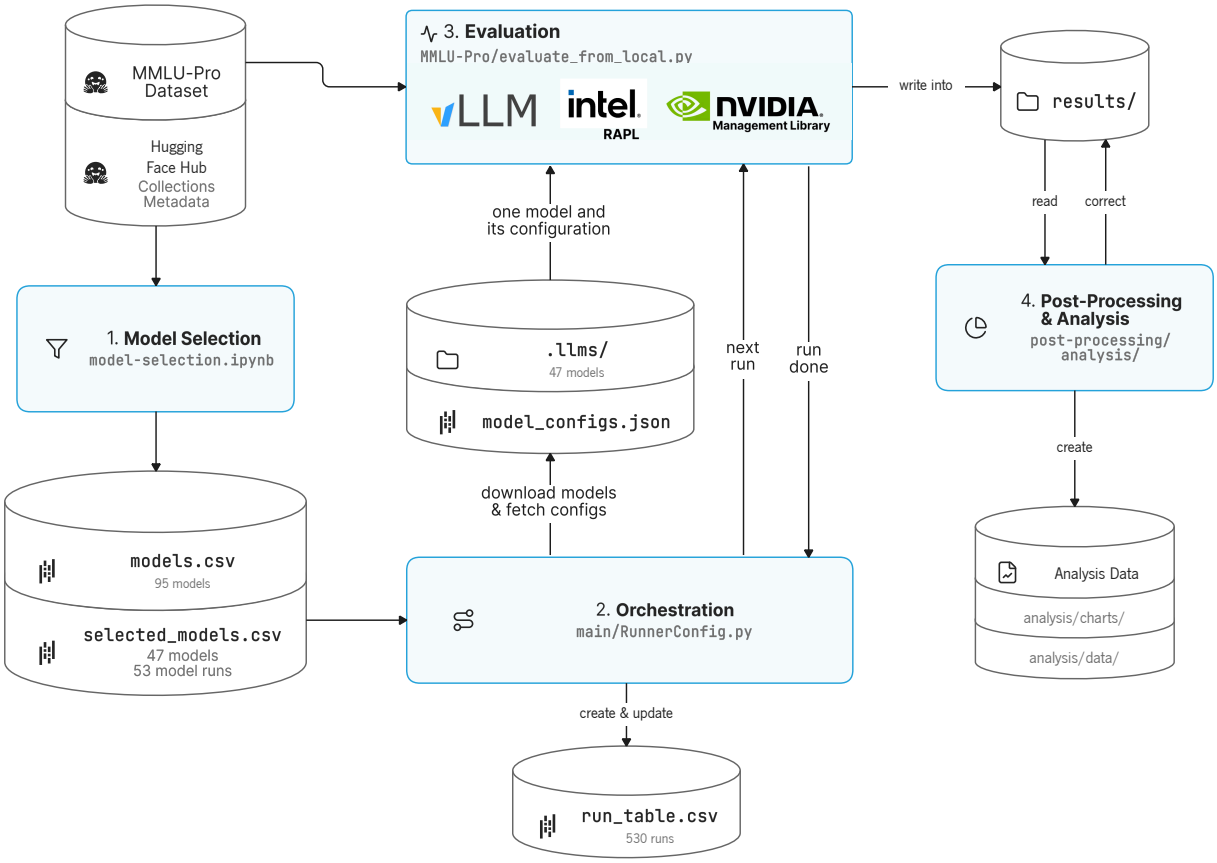


Figure 24: Architecture of the experimental pipeline: its four stages and the files through which they communicate.

```
eco-evolution-of-llms/
├── model-selection/        # stage 1: notebook and its output files
├── experiment-runner/      # code of the Experiment Runner framework
├── experiments/            # stage 2: configuration of each execution
├── dependencies/MMLU-Pro/  # stage 3: evaluation script, summary of a run
├── utils/                  # download and configuration of the models
├── results/                # outputs of each execution, one directory per run
├── post-processing/        # stage 4: corrections of the recorded data
├── analysis/               # stage 4: notebooks, tables and charts
└── requirements.txt        # dependencies and installation instructions
```

Listing 2: Structure of the repository of the experiment (main directories).

weights (Section 5.6) is defined in `experiments/mistral-bf16/` and writes its results to `results/mistral-bf16/`, and the measurement of the idle consumption of the server (Section 4.1.3) is defined in `experiments/idle-consumption/` and writes its results to `results/idle_consumption/`. Both are read by the fourth stage.

The repository of the experiment [39] is organized by stage, as shown in Listing 2. It includes the code of Experiment Runner, in the `experiment-runner/` directory, and an execution is launched by passing its configuration file to the framework (`python experiment-runner/ experiments/main/RunnerConfig.py`). Three modules are shared between components, so that the logic that they contain has a single definition: `utils/models.py`, with the download and the configuration of the models, `run_summary.py`, with the computation of the summary of a run, which is used by the evaluation script and by the post-processing notebook, and `analysis/common.py`, with the paths, the constants and the chart style of the analysis notebooks.

5.2 Model Selection Notebook

The selection of the models (Section 4.2) is implemented in a single Jupyter notebook, `model-selection/model-selection.ipynb`, which executes the six stages of Figure 16 in order. The rules of the selection are declared as constants of the notebook: the list of collections, the keywords and the types of the filter, and the explicit lists of models. This section describes how the models are collected and filtered, how their metadata is obtained, how their operating mode is identified, and how the pool is classified and reduced to the file from which the experiment is built.

5.2.1 Collection and Filtering

The HF Hub is accessed through its API, with the `huggingface-hub` Python library [213]. For each of the collections of Table 11, the `get_collection` function of the library returns the list of models that the collection contains, to which the notebook appends the models that were added to that collection explicitly (Section 4.2.1).

Each model is then examined by a single function, which either rejects it or returns its metadata. The rules that require no access to the Hub are applied first: the explicit list of excluded models and the keywords of Table 12, which are searched in the identifier of the model without distinction of case. Only the models that pass these rules are requested from the Hub, through the `model_info` method of the `HfApi` class, which returns the details of a model given its identifier. The type and the tags of the model, which are part of these details, are then compared with the types that correspond to the generation of images or audio. The size limit is applied once the parameter count of the model has been obtained. For every rejected model, the notebook prints the rule that rejected it, so that its output records the reason for each exclusion.

5.2.2 Sources of the Metadata

Table 16 lists the sources from which each characteristic of a model (Section 4.2.3) is obtained. When several sources are listed, they are tried in order, and the next one is only used when the previous one fails. Most of this information is obtained from the details returned by `model_info`. This is the case of the release date, which is the date on which the repository of the model was created on the Hub (`created_at`).

No single source reliably provides the number of parameters of every model, so four sources are used for the total:

Metadata	**Sources, in Order of Preference**
Release date	Creation date of the model's repository
Total parameters	Safetensors metadata → model configuration → model name → manual input
Active parameters	Model name → manual input
Architectural paradigm	Model name → repository tags → manual list
Operating mode	Model name → chat template → manual list
Maximum context window	Model configuration → manual input

Table 16: Metadata collected for each model and the sources used to obtain it.

1. **Safetensors metadata.** Safetensors is the file format developed by Hugging Face for storing model weights, and the default format on the Hub [214]. Its files include JSON metadata with the number of parameters stored in each numerical format, which `model_info` returns directly, and the total is obtained by adding these counts. The metadata also contains a single total, which is not used because it does not always match the weights: for the two smallest Ministral 3 models, it exceeds the number of parameters in the weights by exactly the size of the embedding matrix that these models share between their input and their output. This is the preferred source, but it is not available for older models, uploaded before the format became standard, or for models stored in other formats.

2. **Model configuration.** The `config.json` file of a model is its structural "blueprint": instead of the weights, it describes the architecture, such as the number of layers and their dimensions. The `accelerate` library [215] can use this file to create an *empty model*, with the right structure but no weights, from which the number of parameters can be counted without downloading anything else (Listing B.1, in the appendix). This fails when a repository has no standard configuration file, or relies on custom code that cannot be executed.

3. **Model name.** Many developers include the parameter count in the name of the model, as discussed below.

4. **Manual input.** If all the automatic sources fail, the notebook asks the user for the value, which must be looked up manually. The answers are saved to a `manual_model_data.json` file, so that each value is only requested once.

The identifier of a model, in the format `organization/model_name`, often reveals a great deal about it. Besides the parameter count, it can indicate the operating mode and the architectural paradigm of the model. Developers use a variety of conventions, such as:

- `-270m-it` (270 million total parameters; instruct)
- `-4B-Chat` (4 billion total parameters; instruct)
- `-1.5B-Instruct` (1.5 billion total parameters; instruct)
- `-4B-Thinking` (4 billion total parameters; reasoning)
- `-30B-A3B` (30 billion total parameters, 3 billion active)

- `-MoE-A2.7B` (2.7 billion active parameters; MoE)
- `-E2B-it` (2 billion effective parameters; instruct)

These conventions are recognized with a set of Regular Expression (Regex) patterns, which extract the total and the active parameter counts whenever the name contains them.

A model is placed in the sparsely activated group when at least one of three conditions is met: its name follows one of the conventions that indicate an active or effective parameter count, which only these models use, its repository tags mention "MoE", or it is part of a short list of models that were identified manually because neither their name nor their tags reveal their architecture. The number of active parameters of such a model is extracted from its name or, when the name does not include it, entered manually. In the files produced by the notebook, this group is identified by a column named `is_moe`, since it was originally created for MoE models only.

The maximum size of the context window is read from the configuration of the model, where it appears under different names depending on the architecture (`max_position_embeddings`, `n_positions`, `seq_length` or `max_sequence_length`). If none of them is found, the value is entered manually.

5.2.3 Identification of the Operating Mode

As shown in Listing 3, a model is identified as a reasoning model (Section 4.2.3) in one of three ways:

1. Its name contains a reasoning-related keyword, such as "reasoning" or "thinking".
2. Its chat template contains tags used to delimit reasoning, such as `<think>` or `<reasoning>`. The chat template is usually written in Jinja, and is stored in the tokenizer configuration of the model.
3. It is part of a manual list, used for models whose reasoning nature is not visible in their name or chat template.

Dual-mode models are identified by the parameter that selects their mode, `enable_thinking`. The notebook searches for this parameter in the chat template and in the tokenizer configuration file of the model, since a template that supports the parameter must mention it. The result is recorded in three columns: `is_thinking` and `is_instruct`, which are both true for a dual-mode model, and `supports_enable_thinking`, which is true for these models only.

```
THINKING_MODELS = {"deepseek-ai/DeepSeek-R1-0528-Qwen3-8B"}
THINKING_WORDS = {"reason", "reasoning", "think", "thinking"}

def check_is_thinking(model_id):
  if model_id in THINKING_MODELS:
      return True

  if any(word in model_id.lower() for word in THINKING_WORDS):
      return True

  tokenizer = AutoTokenizer.from_pretrained(model_id)
  if tokenizer.chat_template:
      thinking_tags = ["<think>", "<reasoning>", "<|think|>", "<|channel>thought"]
      if any(tag in str(tokenizer.chat_template) for tag in thinking_tags):
          return True

  return False
```

Listing 3: Process of identifying a reasoning model (simplified).

5.2.4 Classification and Selection

The remaining stages operate on a table with one row per model. The prioritization of instruction-tuned variants (Section 4.2.4) removes every model for which the table also contains a model with the same identifier followed by one of the suffixes. The result is saved to the `models.csv` file, which holds the 95 models of the pool and their metadata.

The limits of the size categories (Section 4.2.5) are computed with the `jenks_breaks` function of the `jenkspy` library [216], over the total parameter counts of these 95 models, and each model is assigned to the category whose range contains its parameter count.

The final selection (Section 4.2.6) consists of three operations. The models of the two excluded categories are removed. The remaining ones are sorted by decreasing parameter count and by decreasing release date, and only the first model of each combination is kept (Listing B.2, in the appendix). In this operation, the operating mode is represented by the `is_thinking` column, which is true for reasoning and for dual-mode models alike, so that the reasoning and the dual-mode models of the same release, size category and architectural paradigm form a single combination. Finally, each dual-mode model that was selected is replaced by two rows, one per mode, in which the suffix of the mode is added to the identifier of the model and the two columns of the operating mode are set accordingly.

The result is saved to the `selected_models.csv` file, which contains the 53 model runs of the experiment and whose columns are described in Table 17. This is the file from which the run table of the experiment is built (Section 5.3), and from which the analysis obtains the characteristics of each

Columns	Content
`collection` `collection_name`	Identifier and title of the collection of the model, which corresponds to its release.
`model`	Identifier of the model on the Hub, followed by the suffix of the mode in the two runs of a dual-mode model.
`total_parameters` `activated_parameters`	Total and active parameter counts, in billions.
`max_model_length`	Maximum size of the context window, in tokens.
`is_moe`	Whether the model is sparsely activated.
`is_thinking` `is_instruct`	Operating mode of the model run.
`supports_enable_thinking`	Whether the model is a dual-mode model.
`release_date`	Date on which the repository of the model was created.
`category`	Size category of the model.

Table 17: Columns of the `selected_models.csv` file, which has one row per model run.

model. The notebook also generates the charts that describe the pool and the final set in the methodology (Figure 17, Figure 18, Figure 19 and Figure 20).

5.3 Experiment Orchestration

The experiment is defined and executed with the Experiment Runner framework [121], which structures the entire workflow, ensuring that all the runs are executed in the same manner and that their data is collected reliably. This section describes how the design of Section 4.3 is expressed in the framework: how the run table is built, how the environment and the models are prepared, and how each run is launched and its results collected. Figure 25 illustrates the complete workflow, which is implemented in `experiments/main/RunnerConfig.py`.

5.3.1 Structure of an Experiment

In Experiment Runner, an experiment is defined by a single class, `RunnerConfig`. The framework controls the overall flow of the experiment and calls a specific method of this class at each stage, such as before the experiment starts, when a run starts, or when the data of a run has to be collected. Defining an experiment therefore consists in implementing the methods of the stages that matter, and leaving the others empty. This experiment required the four methods listed in Table 18. The remaining ones, which the framework provides for starting and stopping measurements and for interacting with the system under test, were left

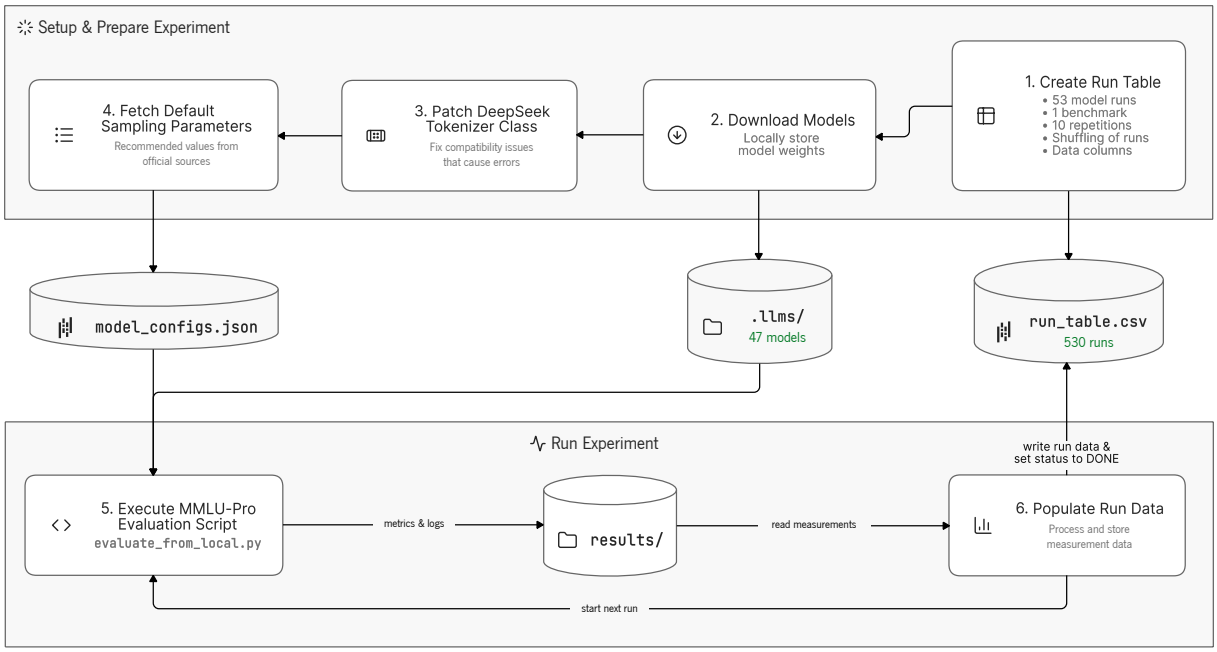


Figure 25: Experiment orchestration workflow.

unmodified, since the measurements are taken by the evaluation script itself (Section 5.4), and not by the framework.

5.3.2 Run Table

In Experiment Runner, the design of an experiment is expressed as a **run table**. The framework first computes the *execution matrix*, which contains every combination of the factors of the experiment (here, the models and the benchmark), repeated the requested number of times. The run table expands this matrix into one row per run, in the order in which the runs are to be executed, and adds columns for the status of each run and for the metrics that it produces. It therefore serves both as the plan of the experiment and as the record of its progress and of its results.

Method	Use in This Experiment
`create_run_table_model()`	Defines the run table: the runs to be executed, their order and the metrics to be collected.
`before_experiment()`	Prepares the environment, once, before the first run: downloads the models, patches their tokenizers and collects their sampling parameters.
`start_run()`	Executes a run, by launching the evaluation script as a separate process and measuring its duration.
`populate_run_data()`	Reads the results of a run and returns them to the framework, which stores them in the run table.

Table 18: Methods of the `RunnerConfig` class implemented for the experiment.

```
def create_run_table_model(self) -> RunTableModel:
  models = pd.read_csv(ROOT_DIR / "model-selection/selected_models.csv")
  model = FactorModel("model", models["model"].unique().tolist())
  benchmark = FactorModel("benchmark", ["MMLU-Pro"])
  self.run_table_model = RunTableModel(
      factors=[model, benchmark],
      exclude_combinations=[],
      data_columns=[
          "runtime_seconds",
          ...
      ],
      repetitions=10,
      shuffle=True,
  )
  return self.run_table_model
```

Listing 4: Creating the run table model.

The run table is defined through the parameters of the `RunTableModel` constructor, as shown in Listing 4.

Each parameter corresponds to one part of the design:

- `factors` defines the two factors of the experiment. The models are read from the CSV file produced by the model selection notebook (`selected_models.csv`, Section 5.2.4), and the benchmark is fixed to MMLU-Pro.
- `exclude_combinations` allows specific combinations of factors to be left out. It is empty, since every model is evaluated on the benchmark.
- `data_columns` lists the metrics recorded for each run, which are those of its summary (Section 4.5.2).
- `repetitions` sets the 10 repetitions of each model run.
- `shuffle` randomizes the order of the runs. The shuffle uses a fixed random seed, so that the same order of execution can be reproduced.

Each run is identified by the index of its model and by its repetition number, from `run_0_repetition_0` to `run_52_repetition_9`. These identifiers are also the names of the directories in which the results of each run are stored.

5.3.3 Preparation of the Environment

Before any run starts, everything on which the runs depend has to be in place. This preparation is handled by the `before_experiment()` method, which is executed only once and performs three tasks.

The first is the download of the models. The repository of every selected model, which contains its weights, its tokenizer and its configuration files, is downloaded with the `snapshot_download` function of the `huggingface_hub` library and stored locally under a `.llms/` directory, the models that are already present being skipped. Downloading everything in advance means that no run depends on the network, which avoids delays and failures in the middle of the experiment.

The second is the correction of four models that could not be loaded during the initial tests, because their `tokenizer_config.json` file declared the wrong tokenizer class. Recent versions of the `vllm` and `transformers` packages verify that the declared class matches the tokenizer files, and refuse to load the model when it does not. Since the correction amounts to a single line, these models were patched immediately after being downloaded, instead of being excluded from the study. Three of them are based on Qwen and use its tokenizer, so their entry was replaced with the correct class. For `deepseek-llm-7b-chat`, the entry was replaced with a generic class, which loads the tokenizer directly from the files of the model (Table B.2, in the appendix).

The third is the collection of the recommended sampling parameters (Section 4.4.1). The baseline is created automatically: the parameters are read from the configuration files of each model (`generation_config.json` or, when it does not exist, `config.json`), any missing value is filled in with the default of vLLM, and the result is saved to the `model_configs.json` file, which is then reviewed manually against the model cards. Models that already have an entry in the file are never overwritten, so that the manual corrections are preserved when the experiment is restarted. The file contains a `default` entry with the settings shared by every model, and one entry per model with its recommended sampling parameters, which override the defaults (Listing B.3, in the appendix). Dual-mode models have one entry for each operating mode.

5.3.4 Launching a Run and Collecting Its Results

With the environment prepared, Experiment Runner goes through the run table one row at a time. Each run is handled by two methods: `start_run()` executes it, and `populate_run_data()` collects its results.

A run consists in executing the evaluation script (`dependencies/MMLU-Pro/evaluate_from_local.py`), to which the decisions of the methodology that depend on the model are passed as arguments (Listing B.4, in the appendix). Table 19 gives the values of these arguments for each

type of model: the size of the context window, the thinking budget, the number of few-shot examples and, for dual-mode models, the operating mode (Section 4.4.1 and Section 4.4.3). The remaining arguments identify the model, the directory in which the outputs of the run are written, a record file shared by all the runs and the `model_configs.json` file, and enable the sampling of the benchmark (Section 4.4.2).

Argument	Instruct Models	Reasoning Models	Dual-mode Models
`--max_model_length`	Model's context window, up to 10,240 tokens		
`--max_thinking_tokens`	0	4,096	Depends on the mode
`--ntrain`	5	0	Depends on the mode
`--enable_thinking`	Omitted	Omitted	`True` or `False`

Table 19: Values of the model-dependent arguments of the evaluation script, according to the type of model.

The script is executed as a **separate process**, which is what gives every run a clean state (Section 4.4, and Listing B.5 in the appendix). Its output is saved to an `experiment.log` file inside the directory that Experiment Runner creates for the run, and the duration of the process, which is the run time, is measured and saved to a `runtime.txt` file in the same directory.

When the script finishes, the directory of the run contains the `runtime.txt` file and a `run_data.json` file written by the script, with the summary of the run (Section 5.4.6). The `populate_run_data()` method reads both files and returns their values to Experiment Runner, which saves them in the corresponding row of the run table. Once all the runs are complete, the run table contains the results of the whole experiment, with one row per run.

5.4 Evaluation Script

Every run of the experiment is carried out by a single script, `evaluate_from_local.py`, adapted from the official evaluation script of MMLU-Pro (Section 4.4). It executes the five phases of a run shown in Figure 22, which this section follows in the same order.

5.4.1 Loading and Configuring the Model

The settings of a run are read from the `model_configs.json` file created during the preparation of the environment (Section 5.3). The engine is then created with the parameters of Table 20, which implement the configuration of the engine described in Section 4.4.1 (Listing B.6, in the appendix).

Parameter	Value	Decision That It Implements
`max_num_seqs`	1	One request at a time: a batch size of exactly one.
`enable_prefix_caching`	`False`	No reuse of computation between prompts.
`disable_log_stats`	`False`	Internal metrics of the engine enabled.
`enforce_eager`	`False`	Optimized execution: CUDA Graphs remain enabled.
`gpu_memory_utilization`	0.90	Share of the VRAM that the engine may use, the default of vLLM.

Table 20: Parameters of the vLLM engine that implement the configuration described in the methodology.

The base output limit is the `max_new_tokens` setting of the configuration file, to which the thinking budget is added to obtain the limit of generated tokens of the run. The stop conditions are obtained from the tokenizer of each model, from which the script reads the end-of-sequence and end-of-turn tokens (such as `<|im_end|>` or `<|eot_id|>`), skipping those that are related to reasoning. For instruct models, the strings `Question:` and `\nQuestion` are also used as stop strings.

5.4.2 Sampling and Prompt Construction

The benchmark is sampled when the script receives the `--sample` flag (Section 4.4.2), and the prompts of the sampled questions are then built (Section 4.4.3). To determine whether a model supports a system role, the script attempts to apply the chat template of the model to a short conversation that contains a system message, and places the task instruction in a system message only when this succeeds. The list of messages is converted into the final prompt by the `apply_chat_template` method of the tokenizer. For dual-mode models, the `enable_thinking` flag is passed to the template at this point, which is what switches the model between its instruct and its reasoning mode. The two adjustments of the Ministral 3 reasoning models are stored in `model_configs.json`: their system prompt (Listing B.7, in the appendix), and the `[THINK]` tag that is appended to the end of their prompts.

5.4.3 Warmup and Thermal Control

After the five warmup inferences (Section 4.4.4), the script enters the loop of Listing 5, which heats or cools each component until both are within their tolerance.

Each component is handled independently, so that one can be heated while the other cools down, as Table 21 summarizes.

The amount of heating applied in each step is estimated, and not calculated exactly. For the CPU, the script runs busy arithmetic loops on every hardware thread, under the assumption that each burst of 5

```
loop:
    read gpu_temp and cpu_temp

    if gpu_temp in [54, 56] and cpu_temp in [33, 37]:
        break                       # baseline reached

    if gpu_temp < 54:  heat GPU     # run the warmup prompt again
    if cpu_temp < 33:  heat CPU     # busy arithmetic on every thread

    if gpu_temp > 56:  cool GPU     # rest
    if cpu_temp > 37:  cool CPU     # rest
```

Listing 5: Pseudo-code of the thermal-regulation loop. Each component is checked independently, so heating and cooling can happen at the same time.

State	Action
GPU & CPU Below Target	Heat GPU & CPU
Only GPU Below Target	Heat GPU, Maintain CPU
Only CPU Below Target	Maintain GPU, Heat CPU
GPU & CPU Above Target	Rest GPU & CPU
GPU Above, CPU Below	Rest GPU, Heat CPU
GPU Below, CPU Above	Heat GPU, Rest CPU

Table 21: Dynamic actions of the thermal-regulation loop according to the state of each component.

seconds raises the temperature by about 2 °C. For the GPU, it uses the calibration of the warmup: the increase of temperature per prompt gives the number of prompts that are needed, and the estimate is updated after every step. These estimates are not exact, but they approach the target in a few steps, and the tolerance absorbs the remainder.

5.4.4 Measured Inference

The cumulative counters are read by a single function, which returns the three readings at once. The counter of the GPU is obtained from NVML, through the `nvmlDeviceGetTotalEnergyConsumption` function of the `pynvml` library. Those of the processors and of their memory are read from the `powercap` interface of the Linux kernel (`/sys/class/powercap/intel-rapl`), in which each processor appears as a domain with its own counter and with a sub-domain for its memory. The script locates these domains when it starts, and stops with an error if it finds none, since unreadable counters would otherwise be recorded as an energy of zero.

Listing 6 shows the **measurement window**. The counters and the clock are read immediately before and immediately after a single call to `llm.generate`, which only returns once the response

has been fully generated, so that the window always contains the whole inference. The calls to `torch.cuda.synchronize()` ensure that no work remains pending on the GPU when the counters are read.

```
for prompt in flat_inference_prompts:
    torch.cuda.synchronize()
    inference_start = time.time()
    cpu_start, dram_start, gpu_start = get_status()

    output = llm.generate(prompt, sampling_params, use_tqdm=False)

    torch.cuda.synchronize()
    inference_end = time.time()
    cpu_end, dram_end, gpu_end = get_status()

    inference_cpu_joules = rapl_delta(cpu_start, cpu_end, CPU_RAPL_MAX_RANGES) / 1e6
    inference_dram_joules = rapl_delta(dram_start, dram_end, DRAM_RAPL_MAX_RANGES) / 1e6
    inference_gpu_joules = (gpu_end - gpu_start) / 1000.0
    ...
```

Listing 6: The measurement window, opened and closed around a single `llm.generate` call.

The differences of the RAPL counters are computed by a function that handles their limited range (Section 4.4.5). The range of each counter is exposed by the kernel in a `max_energy_range_uj` file, next to the counter itself. When a counter restarts during an inference, its final reading is lower than its initial one and their difference is negative, in which case the function adds the range of the counter to the difference, recovering the correct energy. The counter of NVML is 64 bits wide, and needs no such correction.

The metrics of the inference engine are obtained through `llm.get_metrics()`, whose values are cumulative since the engine was started. To isolate the prompt that has just been processed, the script takes a snapshot of these metrics after every prompt and subtracts the previous snapshot from it, which yields the number of prompt and of generated tokens, and the duration of the prefill and decode phases, of that prompt alone. The first snapshot is taken after the warmup, whose prompts are therefore not included. The temperatures of the GPU and of the CPU are read immediately after the window is closed.

5.4.5 Processing of the Responses

The answers are extracted in a second pass over the outputs, performed after the last inference (Section 4.5.1). The reasoning block of a response is located with a regular expression that covers

the delimiters used by the models under study: `<think>`, `<reasoning>`, `<|channel>thought` and `[THINK]`, with their respective closing forms. The opening delimiter is optional in this expression, because many chat templates already insert it at the end of the prompt, so that the response only contains the closing one. The closing delimiter is required, since it is what marks the end of the reasoning. The text of the block is then tokenized with the tokenizer of the model, without special tokens, to obtain the number of thinking tokens.

The three patterns of the answer extraction are implemented as three functions that are tried in sequence, each of them being called only when the previous one finds no match. When the last one also fails, no prediction is stored for the response, and a random option is drawn when the accuracy is computed, with the random number generator that is seeded when the script starts.

5.4.6 Outputs of a Run

Each run produces the directory shown in Listing 7. Experiment Runner creates the directory and writes the run time and the log of the script (Section 5.3), and the script writes everything else.

The files of each subject contain one record per question. A record keeps the original fields of MMLU-Pro, such as the question, its options and the correct answer, and adds the complete response of the model, its prediction and the values of Table 15 (Listing B.8, in the appendix). Storing the complete responses makes it possible to verify the extraction of the answers afterwards, and to analyse the responses in other ways.

The summary of the run (Section 4.5.2) is written to `run_data.json`, whose fields are listed in Table 22. It is computed by a function kept in a module of its own, which is shared by the evaluation script and by the post-processing notebook, so that the summary has a single definition and can be recomputed from the records of a run (Section 5.6). This file is what Experiment Runner reads, in its

```
results/main/
├── run_0_repetition_0/
│   ├── experiment.log          # output of the evaluation script
│   ├── runtime.txt             # total runtime, measured by Experiment Runner
│   ├── run_data.json           # aggregated results of the run
│   ├── log/  summary/          # detailed log and accuracy summary
│   └── <model>/CoT/all/
│       ├── biology.json        # per-prompt records, one file per subject
│       ├── ...
│       └── psychology.json
├── ...
└── run_52_repetition_9/
```

Listing 7: Output directory structure of the experiment.

Group	Fields
Run	`model`
Accuracy	`avg_accuracy`

Energy (J)	`total_inference_gpu_joules`
	`total_inference_cpu_joules`
	`total_inference_dram_joules`
	`total_inference_joules`

Tokens	`total_tokens`
	`total_prompt_tokens`
	`total_generated_tokens`
	`total_thinking_tokens`
	`avg_generated_tokens_per_inference`
	`median_generated_tokens_per_inference`

Responses	`truncated_responses` `unparsed_responses`

Timing (s)	`total_inference_time_seconds`
	`total_prefill_time_seconds`
	`total_decode_time_seconds`
	`avg_time_per_inference_seconds`
	`median_time_per_inference_seconds`

Throughput (tokens/s)	`avg_prefill_tokens_per_second` `avg_decode_tokens_per_second`

Power (W)	`avg_total_power_watts`
	`avg_cpu_power_watts`
	`avg_dram_power_watts`
	`avg_gpu_power_watts`

Temperature (°C)	`avg_gpu_temperature_celsius` `avg_cpu_temperature_celsius`

Efficiency (GPU energy)	`avg_gpu_joules_per_inference`
	`median_gpu_joules_per_inference`
	`gpu_joules_per_generated_token`
	`generated_tokens_per_gpu_joule`
	`accuracy_per_gpu_joule`
	`gpu_joules_per_correct_answer`

Table 22: The fields of the aggregated `run_data.json` file, grouped by the kind of quantity they report.

`populate_run_data()` method, to fill in the run table, and it is the starting point of the analysis (Section 4.6).

The efficiency fields correspond to the metrics defined in Section 4.5.3: `avg_gpu_joules_per_inference` and `median_gpu_joules_per_inference` to the energy per question and to its median, `gpu_joules_per_generated_token` and `generated_tokens_per_gpu_joule` to the energy per token and to the tokens per joule, and `accuracy_per_gpu_joule` and `gpu_joules_per_correct_answer` to the accuracy per joule and to the energy per correct answer.

5.5 Reproducibility and Extension

The pipeline can be executed again, either to reproduce the results of this study or to evaluate models released after it. This section describes what makes both possible: a fixed environment and fixed seeds, the preservation of the recorded data, and the steps through which new models are added.

5.5.1 Reproducing the Experiment

The versions of all the software components are fixed (Section 4.1.4). The Python environment is managed with the uv package manager [217], and the `requirements.txt` file of the repository lists all the required dependencies with their versions, together with the instructions to install them and to reproduce the environment. The same instructions are given in the README of the repository, which also describes the requirements of the hardware and how each stage of the pipeline is executed.

Four elements of the experiment depend on random numbers, and all of them use a fixed seed: the order of the runs in the run table (Section 5.3), the sample of questions of the benchmark (Section 4.4.2), the option that is attributed to a response from which no answer can be extracted (Section 4.5.1), and the sampling of the tokens by the inference engine (Section 4.4.1). The first three use the random number generator of Python, which is seeded with a constant when the configuration of the experiment and the evaluation script are loaded, and the last one uses the default seed of vLLM. An execution that is repeated with the same models therefore schedules its runs in the same order and presents the same questions to every model.

An execution that is interrupted does not have to be started again. Experiment Runner records the state of every run in the run table that it saves with the results, and, when the same command is launched again, it executes only the runs that are not marked as done. Before doing so, it verifies that the configuration of the experiment has not changed since the experiment was started, and asks for confirmation if it has.

The data is preserved at every stage. The results of the main execution are kept as they were recorded, and the data used in the analysis is derived from them by a single notebook (Section 5.6). The analysis is reproduced, in turn, by running the notebook that builds its tables and then the notebooks of each research question, every value reported in Chapter 6 being part of their outputs (Section 4.6).

5.5.2 Adding New Models

The evaluation of a new release requires no change to the logic of the pipeline, and proceeds through its four stages:

1. **Selection.** The identifier of the collection of the release is added to the list of collections of the model selection notebook, which is then run again (Section 5.2). Since the notebook collects all the models again from the HF Hub, and computes the limits of the size categories from the resulting pool, both the pool and the categories can change, and have to be compared with the previous ones.

2. **Orchestration.** Experiment Runner does not add runs to an experiment that is complete, since this would change its run table, and with it the numbering and the order of the runs whose results have already been recorded. The new models can therefore be executed in two ways. The first is an experiment of their own, whose configuration is a copy of that of the main experiment, restricted to the new models, as in the supplementary execution of this study (Section 5.1). The second is to move the results of the main experiment elsewhere and to execute it again from the start, with the complete list of models. In both cases, the new models are downloaded when the execution starts, and their entries are added to the file of sampling parameters without changing the existing ones, after which they should be reviewed against the model cards (Section 5.3).

3. **Evaluation.** The runs are executed by the same evaluation script, with the same arguments (Section 5.4). Their results are only comparable with the existing ones if they are obtained in the same environment (Section 4.1).

4. **Analysis.** The analysis notebooks are run again. When the new models are executed in an experiment of their own, the directory of its results and its number of repetitions are first added to the module that the notebooks share.

5.6 Corrections After the Experiment

The methodology and the previous sections of this chapter describe the pipeline in its final form. Part of it was corrected after the main execution of the experiment, when problems were found in the code or in the data that it had produced. This section brings those corrections together: what was found, which data it affected, and what was done about it (Table 23).

Two principles were followed in all of them. First, nothing was discarded: the code and the data of the main execution, as originally recorded, remain available in the repository of the experiment [39], under the `original-results` tag. Second, every change made to the recorded data is implemented in a single Jupyter notebook (`post-processing/post-processing.ipynb`), which explains and validates each step. Running it on the original data produces the data used in this study.

What was found	Data affected	What was done	Effect on the results
The overflow of the RAPL counters was handled with the wrong counter size	CPU and DRAM energy of 24,250 of the 197,160 inferences	Lost energy reconstructed; script corrected	None: the efficiency metrics use the GPU energy
Only the first of the two processors was read	CPU and DRAM energy of every inference	Cannot be recovered; reported as covering one processor; script corrected	None, for the same reason; stated as a limitation
The metrics per token counted the tokens of the prompts, and efficiency was computed on a partial total	Summary of every run, and the run table	Definitions revised; summaries recomputed from the per-prompt records	The metrics are those defined in Section 4.5.3
Three models of the selection were distributed with 8-bit weights	3 of the 53 model runs	Selection pipeline corrected; the three models executed again with 16-bit weights	The three model runs come from a supplementary execution, with 5 repetitions
The total parameter count reported by the HF Hub was wrong for one model	Parameter count of `Ministral-3-3B-Reasoning` (4.25B recorded, 3.85B in its weights)	Selection pipeline now adds the counts per numerical format	None: size category and selection unchanged
A model that generates text by diffusion had entered the collections	Model pool	Keyword added to the initial filtering	None: the pool has the same 95 models

Table 23: Corrections made after the main execution of the experiment.

5.6.1 Energy of the Processors and of Their Memory

Two of the corrections concern the energy that is read from the RAPL counters, and neither of them affects the analysis, since the efficiency metrics are based on the GPU energy, which was measured in full and needed no correction (Section 4.5.3).

The first is the handling of the overflow of the counters, which restart from zero after reaching their maximum, about 262 kJ on this server. The script used in the main execution assumed that this happened after 2^{32} µJ (about 4,295 J). The registers that hold the counters do have 32 bits, but they count in energy units that depend on the processor model, not in microjoules (Section 2.3.1) [96], and the range of this server corresponds to 2^{32} units of about 61 µJ. As a result, whenever an inference used more than 4,295 J of CPU energy, which happens for generations longer than roughly 50 seconds, the recorded value was too low by one or more "blocks" of exactly 4,295 J. Errors in the handling of these overflows are a known pitfall of measurements based on RAPL, and have affected many measurement tools [93].

Since the power of the CPU is very stable within a run (around 1% variation), the number of missing blocks can be recovered from the duration of the inference: each value was corrected by adding the number of blocks that brings it closest to the typical power of its run multiplied by the inference time.

The corrected values are never more than about 520 J away from this reference, while a wrong number of blocks would cause an error of over 3,700 J. The correction changed 24,250 of the 197,160 recorded inferences, almost all of them from reasoning models. Each corrected record keeps the value originally recorded, in a `rapl_correction` field, and the repository includes a log of every corrected value.

The second is the reading of the processors. The same version of the script only read the counters of the first of the two processors of the server, so that the energy of the second processor, and of the memory attached to it, was not recorded. This cannot be recovered from the collected data, only estimated, and it was decided not to mix estimated values with measured ones. The CPU and DRAM energy of the collected data therefore covers **one of the two processors**, and its total energy is the sum of what was measured (the GPU, the first processor and its memory), not the energy of the whole system.

5.6.2 Summary of a Run

The script used in the main execution computed its efficiency metrics on the total measured energy, and its metrics per token over all the tokens of a run, including those of the prompts. During the analysis, both choices proved to be misleading, for the reasons given in Section 4.5.3: the total covers only part of the system, and a metric over all tokens mostly reflects the length of the prompts. The definitions were revised, a few fields were added, and the summary of every run and the run table were recomputed from the per-prompt records, with the same function that the script now uses at the end of a run (Section 5.4.6). The accuracy of each run was not recomputed, and is the one originally recorded.

5.6.3 Selection of the Models

The remaining corrections concern the model selection notebook (Section 5.2). The first is that of the models distributed with 8-bit weights. The notebook excludes quantized variants through keywords in their names (Section 4.2.2). The three instruct models of the Ministral 3 release are distributed with 8-bit (FP8) weights under names that do not indicate it, so that they were selected and executed in that format, unlike every other model. This was noticed during the analysis, in their energy per generated token, which was 1.6 to 1.7 times lower than that of the reasoning models of the same release and size. The notebook was corrected to exclude these three repositories and to select their 16-bit versions, which Mistral publishes separately. The three models were then executed again with these weights, in a supplementary execution with the same configuration, and these are the runs used in the analysis. With 16-bit weights, their energy per generated token is within 0.5% of that of the reasoning models of the same size, which confirms that the difference came from the format of the weights. Since this execution used the corrected script, its

CPU and DRAM energy covers both processors, unlike that of the main execution. The 8-bit runs of the main execution are kept, and are only used to compare the two formats (Chapter 6).

The server was only available for a limited time, so that these three model runs have **5 repetitions** instead of 10. Fewer repetitions are acceptable here because these models run with a temperature of 0, following the recommendation of their creators to use a temperature below 0.1 [218], so that no random sampling is involved in the generation. The 3B and 8B models gave exactly the same responses in every repetition, and their GPU energy varied by at most 0.3% between repetitions. The 14B model did the same in four of its five repetitions, with a variation of 0.2%. In the remaining one, 205 of the 372 responses were different, the accuracy was 2.2 points lower and the GPU energy 1.4% higher. The analysis summarizes each model run by the median of its repetitions, so that this repetition does not change the values of the 14B model. Differences of this kind between repetitions also occur in the main execution, and are analysed in Chapter 6.

The second correction concerns the parameter count of a model of the same release. The HF Hub reported a total of 4.25B parameters for `Ministral-3-3B-Reasoning`, while its weights contain 3.85B, the same as the instruct model of that size. The notebook took this total directly from the metadata of the Hub, and now adds the counts that the metadata gives for each numerical format, as described in Section 5.2.2. The correction does not change the size category of the model, or the selection.

The third correction followed from the other two. Applying them meant running the notebook again, which collects the models again from the HF Hub. This time, the Gemma 4 collection contained a model that was not in the original pool: DiffusionGemma, which generates text by diffusion and not one token at a time [205]. Models of this kind are outside the scope of the study, so that a keyword was added to the initial filtering to remove them (Section 4.2.2). The pool therefore has the same 95 models as when the experiment was run, apart from the three that were replaced. The Hub also reports slightly lower parameter counts than before for two models of the excluded size categories, which changes neither the limits of the categories nor the selection.

Chapter 6

Results and Discussion

This chapter presents and discusses the results of the experiment described in Chapter 4. It begins with an overview of the results, which establishes the range of accuracy and of energy consumption observed across the models under study, together with the energy and the time that the study itself required. Each of the four RQs is then addressed in a dedicated section, which opens with the question and a summary of its answer, followed by the evidence that supports it. The chapter concludes with a discussion of the meaning and implications of these results, and with an assessment of the threats to their validity.

The analysis comprises the **53 model runs** of the final selection, corresponding to 38 instruct runs and 15 reasoning runs from 47 models, six of which were evaluated in both operating modes. With 10 repetitions of each model run, the experiment comprised 530 runs (Section 4.3). The three Ministral 3 instruct runs, however, were replaced by runs of the same models with 16-bit weights instead of 8-bit weights, obtained in a supplementary execution with only 5 repetitions (Section 5.6). The 8-bit runs are kept only to compare the two formats. Together, the two executions amount to 545 runs and 202,740 inferences. The quantities and the statistical methods employed throughout the chapter are defined in Section 4.6. In brief, the value reported for a model run is the median of its repetitions, energy always refers to the consumption of the GPU, and the intelligence efficiency is expressed in correct answers per kJ. In all figures, colour denotes the size category of a model, the shape of the marker denotes its family, and hollow markers denote reasoning runs.

6.1 Overview of the Results

Figure 26 positions every model run according to its energy per question and its accuracy. The individual values of each model run are reported in Table A.1 and Table A.2, in the appendix.

The most immediate observation is that the model runs differ far more in the energy they consume than in the accuracy they achieve. Accuracy ranges from 18.0% to 79.3%, a factor of 4.4, whereas the energy per question ranges from 184 J to 39,243 J, a factor of **213**. The two operating modes form clearly

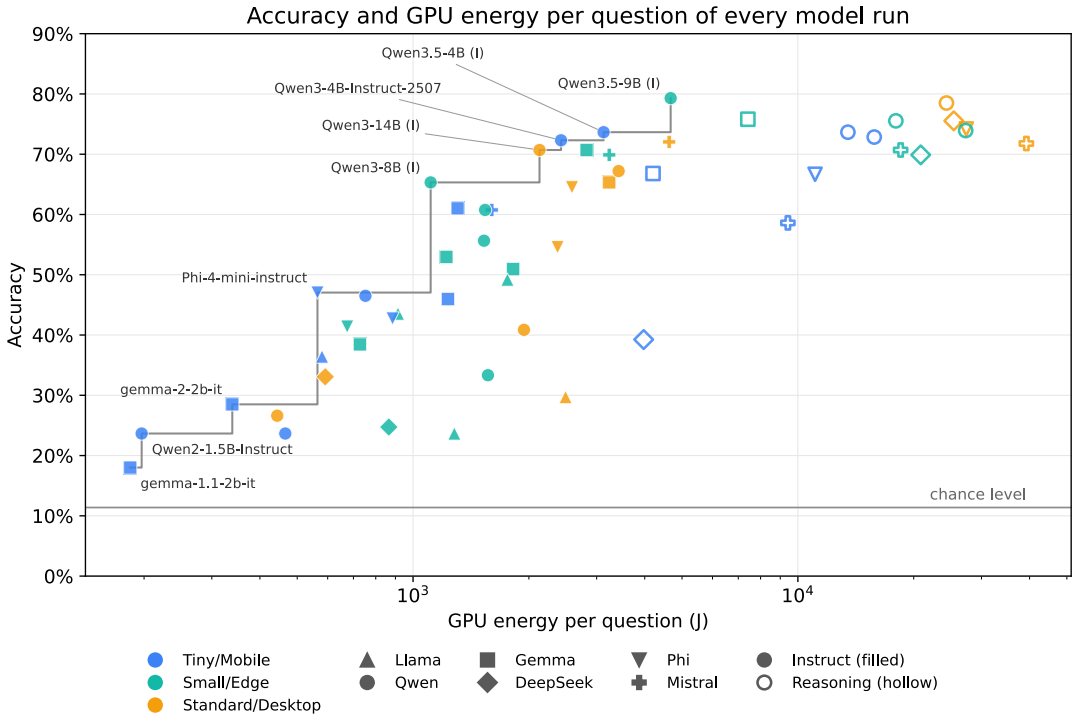


Figure 26: Accuracy and GPU energy per question of every model run, with the Pareto front.

separate groups: every reasoning run consumes more energy per question than every instruct run of the same size category, as a direct consequence of generating between 1,708 and 4,103 tokens per question, compared with 80 to 814 for the instruct runs. The size categories, by contrast, overlap almost entirely in terms of accuracy, and the most accurate model run of the study, Qwen3.5 9B in instruct mode, belongs to the Small/Edge category.

The line drawn in the figure corresponds to the *Pareto front*, which comprises the nine model runs listed in Table 24. All of them are **instruct runs**, and six were released in 2025 or 2026. This does not mean that the instruct runs are generally more accurate than the reasoning runs: the median reasoning run is more accurate than 95% of the instruct runs, partly because all of them were released from 2025 onwards, and for seven of the 15 reasoning runs the only instruct run that is more accurate is Qwen3.5 9B, released in February 2026. The effect of the operating mode itself is assessed on pairs of runs of the same release in Section 6.4.4. The front rises steeply at first and then flattens: between Gemma 1.1 2B and Qwen3 8B, a sixfold increase in the energy per question is accompanied by a gain of 47 points in accuracy, whereas the further fourfold increase up to Qwen3.5 9B yields only 14 additional points. Qwen3 8B marks the point of greatest curvature of the front, and may therefore be regarded as the most balanced compromise between energy consumption and accuracy within this set of models.

The last column of Table 24 reveals a property of the intelligence efficiency that conditions the interpretation of the remainder of the chapter: along the front, it **decreases as accuracy increases**.

Model Run	Size Category	Released	Accuracy	Energy per Question (J)	Correct Answers per kJ
Gemma 1.1 2B	Tiny/Mobile	2024-03	18.0%	184	0.988
Qwen2 1.5B	Tiny/Mobile	2024-06	23.7%	197	1.210
Gemma 2 2B	Tiny/Mobile	2024-07	28.5%	339	0.835
Phi-4-mini	Tiny/Mobile	2025-02	47.0%	564	0.833
Qwen3 8B	Small/Edge	2025-04	65.3%	1,111	0.588
Qwen3 14B	Standard/Desktop	2025-04	70.7%	2,131	0.332
Qwen3 4B Instruct-2507	Tiny/Mobile	2025-08	72.3%	2,425	0.298
Qwen3.5 4B	Tiny/Mobile	2026-02	73.7%	3,127	0.236
Qwen3.5 9B	Small/Edge	2026-02	79.3%	4,671	0.170

Table 24: Model runs on the Pareto front of accuracy and energy per question (all in instruct mode).

The highest value observed in the study, 1.21 correct answers per kJ, belongs to Qwen2 1.5B, whose accuracy of 23.7% is merely twice the chance level. Across the 53 model runs, the rank correlation of the intelligence efficiency is −0.96 with the energy per question and −0.68 with the accuracy. Because energy varies far more widely than accuracy, ranking the models by this ratio is nearly equivalent to ranking them by how little energy they consume. For this reason, the ratio is never reported in isolation in this chapter, but always alongside the accuracy and the energy from which it derives, and its limits as a measure of efficiency are discussed in Section 6.6.5.

Accuracy and energy also vary between the 14 subjects of the benchmark (Figure A.17, in the appendix). Law has the lowest accuracy in both operating modes, with medians of 26.8% and 38.2%, whereas the subjects that involve calculation (engineering, math, chemistry and physics) require 1.3 to 1.8 times the energy of an average question, against 0.4 times for psychology. With 12 to 42 questions each, however, the accuracy of a subject is far less precise than that of a whole run.

Two sources of uncertainty affect the reported values. With respect to energy, the measurements proved highly repeatable: in the 35 model runs whose repetitions produced identical responses, the energy differs by at most 1.0% between repetitions (Figure A.1, in the appendix), while the remaining 18 model runs are examined in Section 6.7. With respect to accuracy, the 372 questions constitute a sample of the benchmark, which entails a margin of error of 5 points at a confidence level of 95% (Section 4.4.2). Consequently, differences of only a few points between two model runs are not interpreted as differences between the corresponding models.

The temperature of the hardware was equally stable. From the same starting state (Section 4.4.4), the GPU reached within five minutes an average of 60 °C to 67 °C, which follows the power of the model,

and the average of a run differs from the other repetitions of its model run by 0.2 °C in the median, with no trend over the 50 days of the main execution (Figure A.2 and Figure A.3, in the appendix).

6.1.1 Energy and Time of the Study

Together, the two executions of the experiment took 1,206.0 hours, or **50.3 days**. Energy was measured during the inferences, which account for 95% of that time, and amounts to **341.8 kWh** on the GPU, and to 457.7 kWh when the processor and the memory that were measured are included. These values exclude the second processor in the main execution (Section 5.6), the cooling of the server and the time during which no energy was recorded, and therefore constitute a lower bound of the consumption of the study. The 15 reasoning runs account for 82% of the GPU energy of the main execution, and five of them for 44%.

For reference, the GPU energy of the study corresponds to the electricity that one person in the European Union consumes at home in **81 days** [219], to **1,810 km** travelled in an electric car [220], or to **18,000 charges** of a smartphone [221]. In the median model run, a single question requires 0.39 Wh in instruct mode and 4.99 Wh in reasoning mode.

6.2 Evolution Across Successive Releases (RQ1)

> **RQ1.** How has the energy efficiency of open LLMs evolved across successive releases?

Answer to RQ1. The released models did not become more intelligence-efficient, although a fixed energy budget came to yield a higher accuracy. The models became substantially more accurate, at a rate of approximately 20 percentage points per year, but their gains in accuracy were accompanied by a higher energy per question, driven by longer responses, while the energy per token remained unchanged for models of the same size. Their intelligence efficiency therefore did not improve in any size category. The energy required to attain a given accuracy, by contrast, decreased by a factor of 1.4 to 3.4 among the instruct runs (Section 6.3), but the newer models used this progress to attain a higher accuracy rather than to consume less.

This section examines the models as they were released, comparing each of them with the earlier models of the same size category. The complementary perspective, namely what a given amount of energy makes it possible to obtain, evolved differently and is addressed in Section 6.3. Table 25 summarizes the yearly

Yearly Change	**Tiny/Mobile** (13 runs)	**Small/Edge** (15 runs)	**Standard/Desktop** (10 runs)
Accuracy (points)	+23.9 ($p < 0.001$)	+17.9 ($p < 0.001$)	+22.9 ($p = 0.001$)
Energy per question	×2.84 ($p < 0.001$)	×1.57 ($p = 0.020$)	×1.67 ($p = 0.098$)
Tokens per question	×2.56 ($p < 0.001$)	×1.73 ($p = 0.004$)	×1.30 ($p = 0.174$)
Energy per token	×1.11 ($p = 0.223$)	×0.91 ($p = 0.940$)	×1.28 ($p = 0.676$)
Correct answers per kJ	×0.62 ($p = 0.004$)	×0.93 ($p = 0.771$)	×0.96 ($p = 0.960$)

Table 25: Yearly change of each metric among the instruct runs of each size category, with the p-value of its association with the release date.

change of each metric among the instruct runs of each size category, estimated according to the procedure described in Section 4.6, and the following subsections examine these trends in turn.

6.2.1 Accuracy and Energy per Question

Figure 27 depicts the accuracy of every model run as a function of its release date, with lines connecting the successive instruct releases of each family.

Accuracy increased markedly in all three size categories, at a rate of **18 to 24 points per year**, which constitutes the clearest trend identified in this study. The median accuracy of the instruct runs rose from 24.7% for the models released in 2023 to 41.8% in 2024, 65.3% in 2025 and 72.2% in 2026. To illustrate the magnitude of this progression, Gemma 1.1 2B, released in March 2024, attains 18.0%, only marginally above the chance level, whereas Qwen3.5 4B, a model of the same size category released two years later, attains 73.7%.

The energy per question, shown in Figure 28, followed a less favourable course, as it **did not decrease in any size category**.

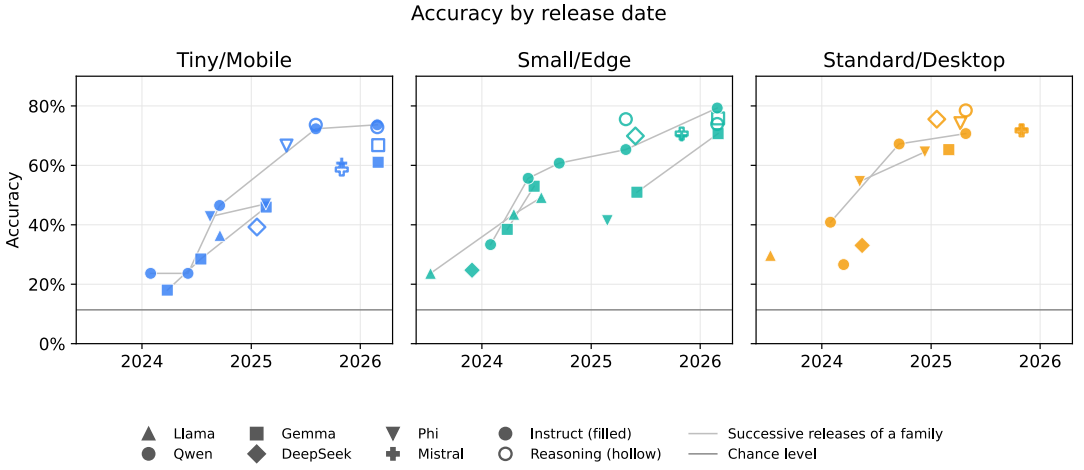


Figure 27: Accuracy of the model runs by release date.

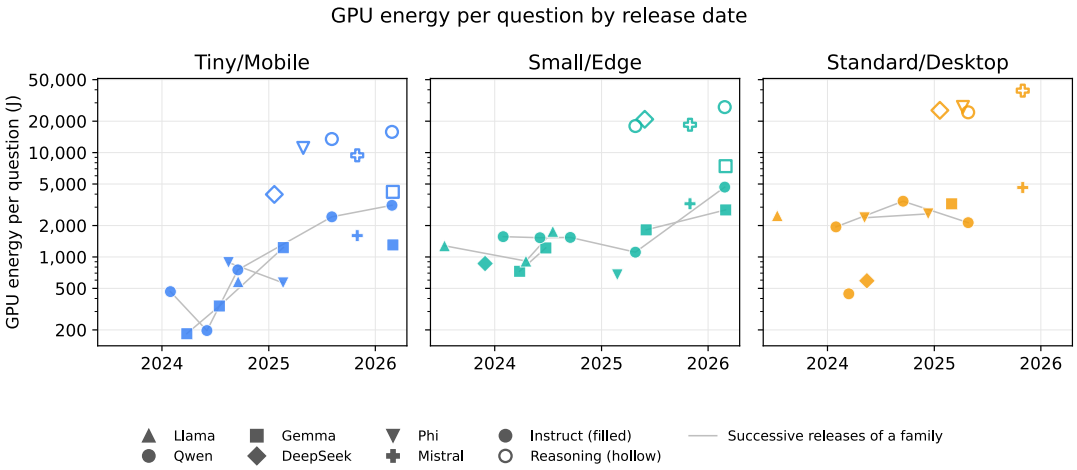


Figure 28: GPU energy per question of the model runs by release date.

Among the instruct runs, the increase was most pronounced in the Tiny/Mobile category, where the energy per question grew by a factor of 2.8 per year: the models released in 2024 consume between 184 J and 885 J per question, whereas the two released in 2026 consume 1,306 J and 3,127 J. In the Small/Edge category, the growth amounts to a factor of 1.6 per year and is concentrated at the end of the period, with the three most recent runs consuming between 2,824 J and 4,671 J, compared with 674 J to 1,820 J until mid-2025. In the Standard/Desktop category, the estimated change is not statistically significant and is, moreover, influenced by the two sparsely activated models released in 2024. When these are excluded, the dense instruct runs remain between 1,942 J and 4,630 J throughout the period, with a yearly change of 1.3 times that is likewise not significant ($p = 0.233$). The reasoning runs, which only appear from 2025 onwards, consume between 3,975 J and 39,243 J per question. The distribution of the energy across the individual questions of each model run is provided in the appendix (Figure A.4, Figure A.5 and Figure A.6).

6.2.2 Intelligence Efficiency

Since accuracy and energy per question increased simultaneously, the evolution of the intelligence efficiency depends on which of the two grew faster. As Figure 29 shows, the intelligence efficiency of the released models **did not improve**.

In the Small/Edge and Standard/Desktop categories, the estimated changes of −7% and −4% per year show no association with the release date. In the Tiny/Mobile category, the estimated change is −38% per year, the only statistically significant one observed for this metric ($\rho = -0.74, p = 0.004$). This decrease depends, however, on the credit that the ratio gives to answers obtained by chance, since the Tiny/Mobile models released in the first half of 2024 attain accuracies of 18.0% to 23.7%, not far above the chance

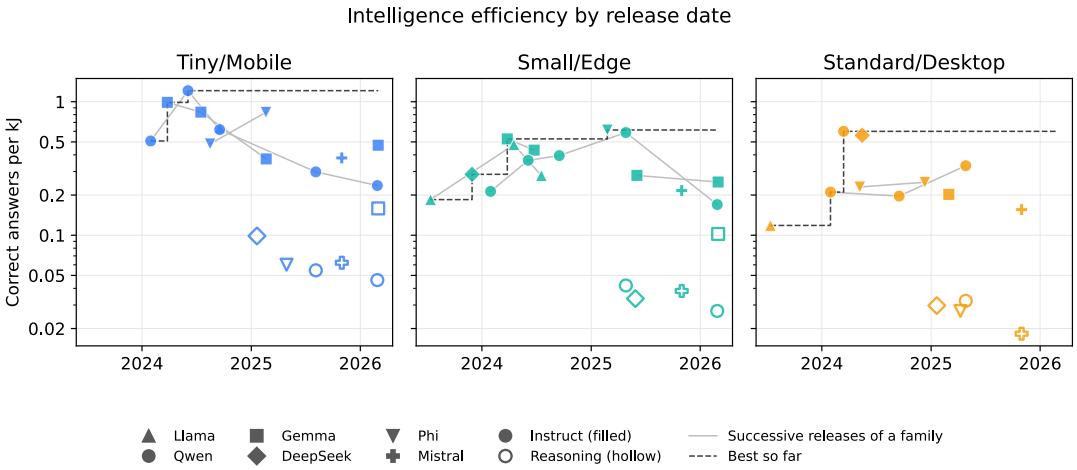


Figure 29: Intelligence efficiency of the model runs by release date.

level of 11.4%, with very little energy. When the accuracy is counted above the chance level (Section 4.6), the estimated change in this category becomes −18% per year and is no longer significant ($\rho = -0.31$, $p = 0.306$), while those of the other two categories become +8% and +13% per year, likewise without significance (Table A.11, in the appendix). The conclusion that holds under both definitions is therefore that the intelligence efficiency of the released models did not improve, and not that it declined. The dashed lines of the figure, which track the best value among the models released up to each date, corroborate this conclusion: in every category, the highest intelligence efficiency belongs to a model of modest accuracy released in 2024 or early 2025, namely Qwen2 1.5B, Phi-4-multimodal and Qwen1.5-MoE-A2.7B, with accuracies of 23.7%, 41.4% and 26.6%, respectively.

This result is robust to the exclusion of the sparsely activated models. Among the dense instruct runs alone, the yearly change is −44% in the Tiny/Mobile category ($p = 0.007$), −2% in Small/Edge and +18% in Standard/Desktop, the latter two being non-significant.

6.2.3 The Three Factors of the Intelligence Efficiency

To understand why the gains in accuracy did not translate into gains in efficiency, the intelligence efficiency can be decomposed into three factors: the accuracy, the tokens per question and the energy per token (Section 4.6). The first of these improved, as shown above, and the behaviour of the remaining two explains why the ratio did not:

- **Tokens per question.** The responses of the models became considerably longer. Among the instruct runs, the tokens per question grew by a factor of 2.6 per year in the Tiny/Mobile category and of 1.7 in the Small/Edge category (Table 25, and Figure A.7 in the appendix), with the median instruct run rising from 194 tokens per question for the models of 2024 to 339 for those of 2025 and

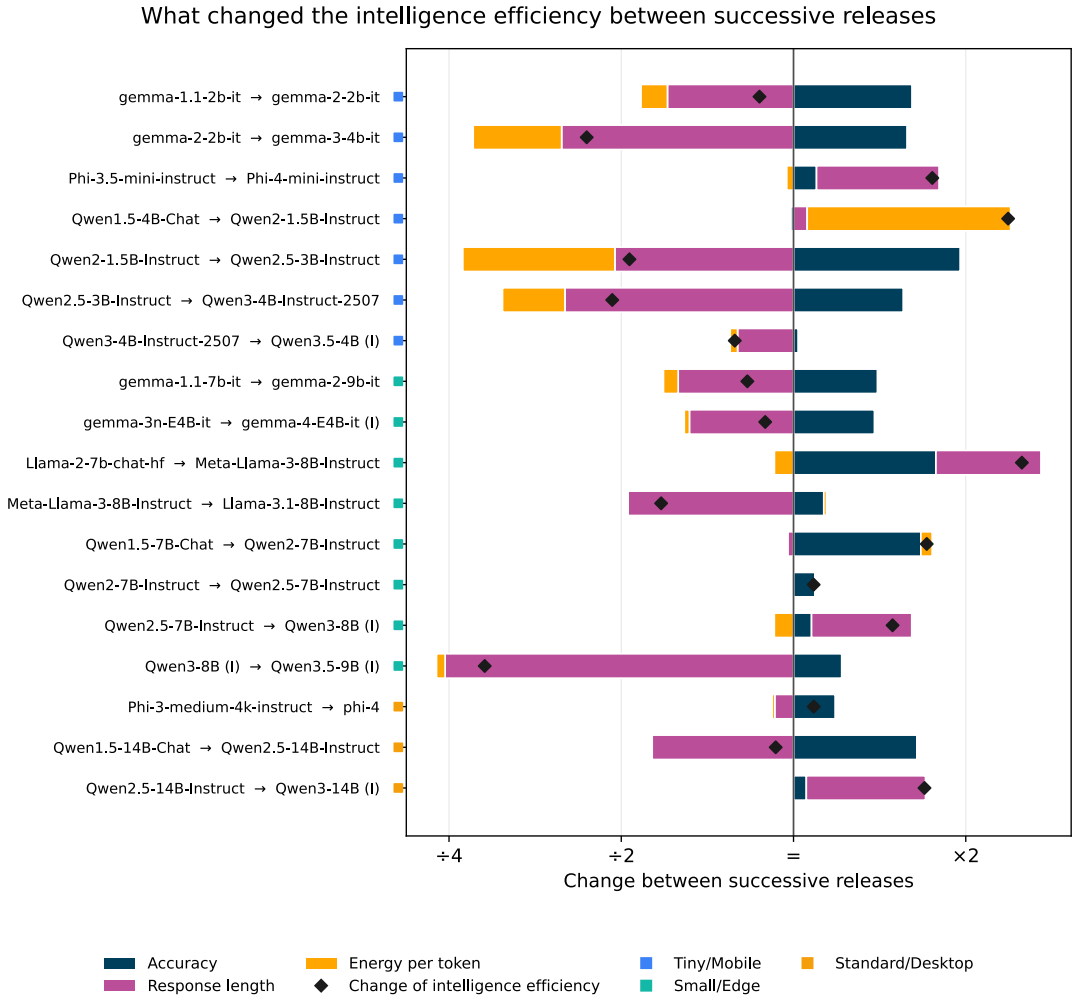


Figure 30: Contribution of each factor to the change of intelligence efficiency between successive releases of the same family.

657 for those of 2026. Part of this growth is concentrated in a small number of very long responses: in the six instruct runs released since August 2025 in which more than 5% of the responses reached the output limit of 2,048 tokens, these responses hold 18.8% to 44.3% of the GPU energy of the run (Table A.8, in the appendix). The typical response nevertheless grew as well, as the median response of the median instruct run went from 106 tokens for the models of 2024 to 152 for those of 2025 and 444 for those of 2026, and the yearly growth of this median remains significant in the two smaller categories. Since every instruct run received identical prompts, containing the same five examples (Section 4.4.3), this growth reflects a change in how the models respond, and not in what they were asked.

- **Energy per token.** The energy required to generate each token remained unchanged. None of its yearly changes is statistically significant and, once the size of the model is taken into account (Section 6.4.1), the residual variation shows no association with the release date ($\rho = -0.17, p =$

0.224, Figure A.8 in the appendix). In other words, a token generated by a recent model requires approximately the same energy as a token generated by an older model of the same size.

Figure 30 illustrates how the three factors combine between each instruct release and the preceding release of the same family, size category and architecture. Each bar partitions the change in intelligence efficiency into the contributions of the three factors, and the diamond indicates the net change.

Across the 18 pairs of successive releases, accuracy improved in 17 and the responses became longer in 13. In the most recent releases, the latter effect outweighs the former: Qwen3.5 9B is 1.2 times as accurate as Qwen3 8B, yet its responses are four times longer. The energy per token, in turn, changed by 10% or less in 15 of the pairs once the size is taken into account, and the larger changes coincide with a new model of a different size. Consequently, the intelligence efficiency improved in **only 8 of the 18 pairs**. The same decomposition, performed between the release years of each size category, yields the same pattern (Figure A.9, in the appendix).

6.3 The Energy Cost of Accuracy (RQ2)

> **RQ2.** What is the energy cost of the accuracy gained by newer models?

Answer to RQ2. Higher accuracy did not necessarily require more energy, but in most cases it was obtained with more. Of the 17 releases that are more accurate than their predecessor, 12 also consume more energy per question, at 2 J to 523 J for each additional point, whereas 5 are more accurate with less energy. Among the models available at each moment, the best accuracy attainable within a fixed energy budget increased, and a given accuracy became progressively cheaper: between early 2024 and April 2025, the energy required to attain 40%, 50% or 60% decreased by a factor of 1.4 to 3.4. The cost of accuracy nonetheless rises steeply with its level, each point costing approximately thirteen times more above 65% than below it, and, among the models measured, no further progress occurred after April 2025 for budgets of 2 kJ per question or less.

This section addresses the question in three steps. It first measures what each release required, in energy, for the accuracy it gained over its predecessor. Since accuracy and energy changed simultaneously from one release to the next, it then separates the two by holding one of them constant (Section 4.6), which corresponds to the perspective of a practitioner who has a fixed amount of energy available per question, or a fixed level of accuracy to attain, and who is therefore not constrained to a particular family or size,

since any model released up to that moment may be selected. Lastly, it assesses how the cost of accuracy varies with its level.

6.3.1 The Cost Between Successive Releases

The most direct measure of the cost of the accuracy gained by a new model is the comparison with the model it succeeds. Among the 18 pairs of successive releases considered in Section 6.2, the successor is more accurate in 17, and in **12 of these it also consumes more energy per question** (Table A.4, in the appendix). In the median pair, the energy per question increases by 42%. In the 12 pairs in which accuracy was obtained with more energy, each additional point costs between 2 J and 523 J per question, with a median of 51 J, and the highest costs belong to the most recent releases: Qwen2.5 7B gained 5.1 points over Qwen2 7B at 2 J per point, whereas Qwen3.5 9B gained 14.0 points over Qwen3 8B at 255 J per point.

In the remaining five pairs, the successor is more accurate while consuming less energy per question, by 2% to 38%: Llama 3 8B, Qwen2 7B, Phi-4-mini, Qwen3 8B and Qwen3 14B. Higher accuracy therefore did not necessarily require more energy. All five were, however, released until April 2025, and in the four pairs completed since then the successor consumes between 29% and 320% more energy per question than its predecessor.

6.3.2 The Accuracy Available at a Given Energy

Figure 31 presents the Pareto front of the model runs released until the end of each year. Since the front of a given year retains the model runs of previous years that were not surpassed, the fronts coincide wherever no improvement occurred.

The front **advanced in every year** of the period. Considering only the models released until the end of 2023, no amount of energy yielded an accuracy above 29.7%. One year later, the front reached 67.2% and exceeded the previous one at every level of energy. In 2025, it reached 78.5%, although its upper end consisted of reasoning runs consuming between 13 kJ and 24 kJ per question, which were superseded in 2026 by the instruct runs of Qwen3.5: Qwen3.5 9B attains 79.3% with 4,671 J per question. In these last two years, however, the advance was not uniform across the levels of energy, as the front only changed above approximately 0.5 kJ per question in 2025, and above approximately 3 kJ in 2026.

Figure 32 depicts the same evolution month by month and for every energy budget, while Table 26 reports its values at the end of each year for four budgets and four accuracy targets, together with the model run that holds each of them.

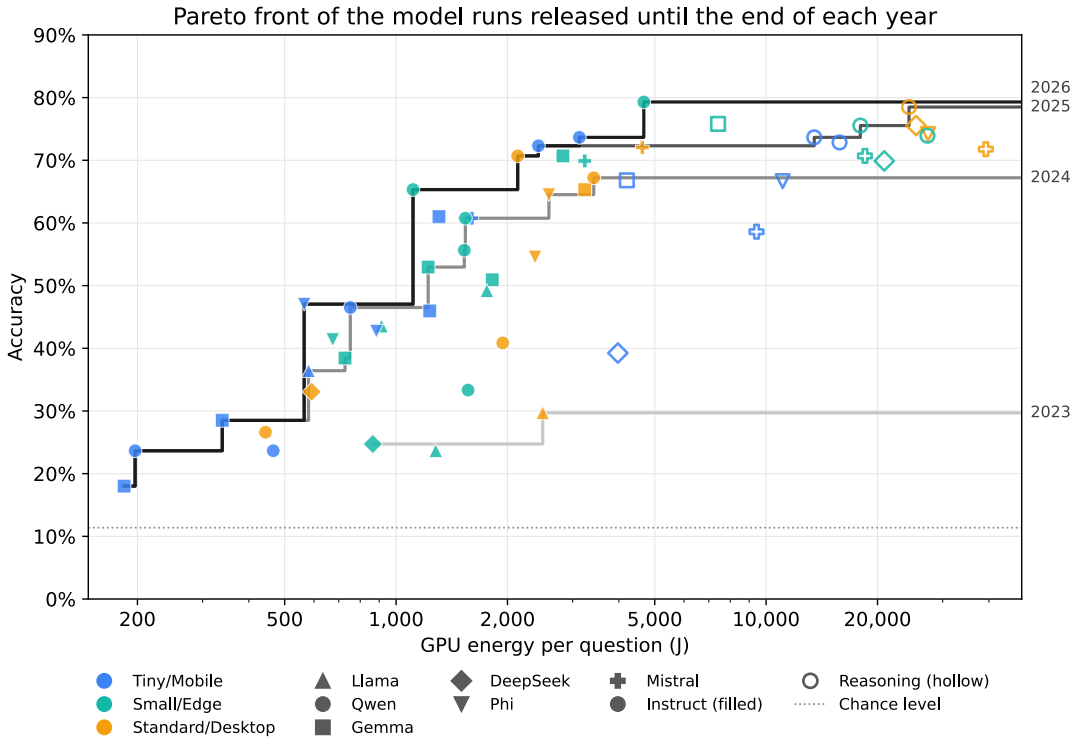


Figure 31: Pareto front of the model runs released until the end of each year.

Three observations can be drawn from these results:

- **At a fixed budget, accuracy increased.** Within 2 kJ per question, the best accuracy rose from 24.7% at the end of 2023 to 60.8% at the end of 2024 and 65.3% in 2025, and within 5 kJ, from 29.7% to 67.2%, 72.3% and, in early 2026, 79.3%. It should be noted that a budget constitutes an

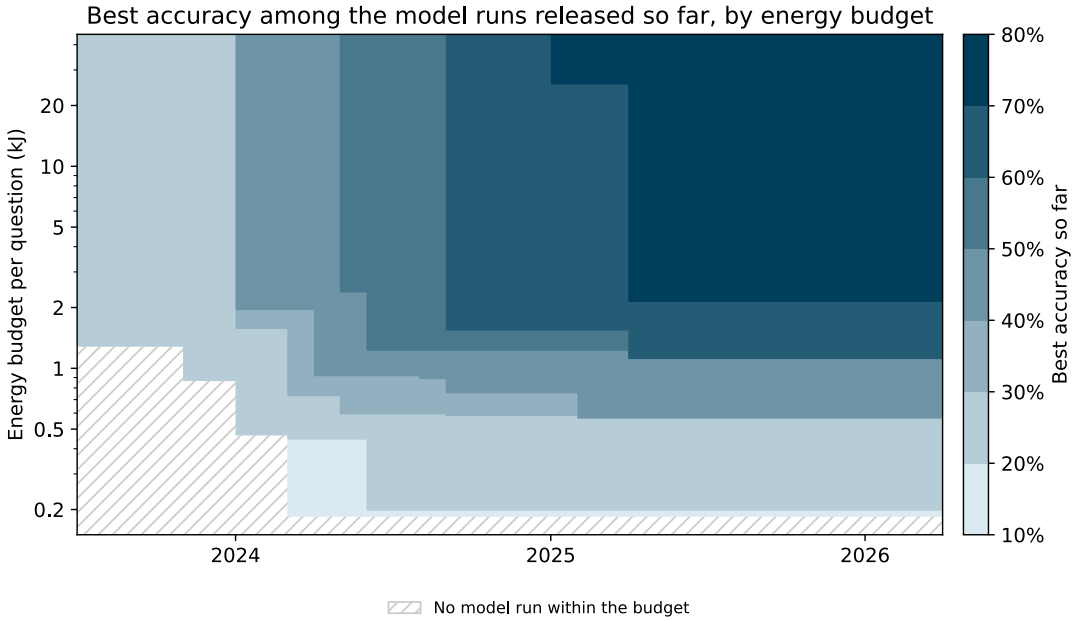


Figure 32: Best accuracy among the model runs released so far, for every energy budget per question.

Level	2023	2024	2025	2026
Best accuracy within an energy budget per question				
0.5 kJ	none	28.5% Gemma 2 2B	28.5% Gemma 2 2B	28.5% Gemma 2 2B
1 kJ	24.7% DeepSeek-LLM 7B	46.5% Qwen2.5 3B	47.0% Phi-4-mini	47.0% Phi-4-mini
2 kJ	24.7% DeepSeek-LLM 7B	60.8% Qwen2.5 7B	65.3% Qwen3 8B	65.3% Qwen3 8B
5 kJ	29.7% Llama 2 13B	67.2% Qwen2.5 14B	72.3% Qwen3 4B Instruct-2507	79.3% Qwen3.5 9B
Lowest energy per question for an accuracy target				
40%	none	752 J Qwen2.5 3B	564 J Phi-4-mini	564 J Phi-4-mini
50%	none	1,221 J Gemma 2 9B	1,111 J Qwen3 8B	1,111 J Qwen3 8B
60%	none	1,539 J Qwen2.5 7B	1,111 J Qwen3 8B	1,111 J Qwen3 8B
70%	none	none	2,131 J Qwen3 14B	2,131 J Qwen3 14B

Table 26: Best model run released until the end of each year, at each energy budget and accuracy target (the last column covers the releases until March 2026).

upper limit, and not the energy actually consumed: within 5 kJ, the model run holding the best accuracy consumed 2,491 J per question at the end of 2023, 3,422 J in 2024, 2,425 J in 2025 and 4,671 J in 2026. These gains were therefore achieved without raising the budget, although not always without consuming more energy.

- **Among the models measured, the lowest budgets ceased to improve.** Within 0.5 kJ per question, the best accuracy has remained at 28.5% since July 2024. Within 1 kJ, it merely rose from 46.5% to 47.0% after 2024, and within 2 kJ no improvement has occurred since April 2025. This stagnation follows from the fact that every model run released since June 2025 consumes more than 1.3 kJ per question. It describes the models that were executed, since only the largest model of each release in each size category was selected (Section 6.7).

- **For a fixed accuracy, the required energy decreased.** An accuracy of 40% was first attained in January 2024 with 1,942 J per question, by Qwen1.5 14B, and thirteen months later with 564 J, by Phi-4-mini, a reduction of 3.4 times achieved by a model with fewer than 4 billion parameters in place of one with 14 billion. Similarly, the energy required for an accuracy of 50% decreased by a factor of 2.1 between May 2024 and April 2025, and that required for 60% by a factor of 1.4 between September 2024 and April 2025. An accuracy of 70% was first attained in January 2025 by a reasoning model, with 25,419 J per question, and three months later by Qwen3 14B in instruct

mode, with 2,131 J. This reduction of 12 times is of a different nature from the other three, as it results from the replacement of a reasoning run by an instruct run. Since April 2025, however, none of these four values has decreased further.

The evolution of these budgets and targets over time, including within each size category, is detailed in the appendix (Figure A.10, Figure A.11 and Table A.5).

These findings are consistent with those of Section 6.2, despite appearing to contradict them. When one model run lies above another at the same energy, its intelligence efficiency is necessarily higher, which is precisely what the front reveals from one year to the next: at approximately 1.2 kJ per question, the most accurate option at the end of 2024 was Gemma 2 9B, with an accuracy of 53.0% and 0.43 correct answers per kJ, whereas one year later it was Qwen3 8B, with 65.3% and 0.59. The models that were actually released, however, shifted towards higher energies, with the median instruct run consuming 899 J per question in 2024, 1,820 J in 2025 and 2,976 J in 2026. Moreover, the intelligence efficiency declines along the front itself, from 1.21 correct answers per kJ at its lower end to 0.17 at its upper end. In short, the best model available **at a given energy** improved from one year to the next, but the models released over time **did not remain at the same energy**, and moved towards higher energies.

6.3.3 The Marginal Cost of Accuracy

Across the whole period, the more accurate model runs of the study are those that consume more energy: among the 38 instruct runs, the rank correlation between the accuracy and the energy per question is 0.79. Accuracy is, moreover, more closely associated with the number of generated tokens (0.70) than with the size of the model (0.46 with the active parameters), although this reflects in part the growth of both accuracy and response length over time (Section 6.2): among the instruct runs released before mid-2025, accuracy is more closely associated with size (0.65) than with tokens (0.47), and since then the two are similar (0.67 and 0.68).

How much energy each level of accuracy requires can be assessed among the best model runs, those of the Pareto front. Figure 33 considers its nine model runs: the left panel shows the energy per correct answer of every model run as a function of its accuracy, and the right panel shows the additional energy per question required for each additional point of accuracy between consecutive runs of the front.

The cost of accuracy rises steeply along the front. The 47 points that separate its first model run from Qwen3 8B, which attains 65.3%, cost 927 J per question, or approximately 20 J per point, whereas the 14 points that separate Qwen3 8B from Qwen3.5 9B cost 3,560 J, or approximately 255 J per point, **thirteen times more**. The individual steps of the front exhibit the same contrast, each point costing

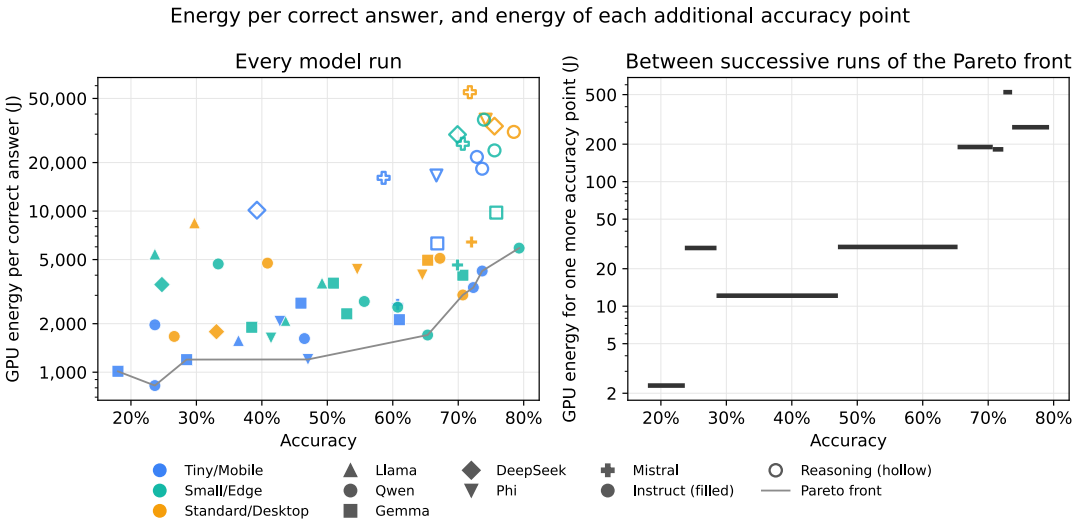


Figure 33: Energy per correct answer of every model run (left) and energy of each additional point of accuracy along the Pareto front (right).

between 2 J and 30 J up to the accuracy of Qwen3 8B, and between 182 J and 523 J above it. Two of the latter steps, however, span fewer than 2 points of accuracy, a difference smaller than the margin of error of the sample of questions, so that their individual values are merely indicative. The energy per correct answer exhibits the same pattern: along the front, it ranges from approximately 1 kJ for the least accurate model runs to 1,701 J for Qwen3 8B and 5.9 kJ for Qwen3.5 9B, and it reaches 54,676 J for Ministral 3 14B Reasoning, the model run with the highest energy consumption.

6.4 The Influence of Design Choices (RQ3)

> **RQ3.** How do model design choices, such as size, architecture and operating mode, influence energy efficiency?

Answer to RQ3. The size of a model determines the energy of each token, whereas its operating mode determines the number of tokens it generates. The energy per token follows the number of active parameters (98% of its variation), so that smaller models attain a higher intelligence efficiency, although size is a weak indicator of accuracy. A reasoning run consumes 2.6 to 19.6 times the energy per question of the instruct run of the same model and, within the output limit of this experiment, is more accurate in 8 of the 12 pairs. Sparse activation delivered the energy of a small model, but not the accuracy of a large one. In the three models observed with both precisions, 16-bit weights required 1.6 to 1.7 times the energy per token of 8-bit weights, with no significant difference in accuracy.

This section examines in turn the three design choices named in the question, namely the size of a model, its architecture and its operating mode, together with a fourth characteristic that the experiment made it possible to observe in three models, the numerical precision of the weights. It concludes by comparing the share of the energy consumption that the size and the operating mode account for, and by examining the execution time.

6.4.1 Model Size

The energy per token of a model is determined almost entirely by its size, as Figure 34 shows. A power law on the number of active parameters, that is, the parameters involved in the generation of each token, accounts for **98% of its variation** among the 47 models:

$$\text{energy per token} \approx 1.03\ \text{J} \times (\text{active parameters, in billions})^{0.91}$$

According to this relation, doubling the number of active parameters multiplies the energy per token by 1.9. The relation also generalizes to models excluded from the fit: when the energy per token of each model is predicted from the remaining 46, the median error is 4.4% (Figure A.12, in the appendix). When the total number of parameters is used instead, the same fit accounts for only 67% of the variation, because the sparsely activated models consume considerably less energy per token than dense models of the same total size. It should be emphasized that this relation characterizes the GPU, the inference engine and the configuration of this experiment, and that other setups would yield different values.

Size is, by contrast, a much weaker indicator of accuracy. Figure 35 tracks the best accuracy among the instruct runs released up to each date in each size category.

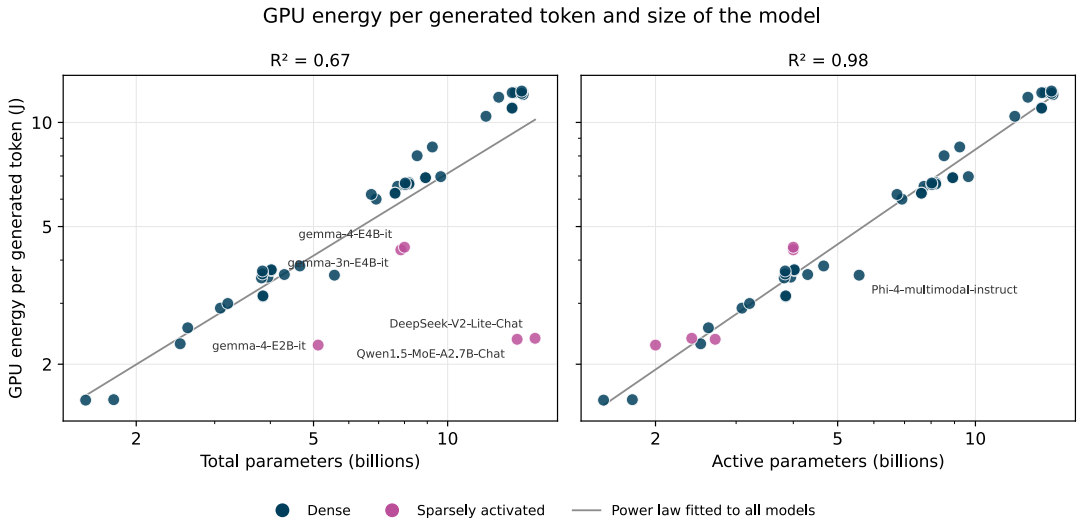


Figure 34: GPU energy per token of the models against their total parameters (left) and their active parameters (right).

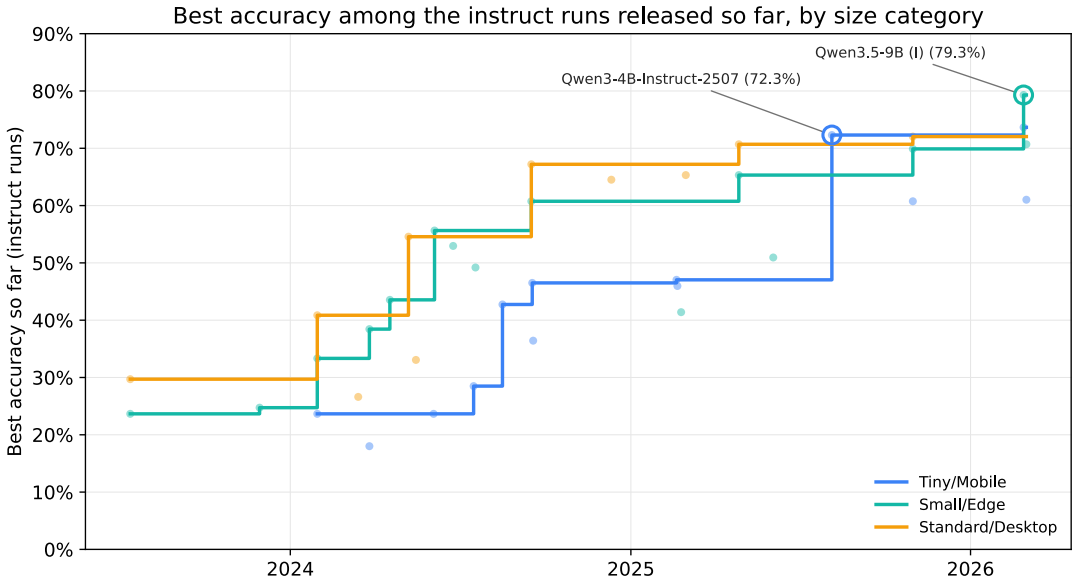


Figure 35: Best accuracy among the instruct runs released so far in each size category.

Until early 2025, the Small/Edge and Standard/Desktop categories remained close to each other, and each new best accuracy of the Tiny/Mobile category had been attained 3 to 12 months earlier by the larger categories. This ordering ceased to hold thereafter. In August 2025, Qwen3 4B Instruct-2507 attained an accuracy of 72.3%, matching the best Standard/Desktop instruct runs of the study (70.7% and 72.0%), and in February 2026 Qwen3.5 9B, a Small/Edge model, attained 79.3%, the highest accuracy of the study. This comparison is subject to one limitation: the final selection contains only three Standard/Desktop instruct runs released after 2024, and none from 2026, so that the most recent small models are compared with older large ones.

A smaller model of equivalent accuracy does not, however, necessarily consume less energy. Qwen3 4B Instruct-2507 requires less than a third of the energy per token of Qwen3 14B (3.73 J against 12.24 J), as its size predicts, yet it generates 3.7 times more tokens per question and ultimately consumes more energy per question (2,425 J against 2,131 J). Size therefore determines the energy of each token, but not the number of tokens that are generated.

With respect to the intelligence efficiency, the smaller models rank higher. Among the instruct runs, its median is 0.51 correct answers per kJ in the Tiny/Mobile category, 0.29 in Small/Edge and 0.22 in Standard/Desktop, and its rank correlation with the number of active parameters is −0.72. This advantage derives from the energy of each token and not from the accuracy, whose median is lower in the smaller categories (46.0%, 50.9% and 59.5%, respectively). It has, moreover, narrowed among the releases of 2025 and 2026, to 0.38, 0.27 and 0.20 correct answers per kJ, because in these releases the smaller models are those that generate the longer responses (542, 447 and 310 tokens per question in the median instruct run of each category).

6.4.2 Architecture

Five of the 47 models are sparsely activated. Two of them are MoE models of the Standard/Desktop category, Qwen1.5-MoE-A2.7B and DeepSeek-V2-Lite, which activate 2.7 and 2.4 billion of their 14.3 and 15.7 billion parameters, respectively. The remaining three are Gemma 3n E4B, Gemma 4 E2B and Gemma 4 E4B, which activate 2 or 4 billion of their 5.1 to 8.0 billion parameters. These are the effective sizes indicated by their names, which the analysis uses for all three models, whereas Google reports 2.3 and 4.5 billion effective parameters for Gemma 4 E2B and E4B [136], a difference too small to alter the relation between size and energy described in Section 6.4.1.

The purpose of this design is to endow a model with the parameters of a large model while computing each token as a small one (Section 2.5.2). Were this fully reflected in the results, a sparsely activated model would exhibit the accuracy of dense models of its total size together with the energy consumption of dense models of its active size. To assess both expectations, Table 27 compares the instruct run of each sparsely activated model with the dense instruct runs released up to six months before or after it, divided into two groups: those of the same size category, which have a similar total number of parameters, and those whose size lies within 1.5 times its active parameters. Accuracy is compared through the difference to the median accuracy of each group, and energy through the ratio to its median.

The results confirm the second expectation, but not the first. With respect to energy, the two MoE models consume approximately a fifth of the energy per token of the dense models of their size category, and the Gemma models approximately two thirds, while both remain close to the dense models of their

Model (total / active parameters, B)	**Dense Models Compared**	**Number**	**Accuracy** (points)	**Energy per Token** (ratio)	**Energy per Question** (ratio)
Qwen1.5-MoE-A2.7B	Same size category	2	−21.1	0.19	0.21
(14.3 / 2.7)	Similar active size	4	+0.5	0.77	1.10
DeepSeek-V2-Lite	Same size category	3	−21.5	0.19	0.25
(15.7 / 2.4)	Similar active size	4	+0.6	0.87	1.29
Gemma 3n E4B	Same size category	3	−14.4	0.63	1.64
(7.9 / 4.0)	Similar active size	5	+3.9	1.18	1.48
Gemma 4 E2B	Same size category	2	−6.2	0.65	0.55
(5.1 / 2.0)	Similar active size	0	none	none	none
Gemma 4 E4B	Same size category	2	−3.9	0.63	0.71
(8.0 / 4.0)	Similar active size	2	+3.5	1.25	1.19

Table 27: Instruct run of each sparsely activated model compared with the dense instruct runs released up to six months before or after it.

active size. With respect to accuracy, the two MoE models fall **21 points below** the dense models of their size category, and within 1 point of the dense models of their active size. The Gemma models, in turn, fall 4 to 14 points below the former group and stand approximately 4 points above the latter.

A comparison within the same family leads to the same conclusion. Qwen1.5-MoE-A2.7B has nearly the same number of parameters as Qwen1.5 14B, yet it is 14.2 points less accurate ($p < 0.001$), while consuming 77% less energy per question. Its accuracy does not differ significantly from that of Qwen1.5 4B (3.0 points higher, $p = 0.348$), at approximately the same energy per question, and lies 6.7 points below that of Qwen1.5 7B ($p = 0.009$), the model with which its developers compare it [222]. DeepSeek-V2-Lite compares more favourably with the dense model that preceded it in its family, being 8.3 points more accurate than DeepSeek-LLM 7B ($p = 0.004$) with 32% less energy per question. It nonetheless remains 21 points below the dense models of its size category and period. Furthermore, none of the five sparsely activated models lies on the Pareto front.

In this experiment, sparse activation therefore delivered **the energy of a small model, but not the accuracy of a large one**, while still requiring the memory of a large model (Section 3.1). As this result rests on five models, of which only two are MoE models, both released in the first half of 2024, it should not be generalized to the architecture as a whole.

6.4.3 Numerical Precision

The numerical precision of the weights is not a characteristic in which the models of this study differ, since all of them are compared with 16-bit weights. Its effect could nonetheless be observed, as the three Ministral 3 instruct models were executed with both 8-bit and 16-bit weights (Section 5.6), which provides an opportunity to measure it on otherwise identical models. With 16-bit weights, each token requires **1.58, 1.70 and 1.74 times** the energy required with 8-bit weights, for the models with 3, 8 and 14 billion parameters, respectively. The accuracy of the two versions differs by at most 2.2 points, and none of the differences is statistically significant (p between 0.229 and 0.890, Table A.6 in the appendix). Although three models of a single family do not allow this result to be generalized, it indicates that the precision of the weights can alter the energy per token considerably without a measurable loss of accuracy on this benchmark, and it confirms that the precision must be identical across all the models of a comparison.

6.4.4 Operating Mode

The final selection contains **12 pairs** of an instruct run and a reasoning run of the same release and size: six dual-mode models, in which both runs use the same weights, and six pairs of separate variants. Two

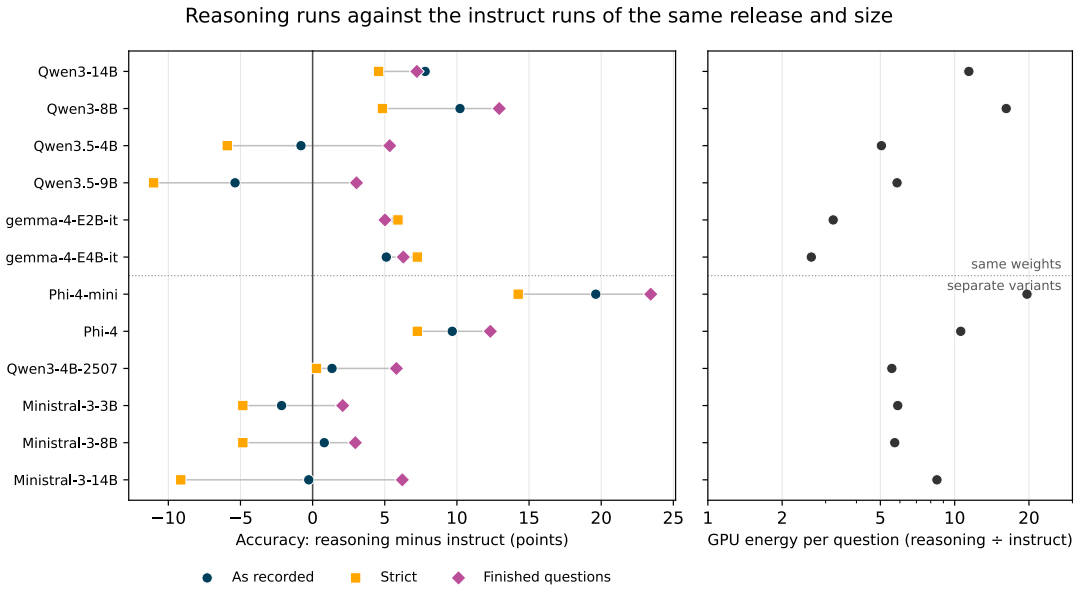


Figure 36: Difference of accuracy between the reasoning run and the instruct run of each pair, in three views (left), and ratio of their GPU energy per question (right).

conditions of the experiment must be borne in mind when interpreting this comparison. First, the instruct runs are not devoid of reasoning, since their prompts include five examples answered step by step and instruct the model to proceed likewise (Figure 23). Second, the reasoning runs are subject to an output limit of 6,144 tokens, three times that of the instruct runs, which a portion of their responses reached. Figure 36 compares the two runs of each pair, and the corresponding values are listed in Table A.7, in the appendix.

In terms of energy, the result is consistent across all pairs. The reasoning run consumes between **2.6 and 19.6 times** the energy per question of the instruct run, and 5.9 times in the median pair. Since the energy per token is identical in the two modes, with a ratio between 0.98 and 1.02, this difference is entirely attributable to the number of generated tokens. As a consequence, the intelligence efficiency of the reasoning run amounts to between 7% and 41% of that of the instruct run (Figure A.13, in the appendix).

In terms of accuracy, the result depends on how the responses that reached the output limit are treated, which is why the figure presents three views (Section 4.6). *As recorded*, the reasoning run is more accurate in **8 of the 12 pairs**, by 3.2 points in the median pair, with differences ranging from −5.4 points for Qwen3.5 9B to +19.6 points for Phi-4-mini. Seven of these differences are statistically significant, six in favour of the reasoning run and one in favour of the instruct run. Under the *strict* view, in which every response that reached the limit is counted as incorrect, the reasoning run is more accurate in 7 pairs. On the *finished questions*, it is more accurate in all 12, by 6.0 points in the median pair.

The discrepancy between the views is explained by the truncated responses. In 13 of the 15 reasoning runs, between 8.3% and 29.6% of the responses reached the output limit, most of them while the model

was still reasoning (Table A.8, in the appendix), and the four pairs in which the reasoning run is less accurate as recorded are precisely those with the highest share of responses at the limit. Only the two Gemma 4 models completed nearly all of their responses. Within the output limit of this experiment, the reasoning mode therefore did not always improve accuracy, and the cases in which it did not are associated with the truncated responses. The finished questions nevertheless constitute the easier part of the benchmark: on the questions that no reasoning run left unfinished, the instruct runs achieve a mean accuracy of 64.4%, against 34.8% on the remaining ones. This view thus reveals what the reasoning mode achieves when it is able to conclude, rather than what these runs would achieve under the higher limit recommended by their developers (Section 6.7).

With the accuracy as recorded and with the strict accuracy, no reasoning run lies on the Pareto front of the study, since for each of them an instruct run attains at least the same accuracy with less energy, but this only holds since February 2026. Until then, the upper end of the front as recorded consisted of three reasoning runs, Qwen3-4B-Thinking-2507 and Qwen3 8B and 14B in reasoning mode (Section 6.3.2), displaced by the instruct runs of Qwen3.5 4B, with the same accuracy and less than a quarter of the energy, and of Qwen3.5 9B, which attains 79.3% with 4,671 J per question, against 78.5% with 24,329 J for Qwen3 14B, the most accurate reasoning run.

6.4.5 Relative Influence of the Design Choices

The energy per question of a model run is the product of its energy per token and its tokens per question. Table 28 reports the share of the variation of each of these quantities, across the 53 model runs, that is explained by the size of the model, by its operating mode, and by both combined (Section 4.6).

Variation Explained	Size	Operating Mode	Size and Operating Mode
Energy per token	97.7%	0.1%	97.7%
Tokens per question	0.2%	82.6%	82.6%
Energy per question	22.2%	67.7%	86.5%

Table 28: Share of the variation of the energy per token, of the tokens per question and of the energy per question explained by the size and by the operating mode of the model runs.

The two factors of the energy per question thus depend on distinct design choices: the energy per token depends on the size of the model and not on its operating mode, whereas the tokens per question depend on the operating mode and not on the size. Of the two, the **operating mode** accounts for the larger share of the differences in energy per question observed among the model runs of the study. The variation

that remains unexplained originates mostly in the tokens per question, which also differ between models operating in the same mode.

6.4.6 Execution Time

The same two design choices govern the time required to answer a question, which follows its energy almost exactly. Since the average power of the GPU remained between 236 W and 307 W in every model run, the rank correlation between the time and the energy per question is 0.999, so that the conclusions drawn in this chapter for the energy per question apply equally to the execution time. As with the energy, the size of a model determines the speed of generation, with a median of 95 tokens per second in the Tiny/Mobile category, 46 in Small/Edge and 26 in Standard/Desktop, whereas the operating mode determines the number of tokens: a question takes 5.0 s in the median instruct run and 60.0 s in the median reasoning run, twelve times longer (Table A.1 and Table A.2).

6.5 Trajectories of the Model Families (RQ4)

> **RQ4.** Do open model families follow a common trajectory, or do they diverge in their approach to efficiency?

Answer to RQ4. In their results, the families follow a common trajectory: their successive releases are more accurate, generate longer responses and consume more energy per question, with an energy per token that depends on the size of the model. Where the trajectories differ, the difference follows the period of the releases more closely than the family, and no significant difference between families was found at equal release date and size category. In their approach, by contrast, the families diverge: Gemma adopted sparse activation and the least costly reasoning mode, Phi shortened its responses, and Qwen lengthened them the most. This evidence is nonetheless limited, as only four of the six families can be followed over time, and only two of them until 2026.

The six families are not equally represented in the study (Table A.9, in the appendix). A trajectory can only be traced where a family has instruct runs from more than one release in the same size category, a condition that is met in **eight cases**: three for Qwen, two for Gemma, two for Phi and one for Llama. DeepSeek and Mistral have no such case, and can therefore only be compared with the other families

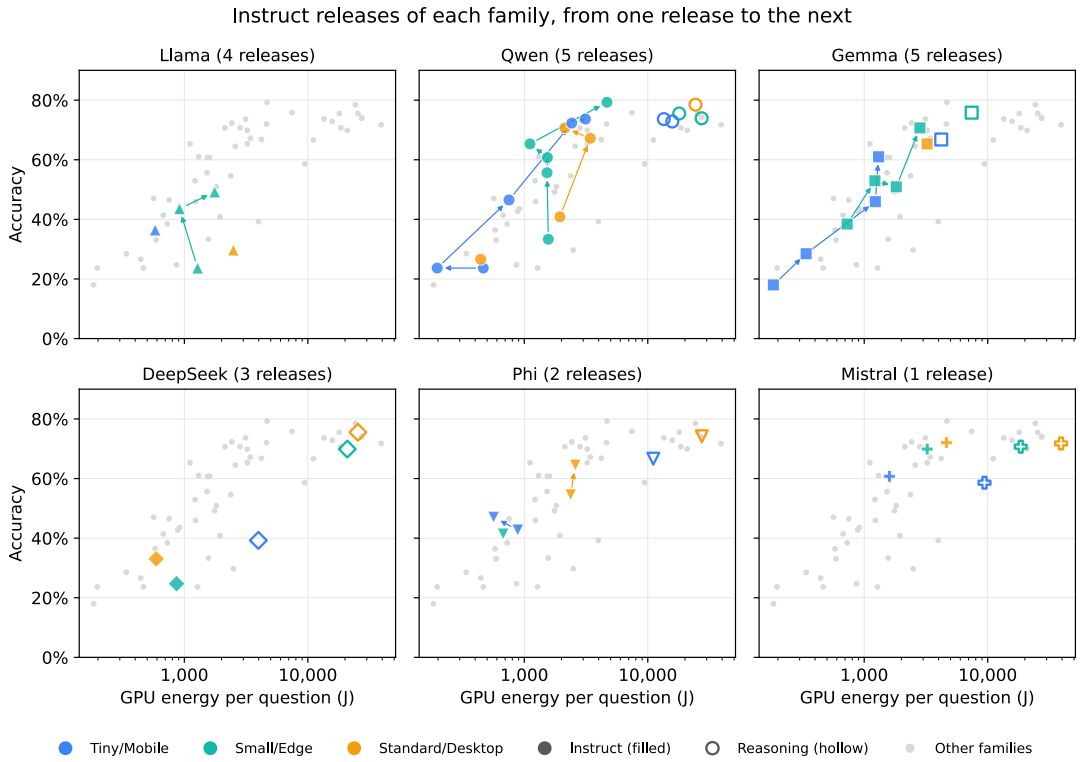


Figure 37: Instruct runs of each family by accuracy and GPU energy per question, with an arrow from each release to the next.

at a single moment. Figure 37 displays the instruct runs of each family, with an arrow connecting each release to the next within each size category.

The trajectories share a common direction. In all eight, the last release is more accurate than the first, by 4.3 to 50.0 points, and in seven it also consumes more energy per question, by a factor of 1.1 to 7.1. The sole exception is the Tiny/Mobile trajectory of Phi, in which Phi-4-mini consumes 36% less energy per question than Phi-3.5-mini.

To identify the origin of these changes, Figure 38 partitions the change in intelligence efficiency between the first and the last release of each trajectory into the contributions of the three factors, whose values are listed in Table A.10, in the appendix.

The three factors behave similarly in nearly every trajectory. The responses became longer in seven of the eight, most notably in the four that extend to 2026, where they grew from 130 to 814 tokens per question in the Tiny/Mobile models of Qwen and from 80 to 574 in those of Gemma. The energy per token remained stable in seven, the exception being the Small/Edge trajectory of Gemma, in which dense models were succeeded by sparsely activated ones. As a result, the intelligence efficiency increased in four trajectories, by 8% to 75%, and decreased in the other four, by 20% to 53%.

These two groups are distinguished by the period they cover more clearly than by the family to which they belong. The four trajectories in which the intelligence efficiency decreased are the only ones that

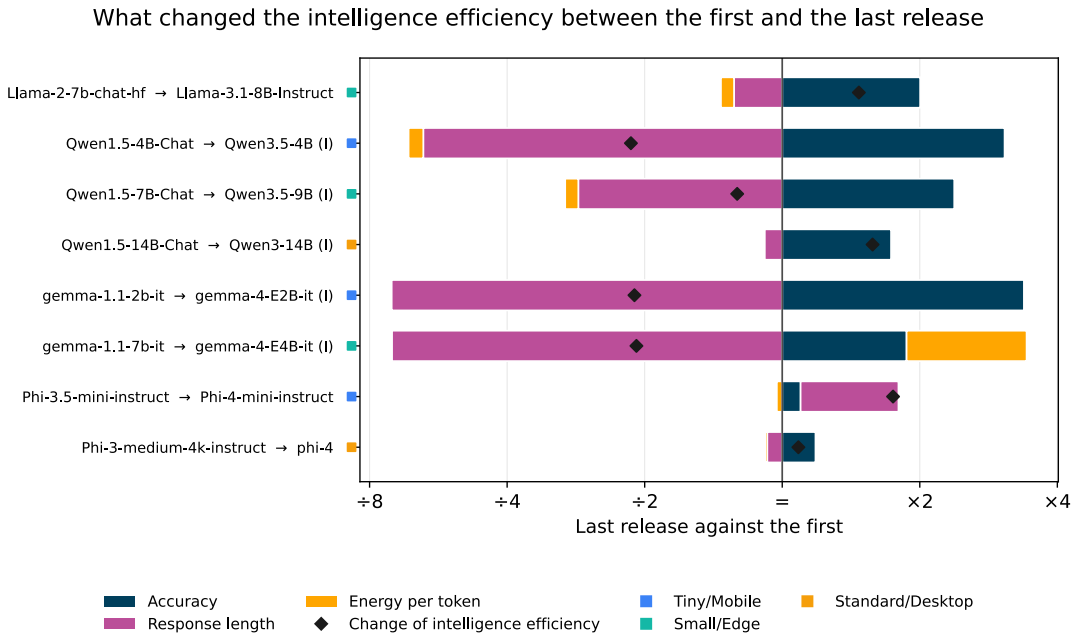


Figure 38: Contribution of each factor to the change of intelligence efficiency between the first and the last release of each trajectory.

include releases from 2026, and Qwen is present in both groups: its intelligence efficiency increased in the Standard/Desktop trajectory, which ends in April 2025, and decreased in the other two, which extend to February 2026. This pattern is consistent with the growth of the response length being a characteristic of the most recent releases, rather than a choice of particular families. The two explanations cannot, however, be separated with the available data, given that Qwen and Gemma are the only families with releases in 2026.

A comparison of the families at the same moment supports the same conclusion. A linear model of each metric, with the release date, the size category and the family as predictors (Section 4.6), reveals **no significant effect of the family** on the accuracy ($p = 0.170$), the tokens per question ($p = 0.316$), the energy per question ($p = 0.400$) or the intelligence efficiency ($p = 0.434$, Figure A.16 in the appendix). This does not demonstrate that the families are equivalent: with 2 to 14 instruct runs per family, a difference of several points of accuracy between two of them could remain undetected.

What distinguishes the families is therefore not the direction of their trajectories, but the means they adopted, which Table 29 summarizes on the basis of the design choices examined in Section 6.4.

The convergence of the results thus coexists with a **diversity of means**. Two of these choices lowered one of the factors of the energy per question, namely the shorter responses of Phi and the sparse activation of Gemma, and each was adopted by a single family. The lengthening of the responses, which raised it, is common to the two families whose releases extend to 2026.

Family	Reasoning Mode	Energy of Reasoning (ratio)	Distinguishing Choice
Llama	None in the selection	none	Responses of 137 to 266 tokens per question in every release, the last of which dates from 2024
Qwen	Same weights, and one pair of separate variants	5.0 to 16.2	Largest growth of the response length, from 130 to 814 tokens per question in Tiny/Mobile
Gemma	Same weights, in Gemma 4	2.6 to 3.2	Sparse activation in the smaller models, with 45% less energy per token in Small/Edge
DeepSeek	Separate models, with no instruct counterpart	none	An MoE model in 2024, and reasoning models only in 2025
Phi	Separate variants	10.6 to 19.6	Shorter responses in Phi-4-mini than in its predecessor, with 38% fewer tokens per question
Mistral	Separate variants	5.7 to 8.5	A single release, at the end of 2025

Table 29: Approach of each family in the final selection: form of the reasoning mode, ratio of its energy per question to that of the instruct run, and distinguishing choice.

6.6 Discussion

The four answers share a single explanation, which lies in the three factors of the intelligence efficiency: across successive releases, the models answered more questions correctly, generated more tokens to do so, and spent on each token the same energy as an older model of the same size. This section considers why this may have happened and what it implies. Since its interpretations extend beyond what the experiment can establish, they are presented as hypotheses, each with the evidence that supports it and the limits that qualify it.

6.6.1 The Invariance of the Energy per Token

The regularity of the energy per token suggests that **no structural improvement in efficiency occurred in these models** during the period studied, in the sense of a change that renders a model of a given size cheaper to execute. The number of active parameters accounts for 98% of its variation across six families and nearly three years of releases, and the remainder shows no association with the release date (Section 6.4.1). The experiment would have detected such an improvement, since the energy per token does change with the amount of computation, as with 8-bit weights or sparse activation, while it remains identical in the two operating modes of a model. This regularity also explains why previous studies found the number of parameters to be a poor predictor of the energy of a request [196], and the length of the response a strong one [73], [195]: the parameters determine the cost of a token, and not the number of tokens.

The progress of these models was therefore of a different nature: their parameters did not become cheaper to compute, but were used more effectively, so that a smaller model became sufficient for a given accuracy (Section 6.3). Sparse activation, the only structural change in the sample, lowered the cost of a token without bringing the accuracy of a larger model, possibly because the models concerned were derived from small dense models, as Qwen1.5-MoE-A2.7B was from Qwen-1.8B [222]. This conclusion is, however, limited to what the experiment varies. The hardware, the inference engine and the serving of one request at a time were fixed, and these are precisely the dimensions in which large gains have been reported, such as the improvement of more than 100 times in one year measured by MLPerf Power in systems running a fixed set of models [203]. The gains in efficiency reported elsewhere may thus originate largely in the systems that execute the models, rather than in the models themselves.

6.6.2 The Origin of the Accuracy Gains

The gains in accuracy appear to have **two sources, which succeeded each other**. Until mid-2025, they were not accompanied by longer responses: the median accuracy of the instruct runs rose from 24.7% for the models of 2023 to 41.8% for those of 2024 with responses of similar length, gains that can only originate in the way the models were trained. Since then, accuracy has continued to rise, but every instruct run generates between 419 and 814 tokens per question, against 80 to 339 for those released before. The difference in length between the two operating modes narrowed accordingly, from 11 to 19 times more tokens in reasoning mode for the pairs released until April 2025 to 2.6 to 8.5 times for those released since August 2025, mostly because the instruct responses grew.

The experiment cannot establish why the recent responses are longer. The prompts of the instruct runs ask for step-by-step answers (Figure 23), so that every instruct response is expected to contain some reasoning, and the growth may originate either in a more thorough compliance of the recent models with this instruction, or in a training that makes them more verbose in general. The examples included in the prompt cannot account for it, since their answers are brief, with a median of 54 words, and it is the older models that remain close to them, whereas the median instruct run of 2026 writes 259 words. Whatever its origin, the additional length coincided with the additional accuracy of the recent models. Separating the two origins would require prompts that do not request step-by-step answers (Section 6.7).

6.6.3 The Rising Cost of Accuracy

Each additional point of accuracy costs more than the previous one. Along the Pareto front, a point costs approximately thirteen times more above 65% than below it (Section 6.3), possibly because

the last points come from the hardest questions, which elicit the longest responses: on the questions that a reasoning run left unfinished, the instruct runs attain a mean accuracy of 34.8%, against 64.4% on the others (Section 6.4.4). The reasoning mode illustrates this rise most clearly: it multiplied the energy of a question by 2.6 to 19.6 for a gain of 3.2 points in the median pair, within the output limit of this experiment, so that, since February 2026, no reasoning run lies on the Pareto front. Dauner and Socher [79] likewise attributed the cost of reasoning to its additional tokens, whereas Jin et al. [118] found reasoning to be a better trade-off than a larger model. The conditions of this experiment may account for part of the disagreement, since the instruct prompts already ask for step-by-step answers and the reasoning runs are truncated, and the task for another part: extended reasoning appears necessary only for a minority of the questions of this benchmark, those of the subjects that involve calculation (Figure A.18, in the appendix). The cost of reasoning is not fixed, however. The Gemma 4 models reason at 2.6 to 3.2 times the energy of their instruct runs, and Qwen3 8B and Phi-4-mini at 16 to 20 times, which suggests that developers have room to make reasoning more economical.

6.6.4 The Reinvestment of Efficiency Gains in Accuracy

Whether these models became more efficient depends on what is held constant, and this reconciles the studies that appeared to disagree in Section 2.7. The released models consumed as much energy as their predecessors or more, as Luccioni and Gamazaychikov observed [78], while the best model available at a given energy improved every year, as Saad-Falcon et al. observed for local inference [37], and by a similar margin: the best accuracy within 2 kJ per question increased 2.6 times between 2023 and 2025, against the 3.1 times that these authors attribute to the models. The decrease of the energy required for a fixed accuracy, in turn, mirrors what has been reported for the compute used in training and for prices [199], [200], [201].

Taken together, these results describe a pattern in which **the gains in efficiency are reinvested in accuracy**. Each year, a given accuracy became available with less energy, and each year the released models used that margin to attain a higher accuracy, so that the median instruct run of 2026 consumes 2,976 J per question, against 899 J in 2024. In the models of this study, this margin was absorbed by the length of the responses. Since a token requires approximately the same energy in a new model as in an older one of the same size, a release is more intelligence-efficient than such a predecessor only if its accuracy grows faster than its responses, and in the most recent releases the responses grew faster (Section 6.2). With the hardware and the inference engine held constant, the change of efficiency from one release to the next was therefore decided by the balance between the accuracy and the number of

tokens generated, and not by the cost of computing each token. A possible reason why this number was allowed to grow is that a ranking by accuracy does not reveal it: benchmarks rank models by their accuracy alone, and the leaderboards that measure energy do not measure accuracy, which is the practice that Schwartz et al. [35] designated as Red AI (Chapter 2). This answers the question left open in Chapter 1: in the families and sizes studied, the models as a whole became capable of greater efficiency, but each new release used this progress to raise accuracy, and the tension described in Section 1.2 was resolved in favour of capability.

6.6.5 The Limits of a Single Ratio

The intelligence efficiency, that is, the relation between the accuracy of a model and the energy it consumes (Section 1.2), was expressed in this chapter by a single ratio, the number of correct answers per kJ. Taken alone, this ratio proved to be an unreliable guide to that relation, for three reasons:

- **It follows the energy rather than the accuracy.** The energy per question varies by a factor of 213 and the accuracy by a factor of 4.4, so that the ratio orders the models almost exactly by how little energy they consume (Section 6.1).
- **It credits the answers obtained by chance**, which favours small models that consume very little energy. With the accuracy counted above the chance level, the only decline observed loses its significance (Section 6.2).
- **It decreases along the Pareto front**, so that a lower ratio is the expected consequence of choosing a more accurate model, and not, in itself, evidence of inefficiency.

The ratio is therefore meaningful only between models of similar accuracy or of similar energy. Its value in this study lay less in ranking the models than in its decomposition into three factors, which revealed that only the accuracy and the length of the responses changed.

6.6.6 Implications

These results have practical implications for four groups:

- **For those who select a model.** A selection is better based on the accuracy that is required or on the energy that is available than on the ratio (Section 6.6.5), and the most recent model is not

necessarily the most suitable: for budgets of 2 kJ per question or less, the best models of this study were released in April 2025 or earlier.

- **For those who develop models.** Since the energy per token is fixed by the size of the model, the length of the responses is where efficiency was lost and where it can be recovered. Longer responses are not an unavoidable cost of accuracy, as five releases were more accurate than their predecessor while consuming less energy (Section 6.3.1).

- **For those who measure efficiency.** The ratio should be reported together with the accuracy at a fixed energy and the energy at a fixed accuracy, and three conditions that altered the results of this study should be stated: the output limit, the numerical precision of the weights, which the name of a repository does not always indicate (Section 5.6), and the random seed.

- **For environments with limited energy and hardware.** Their access to these models, a matter of equity (Section 1.2), improved until April 2025, as a given accuracy came to require less energy and a smaller model (Section 6.3.2). Since then, a given accuracy has required less memory but not less energy, as the recent small models generate the longest responses, so that the recent gains serve mainly the settings in which energy is less scarce.

6.7 Threats to Validity

The threats to the validity of this study are organized into four groups, each presented with the measures taken to address it. The verifications mentioned below are detailed in Table A.11, in the appendix.

6.7.1 Construct Validity

- **Representativeness of the benchmark.** The capability of a model is represented by its accuracy on 372 multiple-choice questions of a single benchmark, in English. Other tasks may rank the models differently and elicit responses of different lengths.

- **Exposure to the benchmark.** MMLU-Pro has been public since 2024, and 56.6% of its questions are adapted from MMLU, which has been public since 2021 [7]. Models released after these dates may have encountered the questions during training, which would inflate the gains in accuracy of the most recent releases. This effect cannot be measured here, and it affects any comparison over time conducted on a public benchmark. The energy measurements are not affected.

- **Measure of the intelligence efficiency.** The ratio that expresses it credits the answers that a model would obtain by chance, 11.4% in this sample, and follows the energy far more closely than the accuracy (Section 6.6.5). With the accuracy counted above the chance level, the decline observed in the Tiny/Mobile category is no longer significant, and no trend becomes significant in the other two categories, so that the absence of improvement is the only conclusion drawn from this metric.

- **Scope of the energy measurement.** The efficiency metrics rely on the energy of the GPU, which represents 71% to 76% of the energy that was measured (Section 4.5.3), and a smaller share of that of the whole server. The idle GPU draws 35 W (Section 4.1.3), which corresponds to 11.3% to 14.6% of the GPU energy of each model run. Subtracting this idle consumption changes the position of a model run in the ranking by intelligence efficiency by one place at most, and leaves the Pareto front and the trends unchanged.

- **Prompting of the operating modes.** Each type of model is prompted in the manner for which it was designed: instruct models with five examples and an instruction to answer step by step, and reasoning models without examples. The comparison of the two modes is therefore not a comparison between the presence and the absence of reasoning. The examples can also affect the energy by themselves: in code generation, prompts with three examples have been found to consume less energy than prompts without examples [198].

- **Prompt and response length.** The growth of the response length across releases, on which the answer to RQ1 rests, was measured with prompts that request step-by-step answers (Figure 23). It may partly reflect a closer compliance of the more recent models with this instruction, rather than a general tendency to answer at greater length, in which case it would be smaller with prompts that request the answer alone (Section 6.6). As the experiment contains no run without the instruction, the two cannot be separated.

6.7.2 Internal Validity

- **Output limit.** In 13 of the 15 reasoning runs, between 8.3% and 29.6% of the responses reached the output limit, as did 5.9% to 15.1% of the responses of six instruct runs, among which are the three most accurate model runs of the Pareto front. For Qwen3, Qwen3.5, the distilled DeepSeek-R1 models and Phi-4-reasoning, the developers recommend or use an output length of 32,768 tokens [223], [224], [225], [226], approximately five times the limit of the reasoning runs. Such a length

was not feasible in this study, for reasons of memory and of time: the context window of every model was limited to 10,240 tokens, the largest that the memory of the GPU could accommodate with the largest models of the study (Section 4.4.1), and the reasoning runs already account for 920 of the 1,125 hours of inference of the 53 model runs. The accuracy of the affected runs is therefore that of a restricted run, and their energy is a lower bound. When every response that reached the limit is counted as incorrect, the position of a model run in the ranking changes by two places at most, no reasoning run enters the Pareto front, and the trends remain unchanged.

- **Random seed.** The inference engine uses the same random seed in every run (Section 4.4.1), so that in 35 of the 53 model runs every repetition produces the same responses, and the accuracy of a model that samples its responses corresponds to a single realization. In the remaining 18, the accuracy differs between repetitions by 3.4 points in the median model run, and by 6.7 at most. A possible explanation is the number of warm-up prompts, which varied between repetitions and may alter the random numbers used for the questions that follow. This explanation is consistent with 12 of these 18 model runs but not with the other 6, so the cause was not fully identified, and the engine does not guarantee identical results from one run to another [208].

- **Supplementary execution.** The three 16-bit Ministral 3 instruct runs were executed after the others, with five repetitions instead of ten and without a control run (Section 5.6). Their energy per token is equal to that of the reasoning variants of the same models, which were measured in the main execution (Table A.1 and Table A.2), so the conditions can be assumed to be the same.

- **State of the hardware.** The temperature of the hardware was controlled before every run (Section 4.4.4), and the model runs that repeat the same responses differ by at most 1.0% in energy between repetitions.

6.7.3 External Validity

- **Single configuration.** The results were obtained on one GPU model, with one inference engine and one request at a time (Section 4.1.1 and Section 4.4.1). Other hardware, other engines or the batching of requests would yield different values of energy, and possibly different relations between the models.

- **Unquantized models.** All the models were executed with 16-bit weights, whereas models are frequently quantized for local execution (Section 2.5). This was a condition of the comparison and

not a preference, since no quantization procedure could be applied identically to every model of the study (Section 3.1 and Section 4.2). The comparison of Section 6.4.3 indicates that the energy would be lower with fewer bits. Whether the relations between the models would be preserved was not tested.

- **Selection of the models.** A single model was selected for each release, size category, architecture and operating mode, namely the one with the most parameters (Section 4.2). Two consequences follow. First, the smaller models of the most recent releases were not executed, in particular Gemma 3 270M and 1B, Qwen3 0.6B and 1.7B, and Qwen3.5 0.8B and 2B, so that the absence of progress at the lowest energy budgets describes the models measured, and not necessarily those available. Second, the instruct models selected in the Tiny/Mobile category became larger over time, from 1.5 to 4.0 billion parameters until 2024 to 3.8 to 5.1 billion since 2025 ($\rho = 0.75$, $p = 0.003$). This contributes to the growth of the energy per question in that category, although its energy per token exhibits no significant trend (Table 25).

- **Coverage of the models.** The study covers six families, models with up to 16 billion parameters and releases until March 2026. Larger models, including the larger MoE models, could not be executed (Section 4.2).

6.7.4 Conclusion Validity

- **Size of the groups.** The trends rest on 10 to 15 instruct runs per size category, the trajectories on two to five releases, and the comparison of the families on 2 instruct runs for DeepSeek to 14 for Qwen. The absence of a significant difference is therefore not evidence that no difference exists.

- **Sample of questions.** The accuracy of a model run has a margin of error of 5 points (Section 4.4.2). Differences of a few points were not interpreted, and two model runs were compared question by question whenever their accuracies were compared directly.

- **Number of tests.** A significance test at the level of 0.05 indicates an effect where none exists in 5% of the cases, and this chapter reports many such tests, so that a few of the results described as significant may be due to chance. No correction was applied for this. The conclusions therefore rest on the results whose p-values lie far below 0.05, such as the growth of the accuracy and of the response length, whereas a result with a p-value closer to that level, such as the growth of the energy per question in the Small/Edge category ($p = 0.020$), is treated as weak evidence.

Chapter 7

Conclusions and Future Work

This dissertation set out to determine whether the energy efficiency of open LLMs improves from one release to the next, or whether it is being compromised in favour of raw performance. New models are released at a rapid pace, yet none of the studies reviewed in Section 2.7 measures both the energy and the accuracy of successive releases of the same model families under fixed conditions, so that it remained unclear whether each new model is more efficient than the one it replaces. To answer this question, 47 open models from 20 releases of six families, published between July 2023 and March 2026, were evaluated on a single GPU, with a single inference engine and on the same 372 questions of MMLU-Pro, under conditions designed to control the factors identified in Chapter 3. The two executions of the experiment comprise 545 runs and 202,740 inferences, and required 50 days of computation and 342 kWh of GPU energy (Section 6.1.1). This chapter draws together what these measurements reveal, assesses the contributions and the limits of the study, and outlines the directions it opens.

7.1 Answers to the Research Questions

The first two questions require two perspectives, which are kept distinct in what follows: the models *as they were released*, each compared with the earlier models of its size category, and the best accuracy attainable *within a fixed energy budget*, which considers every model released up to a given moment.

- **RQ1. Evolution across successive releases.** As they were released, the models became substantially more accurate, by approximately 20 percentage points per year, but not more intelligence-efficient. Their gains in accuracy were accompanied by a higher energy per question, because their responses grew longer while each token required as much energy as in an older model of the same size. The accuracy attainable within a fixed energy budget, by contrast, did increase, as the answer to RQ2 details (Section 6.2).

- **RQ2. Energy cost of accuracy.** Higher accuracy did not necessarily require more energy, as five releases became more accurate while consuming less, but most releases obtained it with more. Within a fixed energy budget, the best attainable accuracy increased, and between early 2024 and April 2025 the energy required to attain a given accuracy decreased by a factor of 1.4 to 3.4. Each additional point, however, costs considerably more at high accuracy than at low accuracy, and, among the models measured, the lowest budgets have not improved since April 2025 (Section 6.3).

- **RQ3. Influence of design choices.** The size of a model determines the energy of each token, which follows the number of its active parameters (98% of the variation), whereas its operating mode determines how many tokens it generates. Reasoning multiplied the energy of a question by 2.6 to 19.6 for a modest gain in accuracy within the output limit of this experiment, so that, since February 2026, no reasoning run lies on the Pareto front. Sparse activation gave the energy of a small model, but not the accuracy of a large one (Section 6.4).

- **RQ4. Trajectories of the families.** The families follow a common trajectory in their results: their successive releases are more accurate, generate longer responses and consume more energy per question, and where they differ, the difference follows the period of the releases more closely than the family. In their approach, by contrast, they diverge: Gemma adopted sparse activation and the least costly reasoning mode, Phi shortened its responses and Qwen lengthened them the most. This evidence rests on the four families that can be followed over time (Section 6.5).

7.2 The Trajectory of Open Models

Taken together, these answers resolve the question left open in Chapter 1, of whether energy efficiency is improving with each consecutive release or being compromised in favour of raw performance. Neither alternative holds on its own. Taken as a whole, the models became capable of greater efficiency, since a given accuracy came to require less energy. The models released over time, however, used this progress to attain a higher accuracy rather than to consume less, while their responses grew longer. In the period studied, **the gains in efficiency were reinvested in accuracy**.

This confirms the concern raised in Section 1.2, that the advances which demand more tokens could offset the gains in efficiency, and in the most recent releases the longer responses more than offset them. The gains that were offset, however, did not come from cheaper tokens. The improvements of hardware and inference engines were excluded by design, and no release made a token cheaper than its number of active parameters predicts. Sparse activation, the only structural change in the sample, lowered the

energy of a token by reducing the number of parameters that each token activates, and these models consume approximately as much energy per token as dense models of their active size. The progress lay instead in the use that the models make of their parameters, so that a smaller model became sufficient for a given accuracy (Section 6.6). The tension described in Section 1.2 was thus resolved in favour of capability, in the manner that Schwartz et al. [35] designated as Red AI.

The evolutionary trajectory that this dissertation set out to characterize comprises two phases. Until mid-2025, accuracy rose with responses of similar length, gains that can be attributed only to the way the models were trained. Since then, accuracy has continued to rise while the responses grew longer, whether because the recent models follow the instruction to answer step by step more thoroughly or because their training makes them more verbose, two origins that this experiment cannot separate. The landscape follows this path collectively, but the divergent means of the families show that the growth of the energy per question is not inevitable: Phi shortened its responses, and Gemma adopted the least costly reasoning mode. For access to these models in environments with scarce energy and restricted hardware, a matter of equity (Section 1.2), the outcome is mixed: until April 2025, a given accuracy came to require a smaller model and less energy, but since then only less memory, as the small models now generate the longest responses, so that the recent gains serve mainly the settings in which energy is less scarce.

7.3 Contributions

The contributions announced in Chapter 1 can now be stated in the light of the results:

- **Evidence on the evolution of efficiency.** This study measures, under fixed conditions, both the energy and the accuracy of successive releases of open model families, which none of the studies in Section 2.7 had done. Its results show that the energy per token follows the number of active parameters across six families and three years of releases, and that the intelligence efficiency of the released models did not improve. By separating the models as released from the accuracy attainable within a fixed energy budget, they also reconcile studies that appeared to disagree: Luccioni and Gamazaychikov [78] observed that newer models consume as much energy as their predecessors or more, and Saad-Falcon et al. [37] that the best local model improves every year (Section 6.6).

- **A method for comparing successive releases.** Chapter 3 identifies the factors that such a comparison has to control, and Chapter 4 describes how each was handled. The decomposition of the intelligence efficiency into accuracy, tokens per question and energy per token traced every change to its origin, and the analysis showed that the number of correct answers per kJ is meaningful

only between models of similar accuracy or energy. Studies of efficiency should therefore report it with the accuracy at a fixed energy and the energy at a fixed accuracy (Section 6.6.5).

- **Open artefacts.** The dataset, with the records of every inference and the complete responses of the models, and the pipeline that produced it, from the selection of the models to the analysis, make the results verifiable and allow the experiment to be extended to new releases (Section 5.5.2).

7.4 Recommendations for Greener Models

Beyond the implications discussed for each group of stakeholders in Section 6.6, the results suggest four recommendations for the development of more energy-efficient LLMs:

- **Treat the length of the responses as a design target.** Since the energy of a token is fixed by the size of the model, the length of the responses is where efficiency was lost and where it can be recovered. Training models to answer concisely, and reporting the number of tokens they generate per answer, would make this cost visible, and the five releases that became more accurate with less energy show that longer responses are not an unavoidable cost of accuracy (Section 6.3.1).
- **Make reasoning proportional to the question.** Reasoning multiplied the energy of a question for a modest gain in accuracy, and appears necessary mainly for the questions that involve calculation (Figure A.18, in the appendix). Models that reason only when a question requires it, or within a limited budget, could retain much of its benefit at a lower cost, as the Gemma 4 models, whose reasoning mode is far less costly than that of the other families, suggest is possible.
- **Develop and select small models.** Smaller models attain a higher intelligence efficiency (Section 6.4.1), and recent small models attain accuracies that previously required much larger ones. Continuing to develop them, and selecting the smallest model that attains the required accuracy rather than the most recent one, is a direct means of reducing the energy that a task requires.
- **Compare models within an energy budget.** New models are presented and compared by their accuracy on benchmarks, while the leaderboards that measure energy do not measure accuracy. Reporting, at the release of a model, the accuracy it attains within given energy budgets would distinguish the models that are more accurate at equal energy from those that consume more to become more accurate, and would give developers an incentive to retain part of their gains in efficiency instead of reinvesting all of them in accuracy.

7.5 Limitations

These conclusions are bounded by the conditions under which they were obtained (Section 6.7). The models were executed on a single GPU model, with one inference engine serving one request at a time, and evaluated on a single benchmark of multiple-choice questions in English. They have at most 16 billion parameters, the energy considered is that of the GPU alone, and the output limit truncated part of the reasoning responses, whose energy is therefore a lower bound. The growth of the responses, on which the main conclusions rest, was moreover measured with prompts that request step-by-step answers, and may partly reflect a closer compliance of the recent models with this instruction. These conditions delimit the scope of the conclusions rather than invalidate them. They were the same for every model, which is what allows the differences between releases to be attributed to the models, and the verifications described in Section 6.7, such as counting the truncated responses as incorrect or subtracting the idle consumption of the GPU, leave the trends unchanged.

7.6 Future Work

Four directions for future research follow from these limitations and the hypotheses of the discussion:

- **Relaxing the conditions that shape the main conclusions.** Repeating the measurements with prompts that request the answer alone would establish whether the longer responses of the recent instruct models originate in their training or in their compliance with the step-by-step instruction (Section 6.6). Executing the reasoning runs with the output length that their developers recommend, approximately five times the limit used here, on a GPU with more memory, would establish the full cost and benefit of the reasoning mode. Varying the random seed between repetitions and measuring the energy of the whole system would, in turn, complete the measurements.

- **Extending the range of models and hardware.** Larger models, including the larger MoE models, would show whether the energy per token continues to follow the number of active parameters at scale, and whether sparse activation then brings the accuracy of a large model, while the smallest models of the latest releases, which the selection rule excluded, would show whether the lowest budgets truly stagnated. Repeating the experiment on hardware used for local execution, such as Apple silicon, NPUs and edge devices, with local runtimes such as llama.cpp [126] and Ollama [128] and the quantized models they frequently execute (Section 2.5), would test whether the relations between the models hold beyond the server-class GPU of this study. Closed models,

such as GPT-5, Claude or Gemini, could be compared in the same way if their providers disclosed the energy of a request under documented conditions.

- **Extending the conditions of use.** The experiment served one request at a time, presented each question in a context of its own and represented capability by a single benchmark. Batched serving, tasks with long contexts, which Section 1.2 identifies as a source of computational cost, and other tasks such as code generation (Section 2.6.7), evaluated on benchmarks published after the models so that their questions could not have been seen in training, would test whether the conclusions extend to these settings. Varying the systems that execute the models, as well as the models themselves, would also separate the contribution of each to efficiency, and test whether the gains reported elsewhere originate mainly in the systems (Section 6.6) [203].

- **A continuous, efficiency-aware benchmark.** The pipeline can evaluate new releases without changes to its logic (Section 5.5.2). Executed periodically, it would turn this study into a continuous record of the efficiency of open models, showing whether the stagnation of the lowest budgets persists and whether the responses continue to lengthen. Published with the accuracy at a fixed energy and the energy at a fixed accuracy, such a record would bring together what accuracy leaderboards and energy leaderboards, such as the AI Energy Score [78], currently measure separately, and provide the incentive that, according to this study, the development of open models has lacked.

7.7 Final Remarks

Between July 2023 and March 2026, open LLMs advanced at a remarkable pace. On the same questions and the same hardware, the best accuracy available rose from below 30% to nearly 80%, and accuracies that had required a model with 14 billion parameters came within reach of one with fewer than 4 billion. This progress was real, but it did not lie where efficiency is usually sought: the energy of each token remained fixed by the size of the model, and what improved was the use that the models make of their parameters.

The central finding of this dissertation concerns what became of that progress. Each advance created a margin, the same accuracy for less energy, and most new releases spent that margin on further accuracy, accompanied by longer responses. The intelligence efficiency of the released models did not improve, therefore, not for lack of progress, but because the progress was reinvested. This changes the question that motivated this work. Whether models become greener does not follow from technical advances alone, since the same progress that made a given accuracy cheaper was used to attain a higher one at a higher

energy per question. It depends on a choice: whether part of each gain is retained, or all of it is spent. The divergent paths of the families show that this choice is open, as Phi shortened its responses and Gemma adopted the least costly reasoning mode.

This choice matters beyond any single model. The length of the responses, which no ranking by accuracy reveals, is multiplied by every question that every user asks, at the scale at which these models are now used. It also determines who can use them: a margin that is retained is what allows a capable model to run where energy is scarce and hardware is limited, whereas the recent gains have mainly served the settings in which neither is. Greener models are therefore a matter of how many tokens they generate, and not only of how each token is computed.

The transition from Red AI to Green AI that motivated this work does not, then, depend only on technical breakthroughs. The gains in efficiency that it requires were achieved in the period studied, and were spent. What it requires is that part of them be retained, and the pipeline developed in this study makes it possible to verify, release after release, whether they are.

Bibliography

[1] A. Chatterji *et al.*, "How People Use ChatGPT," Working Paper 34255, Sep. 2025. doi: 10.3386/w34255.

[2] G. A. García-Mireles, M. Á. Moraga, F. García, and C. Calero, "Sustainability in the Field of Software Engineering: A Tertiary Study," *ACM Trans. Softw. Eng. Methodol.*, Jul. 2025, doi: 10.1145/3747178.

[3] R. Verdecchia, J. Sallou, and L. Cruz, "A systematic review of Green AI," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 13, no. 4, 2023, doi: 10.1002/widm.1507.

[4] W. X. Zhao *et al.*, "A Survey of Large Language Models," *arXiv preprint*, 2025, doi: 10.48550/arXiv.2303.18223.

[5] H. Naveed *et al.*, "A Comprehensive Overview of Large Language Models," *ACM Transactions on Intelligent Systems and Technology*, vol. 16, no. 5, 2025, doi: 10.1145/3744746.

[6] L. Phan *et al.*, "Humanity's Last Exam," *arXiv preprint*, Sep. 2025, doi: 10.48550/arXiv.2501.14249.

[7] Y. Wang *et al.*, "MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark," *arXiv preprint*, Nov. 2024, doi: 10.48550/arXiv.2406.01574.

[8] J. Austin *et al.*, "Program Synthesis with Large Language Models," *arXiv preprint*, 2021, doi: 10.48550/arXiv.2108.07732.

[9] Z. Zheng *et al.*, "LiveCodeBench Pro: How Do Olympiad Medalists Judge LLMs in Competitive Programming?," *arXiv preprint*, Jun. 2025, doi: 10.48550/arXiv.2506.11928.

[10] C. E. Jimenez *et al.*, "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?," *arXiv preprint*, Nov. 2024, doi: 10.48550/arXiv.2310.06770.

[11] A. A. Chien, L. Lin, H. Nguyen, V. Rao, T. Sharma, and R. Wijayawardana, "Reducing the Carbon Impact of Generative AI Inference (today and in 2035)," in *Proceedings of the 2nd Workshop on Sustainable Computer Systems*, in HotCarbon '23. Boston, MA, USA: Association for Computing Machinery, 2023. doi: 10.1145/3604930.3605705.

[12] L. Cruz, “Green Software Engineering Done Right: a Scientific Guide to Set Up Energy Efficiency Experiments.” Accessed: Oct. 05, 2026. [Online]. Available: https://luiscruz.github.io/2021/10/10/scientific-guide.html

[13] N. Jain *et al.*, “LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code,” *arXiv preprint*, Jun. 2024, doi: 10.48550/arXiv.2403.07974.

[14] Z. Wang, Z. Chu, T. V. Doan, S. Ni, M. Yang, and W. Zhang, “History, development, and principles of large language models: an introductory survey,” *AI and Ethics*, vol. 5, no. 3, pp. 1955–1971, Jun. 2025, doi: 10.1007/s43681-024-00583-7.

[15] OpenAI, “Introducing GPT-5.” Accessed: Jan. 05, 2026. [Online]. Available: https://openai.com/index/introducing-gpt-5/

[16] Anthropic, “Introducing Claude Sonnet 4.5.” Accessed: Jan. 06, 2026. [Online]. Available: https://www.anthropic.com/news/claude-sonnet-4-5

[17] S. Pichai, “A new era of intelligence with Gemini 3.” Accessed: Jan. 05, 2026. [Online]. Available: https://blog.google/products/gemini/gemini-3/

[18] V. Liu and Y. Yin, “Green AI: exploring carbon footprints, mitigation strategies, and trade offs in large language model training,” *Discover Artificial Intelligence*, vol. 4, no. 1, 2024, doi: 10.1007/s44163-024-00149-w.

[19] M. F. Argerich and M. Patiño-Martínez, “Measuring and Improving the Energy Efficiency of Large Language Models Inference,” *IEEE Access*, vol. 12, pp. 80194–80207, 2024, doi: 10.1109/ACCESS.2024.3409745.

[20] B. Yan *et al.*, “On protecting the data privacy of Large Language Models (LLMs) and LLM agents: A literature review,” *High-Confidence Computing*, vol. 5, no. 2, 2025, doi: 10.1016/j.hcc.2025.100300.

[21] A. Dorca Josa and M. Bleda-Bejar, “LOCAL LLMS: SAFEGUARDING DATA PRIVACY IN THE AGE OF GENERATIVE AI. A CASE STUDY AT THE UNIVERSITY OF ANDORRA,” in *ICERI2024 Proceedings*, in 17th annual International Conference of Education, Research and Innovation. Seville, Spain: IATED, Nov. 2024, pp. 7879–7888. doi: 10.21125/iceri.2024.1931.

[22] Gemma Team *et al.*, “Gemma: Open Models Based on Gemini Research and Technology,” *arXiv preprint*, 2024, doi: 10.48550/arXiv.2403.08295.

[23] H. Touvron *et al.*, “LLaMA: Open and Efficient Foundation Language Models,” *arXiv preprint*, 2023, doi: 10.48550/arXiv.2302.13971.

[24] C. Elsworth *et al.*, “Measuring the environmental impact of delivering AI at Google Scale,” *arXiv preprint*, 2025, doi: 10.48550/arXiv.2508.15734.

[25] N. Jegham, M. Abdelatti, C. Y. Koh, L. Elmoubarki, and A. Hendawi, "How Hungry is AI? Benchmarking Energy, Water, and Carbon Footprint of LLM Inference," *arXiv preprint*, 2025, doi: 10.48550/arXiv.2505.09598.

[26] A. Singh, N. P. Patel, A. Ehtesham, S. Kumar, and T. T. Khoei, "A Survey of Sustainability in Large Language Models: Applications, Economics, and Challenges," in *2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC)*, 2025, pp. 8–14. doi: 10.1109/CCWC62904.2025.10903774.

[27] W. Kwon *et al.*, "Efficient Memory Management for Large Language Model Serving with PagedAttention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626. doi: 10.1145/3600006.3613165.

[28] J. Choquette, "NVIDIA Hopper H100 GPU: Scaling Performance," *IEEE Micro*, vol. 43, no. 3, pp. 9–17, 2023, doi: 10.1109/MM.2023.3256796.

[29] W. Cai, J. Jiang, F. Wang, J. Tang, S. Kim, and J. Huang, "A Survey on Mixture of Experts in Large Language Models," *IEEE Transactions on Knowledge and Data Engineering*, vol. 37, no. 7, pp. 3896–3915, 2025, doi: 10.1109/TKDE.2025.3554028.

[30] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, "GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 4895–4901. doi: 10.18653/v1/2023.emnlp-main.298.

[31] S. Li *et al.*, "Evaluating Quantized Large Language Models," in *Proceedings of the 41st International Conference on Machine Learning*, in Proceedings of Machine Learning Research, vol. 235. 2024, pp. 28480–28524. [Online]. Available: https://proceedings.mlr.press/v235/li24bb.html

[32] Y. Liu *et al.*, "Efficient Inference for Large Reasoning Models: A Survey," *arXiv preprint*, 2025, doi: 10.48550/arXiv.2503.23077.

[33] J. Wang *et al.*, "A Comprehensive Review of Multimodal Large Language Models: Performance and Challenges Across Different Tasks," *arXiv preprint*, 2024, doi: 10.48550/arXiv.2408.01319.

[34] N. Ratner *et al.*, "Parallel Context Windows for Large Language Models," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023, pp. 6383–6402. doi: 10.18653/v1/2023.acl-long.352.

[35] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *Communications of the ACM*, vol. 63, no. 12, pp. 54–63, 2020, doi: 10.1145/3381831.

[36] V. Bolón-Canedo, L. Morán-Fernández, B. Cancela, and A. Alonso-Betanzos, "A review of green artificial intelligence: Towards a more sustainable future," *Neurocomputing*, vol. 599, 2024, doi: 10.1016/j.neucom.2024.128096.

[37] J. Saad-Falcon *et al.*, "Intelligence per Watt: Measuring Intelligence Efficiency of Local AI," *arXiv preprint*, 2025, doi: 10.48550/arXiv.2511.07885.

[38] C. O'Hagan, "AI Large Language Models: new report shows small changes can reduce energy use by 90%." Accessed: Jan. 10, 2026. [Online]. Available: https://www.unesco.org/en/articles/ai-large-language-models-new-report-shows-small-changes-can-reduce-energy-use-90

[39] D. Matos, "Eco-Evolution of LLMs: Replication Package." [Online]. Available: https://github.com/diogogmatos/eco-evolution-of-llms

[40] A. Brandão, D. Matos, M. Guimarães, S. Cunha, and J. Saraiva, "The Green Side of the Lua," in *2026 IEEE International Conference on Software Analysis, Evolution and Reengineering - Companion (SANER-C)*, 2026, pp. 201–209. doi: 10.1109/SANER-C67878.2026.00033.

[41] D. Matos, A. Brandão, M. Guimarães, S. Cunha, and J. Saraiva, "The Extended Green Side of the Lua," *SSRN preprint*, 2026, doi: 10.2139/ssrn.7661830.

[42] R. Pereira *et al.*, "Ranking programming languages by energy efficiency," *Science of Computer Programming*, vol. 205, 2021, doi: 10.1016/j.scico.2021.102609.

[43] B. R. Keeble, "The Brundtland Report: 'Our Common Future'," *Medicine and War*, vol. 4, no. 1, pp. 17–25, 1988, doi: 10.1080/07488008808408783.

[44] C. C. Venters *et al.*, "Software Sustainability: Beyond the Tower of Babel," in *2021 IEEE/ACM International Workshop on Body of Knowledge for Software Sustainability (BoKSS)*, 2021, pp. 3–4. doi: 10.1109/BoKSS52540.2021.00009.

[45] C. Calero, M. Angeles Moraga, and F. Garcia, "Software, Sustainability, and UN Sustainable Development Goals," *IT Professional*, vol. 24, no. 1, pp. 41–48, 2022, doi: 10.1109/MITP.2021.3117344.

[46] P. Lago, N. Meyer, M. Morisio, H. A. Muller, and G. Scanniello, "2nd International workshop on green and sustainable software (GREENS 2013)," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 1523–1524. doi: 10.1109/ICSE.2013.6606768.

[47] K. Erdélyi, "Special factors of development of green software supporting eco sustainability," in *2013 IEEE 11th International Symposium on Intelligent Systems and Informatics (SISY)*, 2013, pp. 337–340. doi: 10.1109/SISY.2013.6662597.

[48] L. Cruz *et al.*, "Greening AI-enabled Systems with Software Engineering: A Research Agenda for Environmentally Sustainable AI Practices," *ACM SIGSOFT Software Engineering Notes*, vol. 50, no. 3, pp. 14–23, Jul. 2025, doi: 10.1145/3743095.3743099.

[49] A. M. Turing, "Computing Machinery and Intelligence," *Mind*, vol. 59, no. 236, pp. 433–460, Oct. 1950, doi: 10.1093/mind/LIX.236.433.

[50] L. R. Bahl, P. F. Brown, P. V. De Souza, and R. L. Mercer, "A Tree-Based Statistical Language Model for Natural Language Speech Recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 7, pp. 1001–1008, 1989, doi: 10.1109/29.32278.

[51] F. Jelinek, *Statistical Methods for Speech Recognition*. in Language, Speech, and Communication. Cambridge, MA, USA: MIT Press, 2022.

[52] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A Neural Probabilistic Language Model," *Journal of Machine Learning Research*, vol. 3, no. 6, pp. 1137–1155, 2003, doi: 10.1162/153244303322533223.

[53] C. Raffel *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020, [Online]. Available: https://jmlr.org/papers/v21/20-074.html

[54] L. Xue *et al.*, "mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021, pp. 483–498. doi: 10.18653/v1/2021.naacl-main.41.

[55] J. Wei *et al.*, "Emergent Abilities of Large Language Models," *Transactions on Machine Learning Research*, 2022, [Online]. Available: https://openreview.net/forum?id=yzkSU5zdwD

[56] A. Radford *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019, [Online]. Available: https://api.semanticscholar.org/CorpusID:160025533

[57] J. Kaplan *et al.*, "Scaling Laws for Neural Language Models," *arXiv preprint*, 2020, doi: 10.48550/arXiv.2001.08361.

[58] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "NVIDIA A100 Tensor Core GPU: Performance and Innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021, doi: 10.1109/MM.2021.3061394.

[59] A. Tirumala and R. Wong, " NVIDIA Blackwell Platform: Advancing Generative AI and Accelerated Computing ," in *2024 IEEE Hot Chips 36 Symposium (HCS)*, Los Alamitos, CA, USA: IEEE Computer Society, Aug. 2024, pp. 1–33. doi: 10.1109/HCS61935.2024.10665247.

[60] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[61] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized autoregressive pretraining for language understanding," in *Advances in Neural Information Processing Systems*, 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/dc6a7e655d7e5840e66733e9ee67cc69-Abstract.html

[62] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-XL: Attentive language models beyond a fixed-length context," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 2978–2988. doi: 10.18653/v1/P19-1285.

[63] P. Villalobos, J. Sevilla, T. Besiroglu, L. Heim, A. Ho, and M. Hobbhahn, "Machine Learning Model Sizes and the Parameter Gap," *arXiv preprint*, 2022, doi: 10.48550/arXiv.2207.02852.

[64] D. Patel, "GPT-4 Architecture, Infrastructure, Training Dataset, Costs, Vision, MoE." Accessed: Jan. 22, 2026. [Online]. Available: https://newsletter.semianalysis.com/p/gpt-4-architecture-infrastructure

[65] D. Patterson *et al.*, "Carbon Emissions and Large Neural Network Training," *arXiv preprint*, 2021, doi: 10.48550/arXiv.2104.10350.

[66] J. McDonald, B. Li, N. Frey, D. Tiwari, V. Gadepally, and S. Samsi, "Great Power, Great Responsibility: Recommendations for Reducing Energy for Training Language Models," in *Findings of the Association for Computational Linguistics: NAACL 2022*, 2022, pp. 1962–1970. doi: 10.18653/v1/2022.findings-naacl.151.

[67] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for modern deep learning research," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, pp. 13693–13696. doi: 10.1609/aaai.v34i09.7123.

[68] Hugging Face, "Displaying carbon emissions for your model." Accessed: Jan. 22, 2026. [Online]. Available: https://huggingface.co/docs/hub/model-cards-co2

[69] C. Stokel-Walker, "The Generative AI Race Has a Dirty Secret," *Wired*, Feb. 2023, Accessed: Jan. 22, 2026. [Online]. Available: https://www.wired.com/story/the-generative-ai-search-race-has-a-dirty-secret/

[70] A. Agrawal, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani, and R. Ramjee, "SARATHI: Efficient LLM Inference by Piggybacking Decodes with Chunked Prefills," *arXiv preprint*, 2023, doi: 10.48550/arXiv.2308.16369.

[71] J. Fernandez, C. Na, V. Tiwari, Y. Bisk, S. Luccioni, and E. Strubell, "Energy Considerations of Large Language Model Inference and Efficiency Optimizations," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL 2025)*, 2025. [Online]. Available: https://aclanthology.org/2025.acl-long.1563/

[72] Z. Lu *et al.*, "Small Language Models: Survey, Measurements, and Insights," *arXiv preprint*, 2024, doi: 10.48550/arXiv.2409.15790.

[73] J.-W. Chung *et al.*, "The ML.ENERGY Benchmark: Toward Automated Inference Energy Measurement and Optimization," in *39th Conference on Neural Information Processing Systems (NeurIPS 2025), Track on Datasets and Benchmarks*, 2025. [Online]. Available: https://arxiv.org/abs/2505.06371

[74] J. Wei *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, in NIPS '22. New Orleans, LA, USA: Curran Associates Inc., 2022.

[75] OpenAI, "Learning to reason with LLMs." Accessed: Oct. 04, 2026. [Online]. Available: https://openai.com/index/learning-to-reason-with-llms/

[76] D. Guo *et al.*, "DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning," *Nature*, vol. 645, no. 8081, pp. 633–638, Sep. 2025, doi: 10.1038/s41586-025-09422-z.

[77] A. Yang *et al.*, "Qwen3 Technical Report," *arXiv preprint*, 2025, doi: 10.48550/arXiv.2505.09388.

[78] S. Luccioni and B. Gamazaychikov, "AI Energy Score v2: Refreshed Leaderboard, now with Reasoning." Accessed: Oct. 05, 2026. [Online]. Available: https://huggingface.co/blog/sasha/ai-energy-score-v2

[79] M. Dauner and G. Socher, "Energy costs of communicating with AI," *Frontiers in Communication*, vol. 10, p. 1572947, 2025, doi: 10.3389/fcomm.2025.1572947.

[80] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory power estimation and capping," in *Proc. Int. Symp. Low Power Electron. Des.*, 2010, pp. 189–194. doi: 10.1145/1840845.1840883.

[81] NVIDIA Corporation, "NVIDIA System Management Interface." Accessed: Jan. 26, 2026. [Online]. Available: https://developer.nvidia.com/system-management-interface

[82] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "RAPL in action: Experiences in using RAPL for power measurements," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 3, no. 2, 2018, doi: 10.1145/3177754.

[83] S. Desrochers, C. Paradis, and V. M. Weaver, "A Validation of DRAM RAPL Power Measurements," in *Proceedings of the Second International Symposium on Memory Systems*, in MEMSYS '16. ACM, 2016, pp. 455–470. doi: 10.1145/2989081.2989088.

[84] Z. Yang, K. Adamek, and W. Armour, "Accurate and Convenient Energy Measurements for GPUs: A Detailed Study of NVIDIA GPU's Built-In Power Sensor," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024, pp. 1–17. doi: 10.1109/SC41406.2024.00028.

[85] M. Breyer, A. Van Craen, P. Domanski, and D. Pflüger, "hws: A Tool for Monitoring Hardware Metrics Across Diverse Vendors: A Case Study on Hyperparameter Optimization Algorithms," in *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024, pp. 1815–1825. doi: 10.1109/SCW63240.2024.00228.

[86] M. Jay, V. Ostapenco, L. Lefevre, D. Trystram, A.-C. Orgerie, and B. Fichel, "An experimental comparison of software-based power meters: Focus on CPU and GPU," in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2023, pp. 106–118. doi: 10.1109/CCGrid57682.2023.00020.

[87] J. Sallou, L. Cruz, and T. Durieux, "EnergiBridge: Empowering Software Sustainability through Cross-Platform Energy Measurement," *arXiv preprint*, 2023, doi: 10.48550/arXiv.2312.13897.

[88] L. Bouza, A. Bugeau, and L. Lannelongue, "How to estimate carbon footprint when training deep learning models? A guide and review," *Environmental Research Communications*, vol. 5, no. 11, p. 115014, Nov. 2023, doi: 10.1088/2515-7620/acf81b.

[89] R. Fischer, "Ground-truthing AI energy consumption: validating CodeCarbon against external measurements," *it - Information Technology*, vol. 67, no. 4–6, pp. 155–163, 2026, doi: 10.1515/itit-2025-0031.

[90] L. F. W. Anthony, B. Kanding, and R. Selvan, "Carbontracker: Tracking and Predicting the Carbon Footprint of Training Deep Learning Models," *arXiv preprint*, Jul. 2020, doi: 10.48550/arXiv.2007.03051.

[91] T. Allen, "EnviroLLM: Resource Tracking and Optimization for Local AI," *arXiv preprint*, Dec. 2025, doi: 10.48550/arXiv.2512.12004.

[92] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, 2012, doi: 10.1109/MM.2012.12.

[93] G. Raffin and D. Trystram, “Dissecting the Software-Based Measurement of CPU Energy Consumption: A Comparative Analysis,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 1, pp. 96–107, 2025, doi: 10.1109/TPDS.2024.3492336.

[94] J. Pan, “RAPL (Running Average Power Limit) driver.” Accessed: Jan. 28, 2026. [Online]. Available: https://lwn.net/Articles/545745/

[95] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, “An Energy Efficiency Feature Survey of the Intel Haswell Processor,” in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, IEEE, 2015, pp. 896–904. doi: 10.1109/IPDPSW.2015.70.

[96] F. Castor, “Estimating the Energy Footprint of Software Systems: a Primer,” *arXiv preprint*, Sep. 2026, doi: 10.48550/arXiv.2407.11611.

[97] C. Imes, “RAPLCap: RAPL power capping C interface with multiple implementations.” Accessed: Jan. 28, 2026. [Online]. Available: https://github.com/powercap/raplcap

[98] C. Belgaid, A. d'Azémar, G. Fieni, and R. Rouvoy, “pyRAPL: A library to measure the energy consumption of Python code.” Accessed: Oct. 16, 2026. [Online]. Available: https://github.com/powerapi-ng/pyRAPL

[99] A. Servetto, “jRAPL: An energy monitoring API in Java.” Accessed: Jan. 28, 2026. [Online]. Available: https://github.com/aservet1/jRAPL

[100] J. Aaldering, “RAPL Energy: A Rust library for getting the energy consumption from RAPL.” Accessed: Jan. 28, 2026. [Online]. Available: https://github.com/JordyAaldering/rapl-energy

[101] NVIDIA Corporation, “NVIDIA Management Library (NVML) API Reference Guide.” Accessed: Aug. 25, 2026. [Online]. Available: http://docs.nvidia.com/deploy/nvml-api/index.html

[102] NVIDIA, “NVIDIA-SMI Documentation.” Accessed: Jan. 28, 2026. [Online]. Available: https://docs.nvidia.com/deploy/nvidia-smi/index.html

[103] NVIDIA Corporation, “nvidia-ml-py: Python Bindings for the NVIDIA Management Library.” Accessed: Oct. 16, 2026. [Online]. Available: https://pypi.org/project/nvidia-ml-py/

[104] J. Samic and D. Carlier, “nvml-wrapper: Safe Rust wrapper for the NVIDIA Management Library.” Accessed: Jan. 28, 2026. [Online]. Available: https://github.com/rust-nvml/nvml-wrapper

[105] C. Wassermann, T. Dollenbacher, C. Terboven, and M. S. Müller, “Leveraging NVML GPM for NVIDIA GPU Monitoring,” in *Proceedings of the Supercomputing Asia and International Conference on High Performance Computing in Asia Pacific Region Workshops*, in SCA/HPCAsiaWS '26. New York, NY, USA: Association for Computing Machinery, Jan. 2026, pp. 59–68. doi: 10.1145/3784828.3785156.

[106] CodeCarbon, "CodeCarbon 3.2.2 documentation." Accessed: Feb. 06, 2026. [Online]. Available: https://mlco2.github.io/codecarbon/

[107] AMD, "AMD SMI 26.2.1 documentation." Accessed: Jan. 28, 2026. [Online]. Available: https://rocm.docs.amd.com/projects/amdsmi/en/latest/index.html

[108] NVIDIA Corporation, "GPU Deployment Kit." Accessed: Jan. 29, 2026. [Online]. Available: https://developer.nvidia.com/gpu-deployment-kit

[109] NVIDIA, "NVIDIA's Next Generation CUDA Compute Architecture: Fermi." Accessed: Jan. 29, 2026. [Online]. Available: https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

[110] Hubblo, "Scaphandre documentation: Introduction." Accessed: Jan. 29, 2026. [Online]. Available: https://hubblo-org.github.io/scaphandre-documentation/index.html

[111] Hubblo, "Scaphandre documentation: Explanations about host level power and energy metrics." Accessed: Jan. 29, 2026. [Online]. Available: https://hubblo-org.github.io/scaphandre-documentation/explanations/host_metrics.html

[112] Hubblo, "Scaphandre documentation: How Scaphandre computes per process power consumption." Accessed: Jan. 29, 2026. [Online]. Available: https://hubblo-org.github.io/scaphandre-documentation/explanations/how-scaph-computes-per-process-power-consumption.html

[113] G. Kp, G. Pierre, and R. Rouvoy, "Studying the Energy Consumption of Stream Processing Engines in the Cloud," in *2023 IEEE International Conference on Cloud Engineering (IC2E)*, 2023, pp. 99–106. doi: 10.1109/IC2E59103.2023.00019.

[114] P. Serrano-Gutierrez and I. Ayala, "Using Energy Consumption for Self-adaptation in FaaS," in *Reuse and Software Quality*, A. Achilleos, L. Fuentes, and G. A. Papadopoulos, Eds., Cham: Springer Nature Switzerland, 2024, pp. 123–134.

[115] Intel Corporation, "Discontinued: Intel® Power Gadget." Accessed: Feb. 06, 2026. [Online]. Available: https://www.intel.com/content/www/us/en/developer/archive/tools/power-gadget.html

[116] Z. Ournani, M. C. Belgaid, R. Rouvoy, P. Rust, J. Penhoat, and L. Seinturier, "Taming Energy Consumption Variations in Systems Benchmarking," in *Proceedings of the 11th ACM/SPEC International Conference on Performance Engineering (ICPE '20)*, Association for Computing Machinery, 2020. doi: 10.1145/3358960.3379142.

[117] M. A. Bokhari, B. Alexander, and M. Wagner, "Towards Rigorous Validation of Energy Optimisation Experiments," *arXiv preprint*, 2020, doi: 10.48550/arXiv.2004.04500.

[118] Y. Jin, G.-Y. Wei, and D. Brooks, "The Energy Cost of Reasoning: Analyzing Energy Usage in LLMs with Test-time Compute," *arXiv preprint*, 2025, doi: 10.48550/arXiv.2505.14733.

[119] N. Alizadeh, B. Belchev, N. Saurabh, P. Kelbert, and F. Castor, "The Cost of AI-Assisted Coding: An Extensive Investigation of Energy vs. Accuracy in Language Models," *arXiv preprint*, 2026, doi: 10.48550/arXiv.2412.00329.

[120] M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, "A Comparative Study of Methods for Measurement of Energy of Computing," *Energies*, vol. 12, no. 11, p. 2204, 2019, doi: 10.3390/en12112204.

[121] M. Karsten, A. C. Dragomir, R. Apsan, V. Stoico, and I. Malavolta, "Experiment Runner: a Tool for the Automatic Orchestration of Experiments Targeting Software Systems," *Science of Computer Programming*, vol. 1, p. 103415, 2025, doi: https://doi.org/10.1016/j.scico.2025.103415.

[122] M. Abdin *et al.*, "Phi-4 Technical Report," *arXiv preprint*, 2024, doi: 10.48550/arXiv.2412.08905.

[123] A. H. Liu *et al.*, "Ministral 3," *arXiv preprint*, 2026, doi: 10.48550/arXiv.2601.08584.

[124] R. Gray and D. Neuhoff, "Quantization," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2325–2383, 1998, doi: 10.1109/18.720541.

[125] J. Lang, Z. Guo, and S. Huang, "A Comprehensive Study on Quantization Techniques for Large Language Models," in *2024 4th International Conference on Artificial Intelligence, Robotics, and Communication (ICAIRC)*, 2024, pp. 224–231. doi: 10.1109/ICAIRC64177.2024.10899941.

[126] ggml, "llama.cpp: LLM inference in C/C++." Accessed: Feb. 08, 2026. [Online]. Available: https://github.com/ggml-org/llama.cpp

[127] Turboderp, "ExLlamaV2: A fast inference library for running LLMs locally on modern consumer-class GPUs." Accessed: Oct. 18, 2026. [Online]. Available: https://github.com/turboderp-org/exllamav2

[128] Ollama, "Ollama: Run AI models locally and in the cloud." Accessed: Feb. 08, 2026. [Online]. Available: https://github.com/ollama/ollama

[129] Red Hat AI and vLLM Project, "LLM Compressor: LLM compression for production inference with vLLM." Accessed: Aug. 20, 2026. [Online]. Available: https://github.com/vllm-project/llm-compressor

[130] C. Hansen, "AutoAWQ: An implementation of the AWQ algorithm for 4-bit quantization." Accessed: Aug. 20, 2026. [Online]. Available: https://github.com/casper-hansen/AutoAWQ

[131] vLLM, "vLLM: A high-throughput and memory-efficient inference and serving engine for LLMs." Accessed: Aug. 25, 2026. [Online]. Available: https://github.com/vllm-project/vllm

[132] N. Shazeer *et al.*, "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," in *International Conference on Learning Representations (ICLR)*, Feb. 2017. Accessed: Aug. 20, 2026. [Online]. Available: https://openreview.net/forum?id=B1ckMDqlg

[133] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: scaling to trillion parameter models with simple and efficient sparsity," *The Journal of Machine Learning Research*, vol. 23, no. 1, p. 120:5232–120:5270, Jan. 2022, Accessed: Aug. 20, 2026. [Online]. Available: https://dl.acm.org/doi/10.5555/3586589.3586709

[134] A. Q. Jiang *et al.*, "Mixtral of Experts," *arXiv preprint*, Jan. 2024, doi: 10.48550/arXiv.2401.04088.

[135] Google, "Gemma 3n model overview." Accessed: Oct. 04, 2026. [Online]. Available: https://ai.google.dev/gemma/docs/gemma-3n

[136] Google, "Gemma 4 model card." Accessed: Oct. 04, 2026. [Online]. Available: https://ai.google.dev/gemma/docs/core/model_card_4

[137] S. Park, S. Jeon, C. Lee, S. Jeon, B.-S. Kim, and J. Lee, "A Survey on Inference Engines for Large Language Models: Perspectives on Optimization and Efficiency," *arXiv preprint*, 2025, doi: 10.48550/arXiv.2505.01658.

[138] K. T. Chitty-Venkata *et al.*, "LLM-Inference-Bench: Inference Benchmarking of Large Language Models on AI Accelerators," in *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, 2024, pp. 1362–1379. doi: 10.1109/SCW63240.2024.00178.

[139] NVIDIA Corporation, "TensorRT-LLM: An easy-to-use Python API to define large language models and perform inference efficiently on NVIDIA GPUs." Accessed: Feb. 08, 2026. [Online]. Available: https://github.com/NVIDIA/TensorRT-LLM

[140] NVIDIA Corporation, "NVIDIA TensorRT: An SDK for high-performance deep learning inference on NVIDIA GPUs." Accessed: Feb. 08, 2026. [Online]. Available: https://github.com/NVIDIA/TensorRT

[141] D. Rein *et al.*, "GPQA: A Graduate-Level Google-Proof Q&A Benchmark," *arXiv preprint*, Nov. 2023, doi: 10.48550/arXiv.2311.12022.

[142] D. Hendrycks *et al.*, "Measuring Massive Multitask Language Understanding," *arXiv preprint*, 2021, doi: 10.48550/arXiv.2009.03300.

[143] D. Hendrycks *et al.*, "Measuring Mathematical Problem Solving With the MATH Dataset," *arXiv preprint*, 2021, doi: 10.48550/arXiv.2103.03874.

[144] Y. Zhang and Math-AI Team, "American Invitational Mathematics Examination (AIME) 2025." [Online]. Available: https://huggingface.co/datasets/math-ai/aime25

[145] M. Chen *et al.*, "Evaluating Large Language Models Trained on Code," *arXiv preprint*, 2021, doi: 10.48550/arXiv.2107.03374.

[146] LeetCode, "LeetCode: The world's leading online programming learning platform." Accessed: Aug. 04, 2026. [Online]. Available: https://leetcode.com/

[147] AtCoder Inc., "AtCoder: A programming contest site for anyone from beginners to experts." Accessed: Aug. 04, 2026. [Online]. Available: https://atcoder.jp/

[148] "Codeforces: Programming competitions and contests, programming community." Accessed: Feb. 08, 2026. [Online]. Available: https://codeforces.com/

[149] ICPC Foundation, "The ICPC International Collegiate Programming Contest." Accessed: Feb. 08, 2026. [Online]. Available: https://icpc.global/

[150] "International Olympiad in Informatics." Accessed: Feb. 08, 2026. [Online]. Available: https://ioinformatics.org/

[151] N. Chowdhury *et al.*, "Introducing SWE-bench Verified." Accessed: Aug. 20, 2026. [Online]. Available: https://openai.com/index/introducing-swe-bench-verified/

[152] A. Cheng *et al.*, "The FACTS Leaderboard: A Comprehensive Benchmark for Large Language Model Factuality," *arXiv preprint*, Dec. 2025, doi: 10.48550/arXiv.2512.10791.

[153] T. A. Chang *et al.*, "Global PIQA: Evaluating Physical Commonsense Reasoning Across 100+ Languages and Cultures," *arXiv preprint*, Oct. 2025, doi: 10.48550/arXiv.2510.24081.

[154] A. Wang *et al.*, "SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/4496bf24afe7fab6f046bf4923da8de6-Paper.pdf

[155] K. Anjum *et al.*, "Domain Specific Benchmarks for Evaluating Multimodal Large Language Models," *arXiv preprint*, Jun. 2025, doi: 10.48550/arXiv.2506.12958.

[156] Google DeepMind, "Gemini 3.1 Pro." Accessed: Oct. 04, 2026. [Online]. Available: https://deepmind.google/models/gemini/pro/

[157] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete Problems in AI Safety," *arXiv preprint*, 2016, doi: 10.48550/arXiv.1606.06565.

[158] A. Cotra, "The case for aligning narrowly superhuman models – AI Alignment Forum," Mar. 2021, Accessed: Jul. 15, 2026. [Online]. Available: https://www.alignmentforum.org/posts/PZtsoaoSLpKjjbMqM/the-case-for-aligning-narrowly-superhuman-models

[159] Hugging Face, "Open LLM Leaderboard: About." Accessed: Oct. 04, 2026. [Online]. Available: https://huggingface.co/docs/leaderboards/open_llm_leaderboard/about

[160] W. Chen *et al.*, "TheoremQA: A Theorem-driven Question Answering dataset," *arXiv preprint*, 2023, doi: 10.48550/arXiv.2305.12524.

[161] X. Wang *et al.*, "SciBench: Evaluating College-Level Scientific Problem-Solving Abilities of Large Language Models," *arXiv preprint*, 2024, doi: 10.48550/arXiv.2307.10635.

[162] Hugging Face, "Open LLM Leaderboard 2 collection." Accessed: Jul. 17, 2026. [Online]. Available: https://huggingface.co/collections/open-llm-leaderboard/open-llm-leaderboard-2

[163] S. Toshniwal, W. Du, I. Moshkov, B. Kisacanin, A. Ayrapetyan, and I. Gitman, "OpenMathInstruct-2: Accelerating AI for Math with Massive Open-Source Instruction Data," *arXiv preprint*, 2024, doi: 10.48550/arXiv.2410.01560.

[164] Art of Problem Solving, "AoPS Wiki: American Invitational Mathematics Examination." Accessed: Feb. 08, 2026. [Online]. Available: https://artofproblemsolving.com/wiki/index.php/American_Invitational_Mathematics_Examination

[165] Art of Problem Solving, "AoPS Wiki: AMC 10." Accessed: Jul. 23, 2026. [Online]. Available: https://artofproblemsolving.com/wiki/index.php?title=AMC_10

[166] Art of Problem Solving, "AoPS Wiki: AMC 12." Accessed: Jul. 23, 2026. [Online]. Available: https://artofproblemsolving.com/wiki/index.php?title=AMC_12

[167] Art of Problem Solving, "AoPS Wiki: 2025 AIME I." Accessed: Jul. 23, 2026. [Online]. Available: https://artofproblemsolving.com/wiki/index.php/2025_AIME_I

[168] Art of Problem Solving, "AoPS Wiki: 2025 AIME II." Accessed: Jul. 23, 2026. [Online]. Available: https://artofproblemsolving.com/wiki/index.php/2025_AIME_II

[169] LLM Stats, "AIME 2025 Leaderboard." Accessed: Jul. 23, 2026. [Online]. Available: https://llm-stats.com/benchmarks/aime-2025

[170] xAI, "Grok 4." Accessed: Jul. 23, 2026. [Online]. Available: https://x.ai/news/grok-4

[171] Google DeepMind, "Gemini 3." Accessed: Feb. 08, 2026. [Online]. Available: https://deepmind.google/models/gemini/

[172] GitHub, "GitHub Copilot: Your AI pair programmer." Accessed: Jul. 23, 2026. [Online]. Available: https://github.com/features/copilot

[173] S. Kulal *et al.*, "SPoC: search-based pseudocode to code," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, no. 1066, Red Hook, NY, USA: Curran Associates Inc., 2019, pp. 11906–11917. Accessed: Jul. 23, 2026. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/7298332f04ac004a0ca44cc69ecf6f6b-Paper.pdf

[174] F. Ribeiro, C. Spiess, P. Devanbu, and S. Nadi, "On LLMs' Internal Representation of Code Correctness," in *Proceedings of the 2026 IEEE/ACM 48th International Conference on Software Engineering*, in ICSE '26. Rio de Janeiro, Brazil: Association for Computing Machinery, 2026, pp. 1160–1172. doi: 10.1145/3744916.3787846.

[175] Q. Zheng *et al.*, "CodeGeeX: A Pre-Trained Model for Code Generation with Multilingual Benchmarking on HumanEval-X," *arXiv preprint*, 2024, doi: 10.48550/arXiv.2303.17568.

[176] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation," in *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023. [Online]. Available: https://arxiv.org/abs/2305.01210

[177] EvalPlus, "EvalPlus Leaderboard." Accessed: Aug. 04, 2026. [Online]. Available: https://evalplus.github.io/leaderboard.html

[178] EvalPlus, "Release repository for MBPP+ data." Accessed: Aug. 04, 2026. [Online]. Available: https://github.com/evalplus/mbppplus_release

[179] Z. Yu, Y. Zhao, A. Cohan, and X.-P. Zhang, "HumanEval Pro and MBPP Pro: Evaluating Large Language Models on Self-invoking Code Generation," *arXiv preprint*, Dec. 2024, doi: 10.48550/arXiv.2412.21199.

[180] LLM Stats, "MBPP Leaderboard." Accessed: Aug. 04, 2026. [Online]. Available: https://llm-stats.com/benchmarks/mbpp

[181] LiveCodeBench, "LiveCodeBench Leaderboard - Holistic and Contamination Free Evaluation." Accessed: Aug. 04, 2026. [Online]. Available: https://livecodebench.github.io/leaderboard.html

[182] OpenAI, "Introducing OpenAI o3 and o4-mini." Accessed: Aug. 04, 2026. [Online]. Available: https://openai.com/index/introducing-o3-and-o4-mini/

[183] F. Ribeiro, J. N. C. de Macedo, K. Tsushima, R. Abreu, and J. Saraiva, "GPT-3-Powered Type Error Debugging: Investigating the Use of Large Language Models for Code Repair," in *Proceedings of the 16th ACM SIGPLAN International Conference on Software Language Engineering*, in SLE 2023. Cascais, Portugal: Association for Computing Machinery, 2023, pp. 111–124. doi: 10.1145/3623476.3623522.

[184] Pallets, "Flask: The Python micro framework for building web applications." Accessed: Aug. 20, 2026. [Online]. Available: https://github.com/pallets/flask

[185] Django Software Foundation, "Django: The web framework for perfectionists with deadlines." Accessed: Aug. 20, 2026. [Online]. Available: https://github.com/django/django

[186] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python." Accessed: Aug. 20, 2026. [Online]. Available: https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

[187] A. Meurer *et al.*, "SymPy: symbolic computing in Python," *PeerJ Computer Science*, vol. 3, p. e103, Jan. 2017, doi: 10.7717/peerj-cs.103.

[188] SWE-bench, "SWE-bench Verified." Accessed: Aug. 20, 2026. [Online]. Available: https://www.swebench.com/verified.html

[189] SWE-bench, "SWE-bench Leaderboards." Accessed: Aug. 20, 2026. [Online]. Available: https://www.swebench.com/index.html

[190] Y. Bisk, R. Zellers, R. Le Bras, J. Gao, and Y. Choi, "PIQA: Reasoning about Physical Commonsense in Natural Language," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. [Online]. Available: https://arxiv.org/abs/1911.11641

[191] P. He, X. Liu, J. Gao, and W. Chen, "DeBERTa: Decoding-enhanced BERT with Disentangled Attention," in *International Conference on Learning Representations (ICLR)*, 2021. [Online]. Available: https://arxiv.org/abs/2006.03654

[192] S. Samsi *et al.*, "From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference," *arXiv preprint*, 2023, doi: 10.48550/arXiv.2310.03003.

[193] A. S. Luccioni, Y. Jernite, and E. Strubell, "Power Hungry Processing: Watts Driving the Cost of AI Deployment?," in *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency (FAccT '24)*, Association for Computing Machinery, 2024. doi: 10.1145/3630106.3658542.

[194] C. Niu *et al.*, "TokenPowerBench: Benchmarking the Power Consumption of LLM Inference," *arXiv preprint*, 2025, doi: 10.48550/arXiv.2512.03024.

[195] E. J. Husom, A. Goknil, L. K. Shar, and S. Sen, "The Price of Prompting: Profiling Energy Use in Large Language Models Inference," *arXiv preprint*, 2024, doi: 10.48550/arXiv.2407.16893.

[196] F. Caravaca, Á. Cuevas, and R. Cuevas, "From Prompts to Power: Measuring the Energy Footprint of LLM Inference," *arXiv preprint*, 2025, doi: 10.48550/arXiv.2511.05597.

[197] S. Poddar, P. Koley, J. Misra, S. Podder, N. Ganguly, and S. Ghosh, "Towards Sustainable NLP: Insights from Benchmarking Inference Energy in Large Language Models," in *NAACL 2025 Main Conference*, 2025. [Online]. Available: https://arxiv.org/abs/2502.05610

[198] S. Cunha, F. Ribeiro, L. Cruz, and J. Saraiva, "Not All Local LLMs Are Equal: A Benchmark of Energy and Performance," in *Proceedings of the IEEE/ACM 10th International Workshop on Green and Sustainable Software*, in GREENS '26. New York, NY, USA: Association for Computing Machinery, Jul. 2026, pp. 99–106. doi: 10.1145/3786148.3788630.

[199] D. Hernandez and T. B. Brown, "Measuring the Algorithmic Efficiency of Neural Networks," *arXiv preprint*, 2020, doi: 10.48550/arXiv.2005.04305.

[200] A. Ho *et al.*, "Algorithmic progress in language models," *arXiv preprint*, 2024, doi: 10.48550/arXiv.2403.05812.

[201] H. Gundlach, J. Lynch, M. Mertens, and N. Thompson, "The Price of Progress: Price Performance and the Future of AI," *arXiv preprint*, 2025, doi: 10.48550/arXiv.2511.23455.

[202] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning," *Sustainable Computing: Informatics and Systems*, vol. 38, p. 100857, 2023, doi: 10.1016/j.suscom.2023.100857.

[203] A. Tschand *et al.*, "MLPerf Power: Benchmarking the Energy Efficiency of Machine Learning Systems from Microwatts to Megawatts for Sustainable AI," in *2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2025. [Online]. Available: https://arxiv.org/abs/2410.12032

[204] F. Oviedo *et al.*, "Energy Use of AI Inference, Efficiency Pathways, and Test-Time Scaling," *Joule*, p. 102430, 2026, doi: 10.1016/j.joule.2026.102430.

[205] Google, "DiffusionGemma model card." Accessed: Oct. 06, 2026. [Online]. Available: https://huggingface.co/google/diffusiongemma-26B-A4B-it

[206] G. F. Jenks, "The Data Model Concept in Statistical Mapping," *International Yearbook of Cartography*, vol. 7, pp. 186–190, 1967.

[207] TIGER Lab, "MMLU-Pro evaluation script for local models (evaluate_from_local.py)." Accessed: Sep. 24, 2026. [Online]. Available: https://github.com/TIGER-AI-Lab/MMLU-Pro/blob/main/evaluate_from_local.py

[208] vLLM, "vLLM documentation: Reproducibility." Accessed: Oct. 08, 2026. [Online]. Available: https://docs.vllm.ai/en/stable/usage/reproducibility/

[209] W. G. Cochran, *Sampling Techniques*, 3rd ed. New York: John Wiley & Sons, 1977.

[210] J. A. Butts and G. S. Sohi, "A static power model for architects," in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, in MICRO 33. New York, NY, USA: Association for Computing Machinery, Dec. 2000, pp. 191–201. doi: 10.1145/360128.360148.

[211] C. Spearman, "The Proof and Measurement of Association between Two Things," *The American Journal of Psychology*, vol. 15, no. 1, pp. 72–101, 1904, doi: 10.2307/1412159.

[212] Q. McNemar, "Note on the sampling error of the difference between correlated proportions or percentages," *Psychometrika*, vol. 12, no. 2, pp. 153–157, 1947, doi: 10.1007/BF02295996.

[213] Hugging Face, "huggingface_hub: Client library to download and publish models, datasets and other repos on the huggingface.co hub." Accessed: Sep. 12, 2026. [Online]. Available: https://pypi.org/project/huggingface-hub/

[214] Hugging Face, "Safetensors: Simple, safe way to store and distribute tensors." Accessed: Sep. 12, 2026. [Online]. Available: https://github.com/safetensors/safetensors

[215] S. Gugger *et al.*, "Accelerate: Training and inference at scale made simple, efficient and adaptable." Accessed: Oct. 16, 2026. [Online]. Available: https://github.com/huggingface/accelerate

[216] M. Viry, "jenkspy: Compute natural breaks (Fisher-Jenks algorithm)." Accessed: Sep. 15, 2026. [Online]. Available: https://pypi.org/project/jenkspy/

[217] Astral, "uv: An extremely fast Python package and project manager, written in Rust." Accessed: Aug. 25, 2026. [Online]. Available: https://github.com/astral-sh/uv

[218] Mistral AI, "Ministral 3 14B Instruct 2512 model card." Accessed: Oct. 07, 2026. [Online]. Available: https://huggingface.co/mistralai/Ministral-3-14B-Instruct-2512

[219] Eurostat, "Electricity and heat statistics." Accessed: Oct. 07, 2026. [Online]. Available: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Electricity_and_heat_statistics

[220] EV Database, "Energy consumption of full electric vehicles." Accessed: Oct. 07, 2026. [Online]. Available: https://ev-database.org/cheatsheet/energy-consumption-electric-car

[221] United States Environmental Protection Agency, "Greenhouse Gas Equivalencies Calculator - Calculations and References." Accessed: Oct. 07, 2026. [Online]. Available: https://www.epa.gov/energy/greenhouse-gas-equivalencies-calculator-calculations-and-references

[222] Qwen Team, "Qwen1.5-MoE-A2.7B model card." Accessed: Oct. 12, 2026. [Online]. Available: https://huggingface.co/Qwen/Qwen1.5-MoE-A2.7B

[223] Qwen Team, "Qwen3-8B model card." Accessed: Oct. 12, 2026. [Online]. Available: https://huggingface.co/Qwen/Qwen3-8B

[224] Qwen Team, "Qwen3.5-9B model card." Accessed: Oct. 12, 2026. [Online]. Available: https://huggingface.co/Qwen/Qwen3.5-9B

[225] DeepSeek-AI, "DeepSeek-R1 model card." Accessed: Oct. 12, 2026. [Online]. Available: https://huggingface.co/deepseek-ai/DeepSeek-R1

[226] Microsoft, "Phi-4-reasoning model card." Accessed: Oct. 12, 2026. [Online]. Available: https://huggingface.co/microsoft/Phi-4-reasoning

[227] Microsoft, "Phi-4-multimodal-instruct model card." Accessed: Oct. 12, 2026. [Online]. Available: https://huggingface.co/microsoft/Phi-4-multimodal-instruct

[228] Universidade do Minho, “Despacho RT-80/2026, Anexo 4: Declaração a incluir na tese de doutoramento (ou equivalente) ou no trabalho de mestrado.” Accessed: Oct. 15, 2026. [Online]. Available: https://alunos.uminho.pt/PT/estudantes/Formataes/3_Despacho%20RT-80-2026-Anexo-4_DeclaracaoIntegridade_e_IA.pdf

[229] Universidade do Minho, “Recomendações Gerais da Universidade do Minho para o uso da Inteligência Artificial no Ensino, na Aprendizagem e na Avaliação.” Accessed: Oct. 15, 2026. [Online]. Available: https://idea.uminho.pt/pt/Documents/Recomendacoes_IA_UMinho_v3-1.pdf

[230] CodeCarbon, “CodeCarbon: Track emissions from compute and recommend ways to reduce their impact on the environment.” Accessed: Jan. 28, 2026. [Online]. Available: https://codecarbon.io/

Appendix A

Details of Results

This appendix complements Chapter 6. It lists the results of every model run, and gathers the figures that support the analysis but are not needed to follow it.

Results of Every Model Run

Table A.1 and Table A.2 list the results of the 53 model runs. Each value is the median of the repetitions of the model run. The time per question is the mean duration of the inferences of a run. The last column gives the share of its responses that reached the output limit (Section 4.4.1).

Model Run	**Released**	**Accuracy** (%)	**Energy per Question** (J)	**Time per Question** (s)	**Tokens per Question**	**Energy per Token** (J)	**Correct Answers per kJ**	**At the Limit** (%)
Tiny/Mobile								
Qwen1.5-4B-Chat	2024-01	23.7	466	1.6	130	3.57	0.508	0.0
gemma-1.1-2b-it	2024-03	18.0	184	0.7	80	2.29	0.988	0.0
Qwen2-1.5B-Instruct	2024-06	23.7	197	0.7	125	1.57	1.210	0.3
gemma-2-2b-it	2024-07	28.5	339	1.2	133	2.55	0.835	0.0
Phi-3.5-mini-instruct	2024-08	42.7	885	3.0	249	3.55	0.483	0.5
Qwen2.5-3B-Instruct	2024-09	46.5	752	2.7	259	2.91	0.618	0.0
Llama-3.2-3B-Instruct	2024-09	36.4	580	2.0	194	3.00	0.636	2.3
Phi-4-mini-instruct	2025-02	47.0	564	2.0	155	3.65	0.833	0.8
gemma-3-4b-it	2025-02	46.0	1,232	4.2	339	3.63	0.373	0.8
Qwen3-4B-Instruct-2507	2025-08	72.3	2,425	8.3	650	3.73	0.298	10.5
Ministral-3-3B-Instruct-2512-BF16	2025-10	60.8	1,602	5.4	511	3.14	0.379	11.0
Qwen3.5-4B	2026-02	73.7	3,127	10.7	814	3.84	0.236	15.1
gemma-4-E2B-it	2026-03	61.0	1,306	4.8	574	2.28	0.472	3.5
Small/Edge								
Llama-2-7b-chat-hf	2023-07	23.7	1,280	4.2	207	6.19	0.185	0.0
deepseek-llm-7b-chat	2023-11	24.7	865	2.9	144	6.00	0.286	1.1
Qwen1.5-7B-Chat	2024-01	33.3	1,565	5.2	239	6.54	0.213	0.0
gemma-1.1-7b-it	2024-03	38.4	728	2.5	91	8.01	0.527	0.0
Meta-Llama-3-8B-Instruct	2024-04	43.5	914	3.1	137	6.68	0.477	0.3
Qwen2-7B-Instruct	2024-06	55.6	1,529	5.1	245	6.25	0.364	0.0
gemma-2-9b-it	2024-06	53.0	1,221	4.2	144	8.49	0.434	0.0

Model Run	Released	Accuracy (%)	Energy per Question (J)	Time per Question (s)	Tokens per Question	Energy per Token (J)	Correct Answers per kJ	At the Limit (%)
Llama-3.1-8B-Instruct	2024-07	49.2	1,759	5.9	266	6.61	0.280	4.8
Qwen2.5-7B-Instruct	2024-09	60.8	1,539	5.1	247	6.24	0.395	0.0
Phi-4-multimodal-instruct	2025-02	41.4	674	2.4	186	3.62	0.614	0.8
Qwen3-8B	2025-04	65.3	1,111	3.8	165	6.75	0.588	0.3
gemma-3n-E4B-it	2025-06	50.9	1,820	6.6	426	4.28	0.280	0.4
Ministral-3-8B-Instruct-2512-BF16	2025-10	69.9	3,235	10.8	468	6.91	0.216	6.7
Qwen3.5-9B	2026-02	79.3	4,671	15.7	669	6.98	0.170	10.2
gemma-4-E4B-it	2026-03	70.7	2,824	9.8	646	4.37	0.250	5.9
Standard/Desktop								
Llama-2-13b-chat-hf	2023-07	29.7	2,491	8.1	211	11.83	0.118	0.4
Qwen1.5-14B-Chat	2024-01	40.9	1,942	6.4	159	12.19	0.210	0.0
Qwen1.5-MoE-A2.7B-Chat	2024-03	26.6	444	1.7	188	2.36	0.600	0.3
Phi-3-medium-4k-instruct	2024-05	54.6	2,374	7.8	195	12.19	0.230	0.3
DeepSeek-V2-Lite-Chat	2024-05	33.1	591	2.5	248	2.38	0.560	0.8
Qwen2.5-14B-Instruct	2024-09	67.2	3,422	11.3	281	12.18	0.196	0.3
phi-4	2024-12	64.5	2,588	8.5	210	12.32	0.249	0.5
gemma-3-12b-it	2025-03	65.3	3,234	10.8	310	10.42	0.202	0.0
Qwen3-14B	2025-04	70.7	2,131	7.0	174	12.24	0.332	0.0
Ministral-3-14B-Instruct-2512-BF16	2025-10	72.0	4,630	15.3	419	11.04	0.156	4.8

Table A.1: Results of the instruct runs, by size category and release date.

Model Run	Released	Accuracy (%)	Energy per Question (J)	Time per Question (s)	Tokens per Question	Energy per Token (J)	Correct Answers per kJ	At the Limit (%)
Tiny/Mobile								
DeepSeek-R1-Distill-Qwen-1.5B	2025-01	39.2	3,975	14.3	2,517	1.58	0.099	20.7
Phi-4-mini-reasoning	2025-04	66.7	11,084	38.3	2,980	3.72	0.060	19.4
Qwen3-4B-Thinking-2507	2025-08	73.7	13,491	45.7	3,596	3.75	0.055	16.7
Ministral-3-3B-Reasoning-2512	2025-10	58.6	9,419	31.6	2,988	3.15	0.062	22.6
Qwen3.5-4B	2026-02	72.8	15,791	53.8	4,103	3.85	0.046	29.6
gemma-4-E2B-it	2026-03	66.8	4,204	15.3	1,858	2.26	0.159	1.7
Small/Edge								
Qwen3-8B	2025-04	75.5	17,973	60.0	2,701	6.65	0.042	14.8
DeepSeek-R1-0528-Qwen3-8B	2025-05	69.9	20,855	69.4	3,142	6.64	0.034	19.1
Ministral-3-8B-Reasoning-2512	2025-10	70.7	18,493	61.4	2,671	6.92	0.038	18.3
Qwen3.5-9B	2026-02	73.9	27,291	91.3	3,920	6.96	0.027	29.3
gemma-4-E4B-it	2026-03	75.8	7,414	25.4	1,708	4.34	0.102	1.6
Standard/Desktop								
DeepSeek-R1-Distill-Qwen-14B	2025-01	75.5	25,419	84.0	2,112	12.03	0.030	9.1
Phi-4-reasoning	2025-04	74.2	27,390	90.2	2,257	12.14	0.027	8.3
Qwen3-14B	2025-04	78.5	24,329	80.1	2,031	11.98	0.032	8.9
Ministral-3-14B-Reasoning-2512	2025-10	71.8	39,243	129.5	3,569	11.00	0.018	24.7

Table A.2: Results of the reasoning runs, by size category and release date.

Precision of the Energy Measurement

Figure A.1 shows the GPU energy of every repetition relative to the median of its model run. The 35 model runs whose repetitions produced the same responses are drawn together in the first row: their energy differs by 1.0% at most between repetitions, and by 0.5% in the median model run. In the other 18 model runs, some responses changed between repetitions, and the energy differs by 7.3% in the median model run and by 20.5% at most, while the energy per token stays within 1.3%. This variation therefore comes from the responses, and not from the measurement.

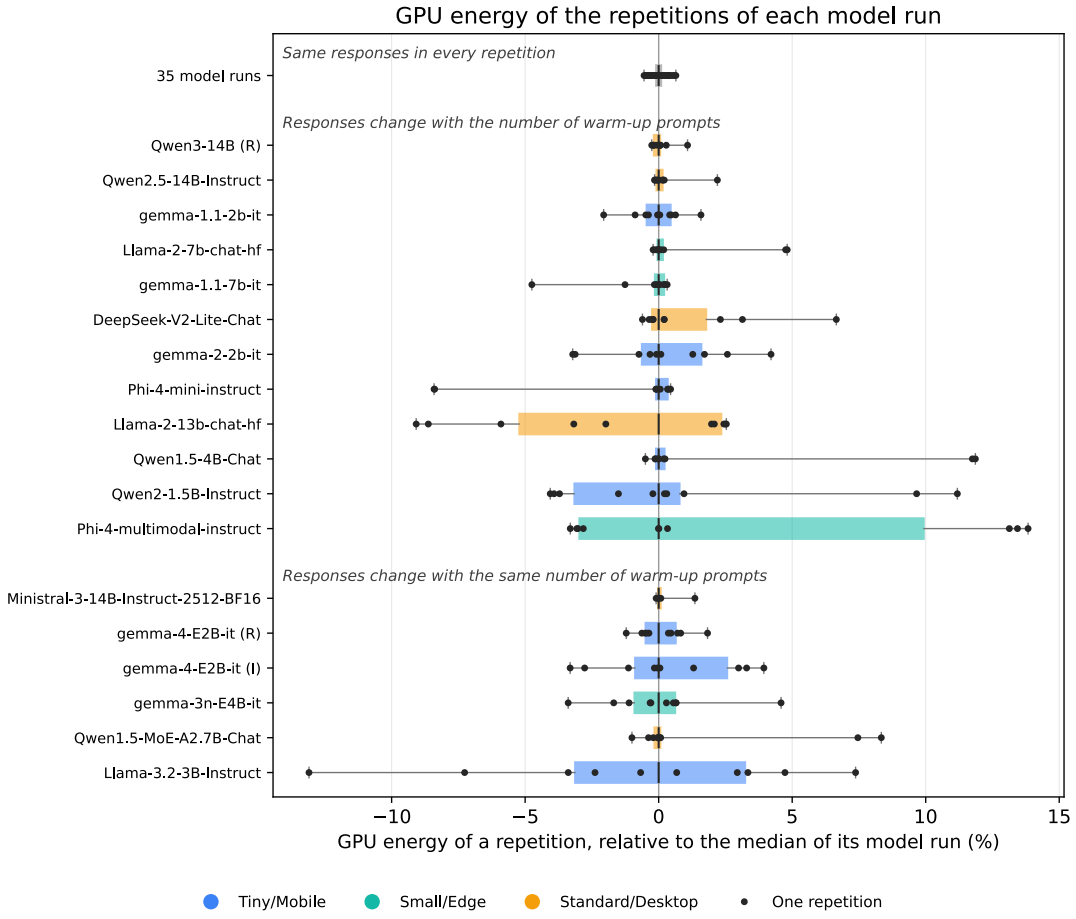


Figure A.1: GPU energy of the repetitions of each model run, relative to its median.

Temperature of the Hardware

Figure A.2 follows the temperature of the GPU and of the CPU, read after each inference (Section 5.4.4), along the 372 questions of a run, as the mean of the runs of each operating mode and size category. Every run starts from the thermal state set before it (Section 4.4.4). The GPU then comes within 1 °C of its final temperature after 0.6 minutes in the median run, and after 4.7 minutes at most, at a level that

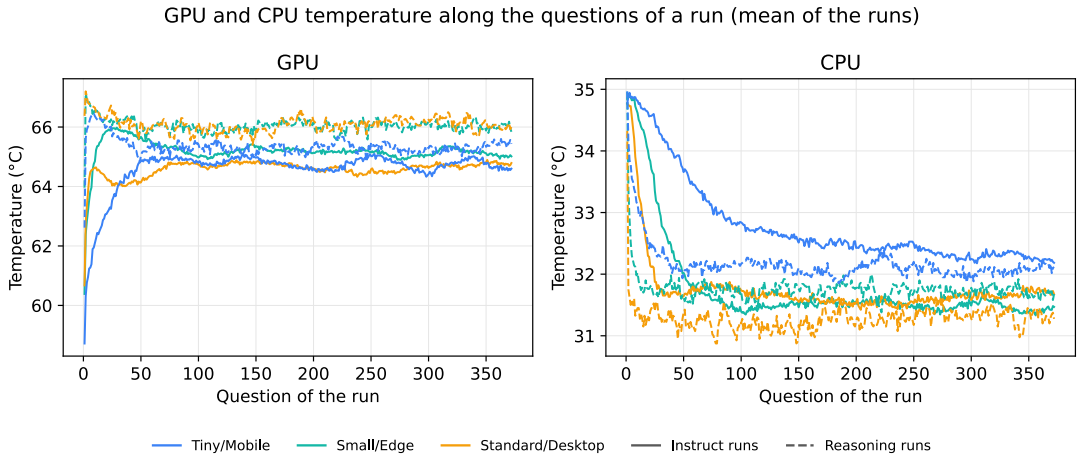


Figure A.2: GPU and CPU temperature along the questions of a run, as the mean of the runs of each operating mode and size category.

Size Category	GPU Temperature (°C)			CPU Temperature (°C)		
	After First Question	Average	Maximum	After First Question	Average	Maximum
Instruct runs						
Tiny/Mobile	59.0	64.7	67.0	35.0	32.7	36.0
Small/Edge	61.0	65.3	67.0	35.0	31.6	35.0
Standard/Desktop	62.0	65.3	68.0	35.0	31.0	35.0
Reasoning runs						
Tiny/Mobile	63.0	65.5	67.5	35.0	32.0	35.0
Small/Edge	64.0	66.2	68.0	35.0	31.7	35.0
Standard/Desktop	66.5	66.2	68.0	34.5	31.3	34.5

Table A.3: Temperature of the GPU and of the CPU during the runs: median of the model runs of each operating mode and size category.

follows the power of the model: the rank correlation between the average GPU temperature of a model run and its GPU power is 0.73. The CPU, which is little used while the model generates, cools from 35 °C to between 31 °C and 33 °C.

Table A.3 gives, for each operating mode and size category, the median over its model runs of the temperature after the first question, of the average temperature over the run and of the maximum.

Figure A.3 shows the average temperature of every run of the main execution, in the order in which the runs were executed, over its 1,189 hours. The average GPU temperature of a run differs from the median of the runs of the same model run by 0.21 °C in the median, and by between −1.2 °C and +1.4 °C, with no association with the order of execution (rank correlation of −0.08, $p = 0.064$). The CPU temperature

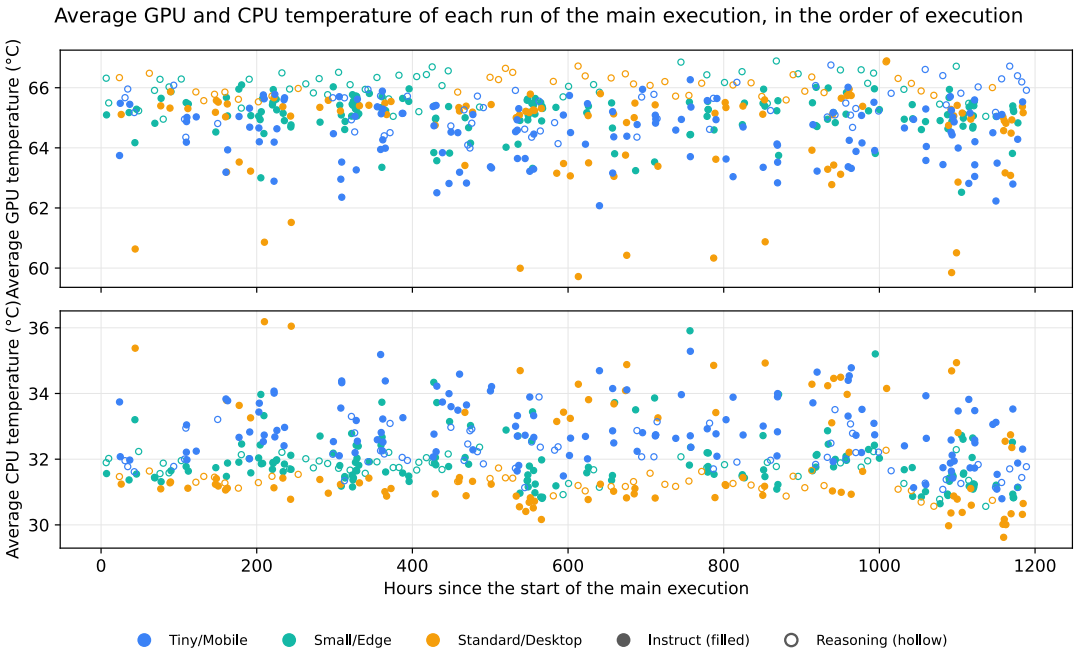


Figure A.3: Average GPU and CPU temperature of each run of the main execution, in the order of execution.

exhibits a weak downward trend (rank correlation of −0.34), of no consequence for the efficiency metrics, which rely on the energy of the GPU.

Evolution Across Successive Releases

Figure A.4, Figure A.5 and Figure A.6 show the distribution of the GPU energy over the 372 questions of each model run, ordered by release date. Each box spans the middle half of the questions, the line inside it is the median, and the whiskers reach the questions that used the least and the most energy. The diamond marks the mean, which is the energy per question used in Chapter 6. In the instruct runs, the mean is always above the median (1.6 times in the median run, and up to 2.6 times), because a small number of long responses use much more energy than a typical one. The whiskers of the most recent instruct runs end where a response reaches the output limit of 2,048 tokens.

Figure A.7 shows the tokens per question and the energy per token of every model run against its release date.

Figure A.8 shows the energy per token of each model run divided by the value that a power law on the active parameters gives for a model of its size. The models that are more than 15% away from that value are labelled.

Figure A.9 repeats the decomposition of Figure 30 between the release years of each size category, using all the instruct runs of each year. The number of runs of each year is given under each bar.

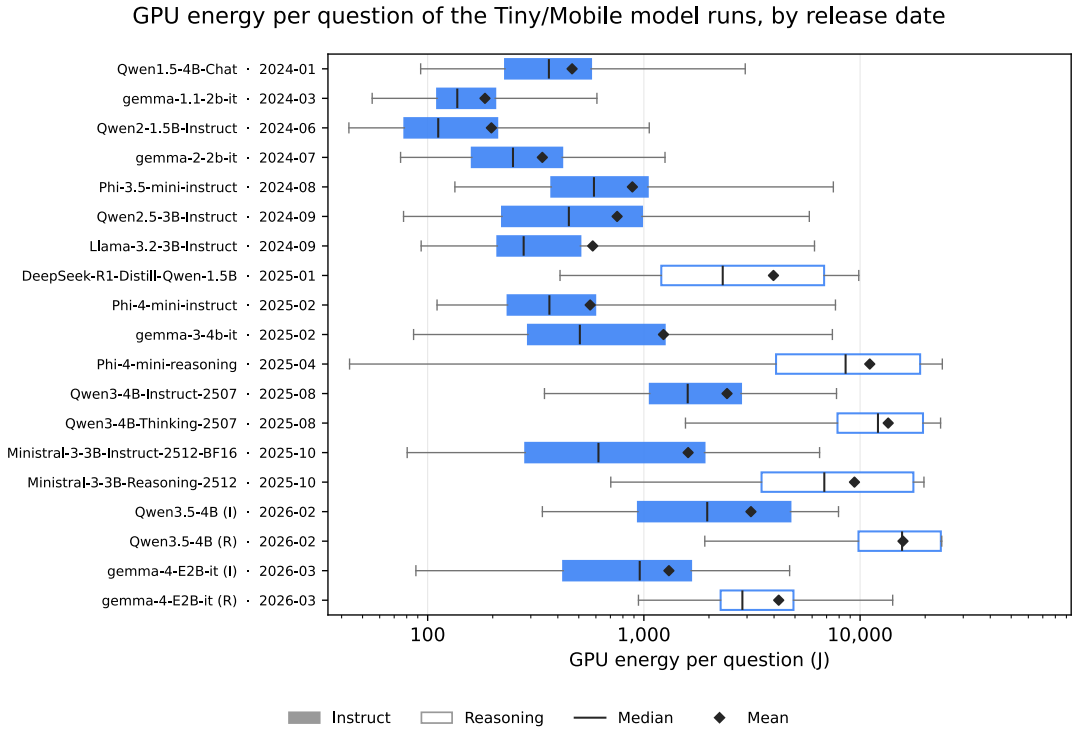


Figure A.4: GPU energy per question of the Tiny/Mobile model runs, by release date.

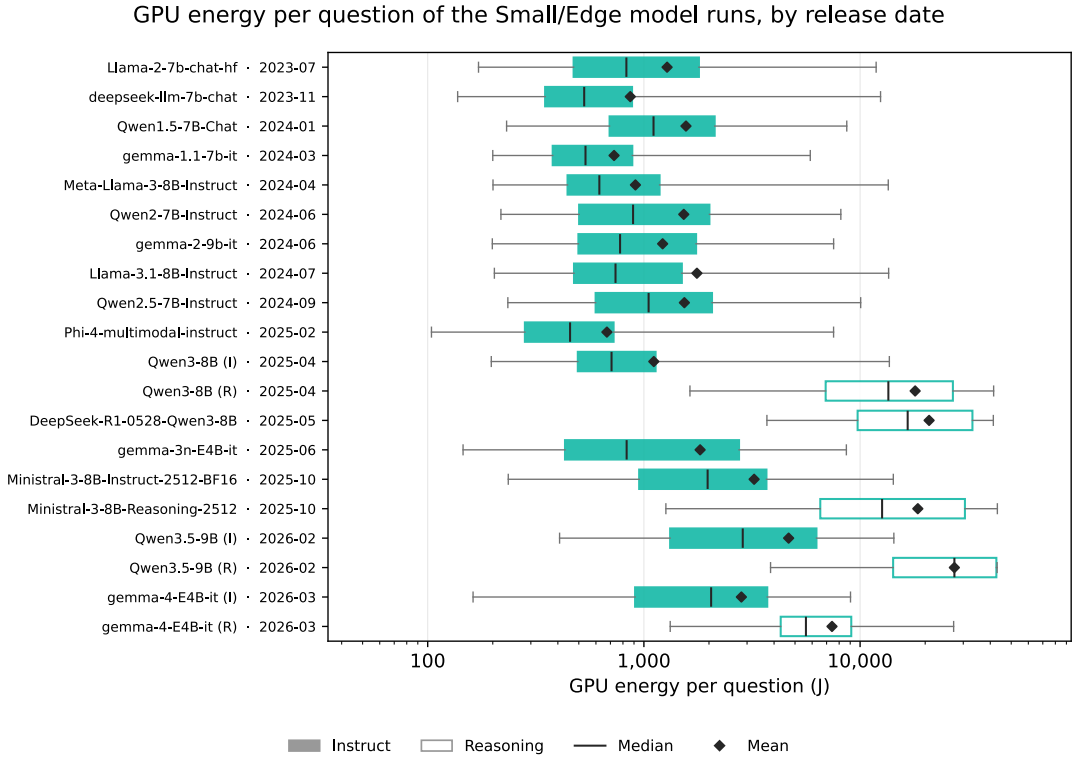


Figure A.5: GPU energy per question of the Small/Edge model runs, by release date.

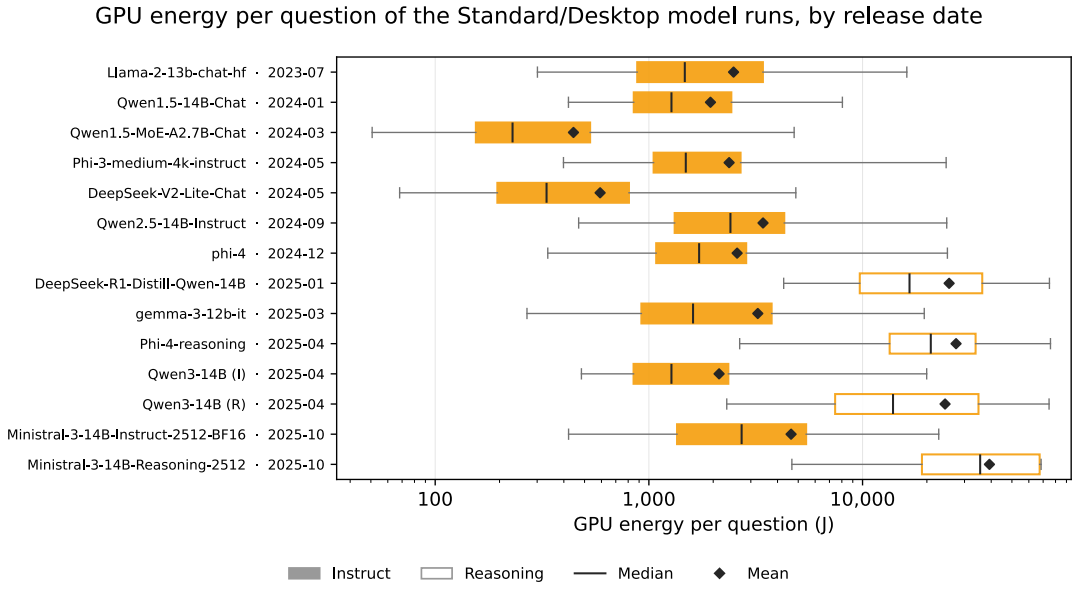


Figure A.6: GPU energy per question of the Standard/Desktop model runs, by release date.

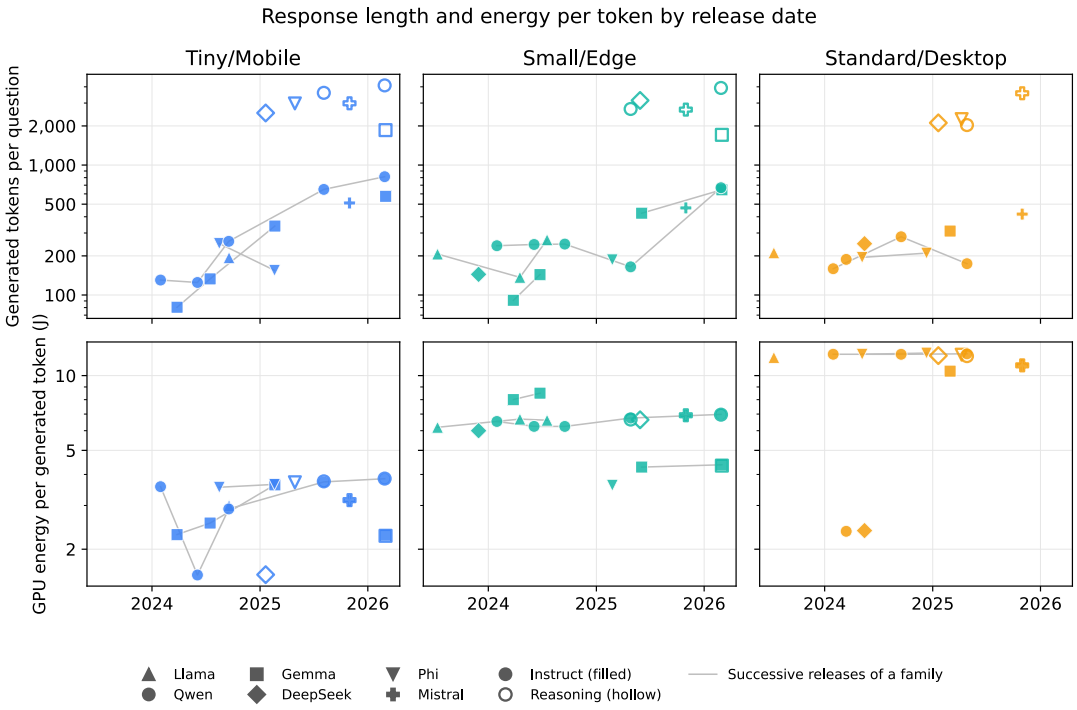


Figure A.7: Tokens per question (top) and energy per token (bottom) of the model runs by release date.

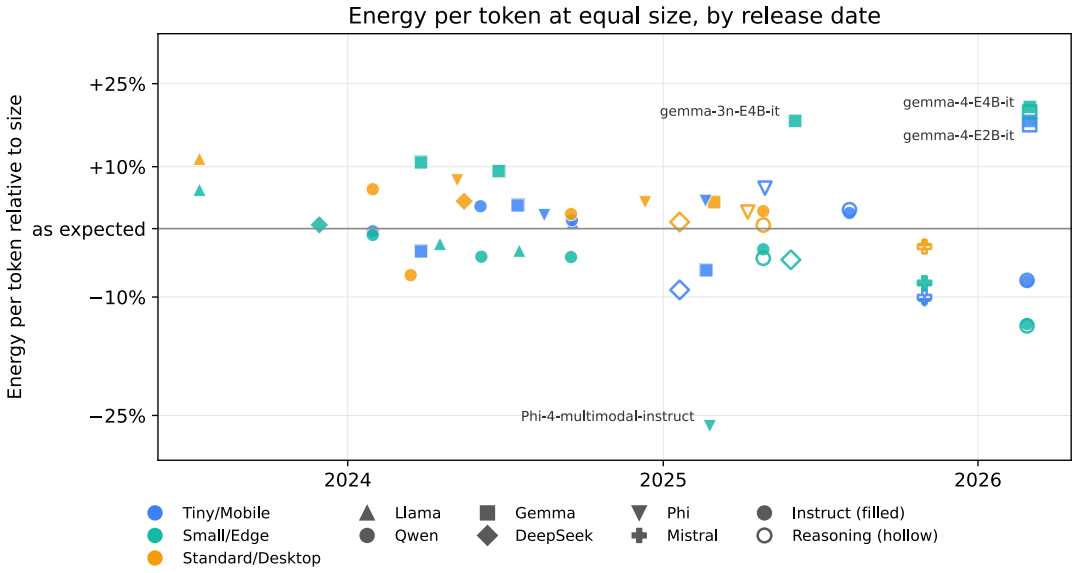


Figure A.8: Energy per token of the model runs relative to the value expected for their size, by release date.

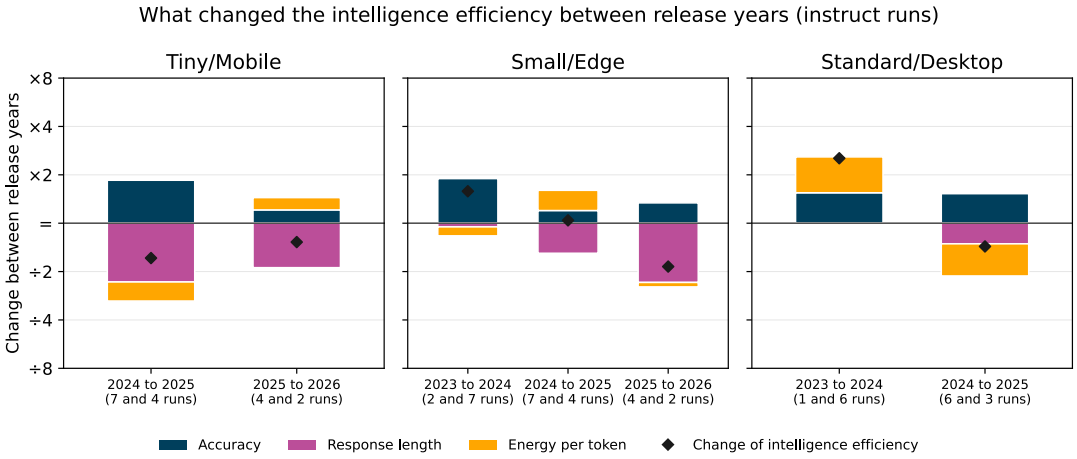


Figure A.9: Contribution of each factor to the change of intelligence efficiency between release years (instruct runs).

The Energy Cost of Accuracy

Table A.4 lists the 18 pairs of successive instruct releases of the same family, size category and architecture, with the change of accuracy and of energy per question from the first model run of each pair to the second. The last column gives the additional energy per question for each additional point of accuracy, in the 12 pairs in which the second model run is more accurate and consumes more energy.

Figure A.10 follows four energy budgets and four accuracy targets over time. On the left, it shows the best accuracy among the model runs released so far that stay within each budget. On the right, it shows the lowest energy per question among those that reach each target. A marker shows each model run that improved the value.

Successive Releases	**Accuracy** (points)	**Energy per Question**, First (J)	**Energy per Question**, Second (J)	**Energy per Question** (ratio)	**Energy per Point** (J)
Tiny/Mobile					
gemma-1.1-2b-it → gemma-2-2b-it	+10.5	184	339	1.84	14.8
gemma-2-2b-it → gemma-3-4b-it	+17.5	339	1,232	3.63	51.1
Phi-3.5-mini-instruct → Phi-4-mini-instruct	+4.3	885	564	0.64	none
Qwen1.5-4B-Chat → Qwen2-1.5B-Instruct	0.0	466	197	0.42	none
Qwen2-1.5B-Instruct → Qwen2.5-3B-Instruct	+22.8	197	752	3.82	24.3
Qwen2.5-3B-Instruct → Qwen3-4B-Instruct-2507	+25.8	752	2,425	3.22	64.8
Qwen3-4B-Instruct-2507 → Qwen3.5-4B	+1.3	2,425	3,127	1.29	523.0
Small/Edge					
gemma-1.1-7b-it → gemma-2-9b-it	+14.5	728	1,221	1.68	34.0
gemma-3n-E4B-it → gemma-4-E4B-it	+19.8	1,820	2,824	1.55	50.8
Llama-2-7b-chat-hf → Meta-Llama-3-8B-Instruct	+19.9	1,280	914	0.71	none
Meta-Llama-3-8B-Instruct → Llama-3.1-8B-Instruct	+5.6	914	1,759	1.93	149.7
Qwen1.5-7B-Chat → Qwen2-7B-Instruct	+22.3	1,565	1,529	0.98	none
Qwen2-7B-Instruct → Qwen2.5-7B-Instruct	+5.1	1,529	1,539	1.01	1.9
Qwen2.5-7B-Instruct → Qwen3-8B	+4.6	1,539	1,111	0.72	none
Qwen3-8B → Qwen3.5-9B	+14.0	1,111	4,671	4.20	254.7
Standard/Desktop					
Phi-3-medium-4k-instruct → phi-4	+9.9	2,374	2,588	1.09	21.5
Qwen1.5-14B-Chat → Qwen2.5-14B-Instruct	+26.3	1,942	3,422	1.76	56.2
Qwen2.5-14B-Instruct → Qwen3-14B	+3.5	3,422	2,131	0.62	none

Table A.4: Change of accuracy and of GPU energy per question between successive instruct releases of the same family.

	Tiny/Mobile		**Small/Edge**		**Standard/Desktop**	
Typical instruct run of each period (medians)						
Released	until 2024	after	until 2024	after	until 2024	after
Number of instruct runs	7	6	9	6	7	3
Accuracy	28.5%	60.9%	43.5%	67.6%	40.9%	70.7%
Energy per question (J)	466	1,454	1,280	2,322	2,374	3,234
Correct answers per kJ	0.636	0.376	0.364	0.265	0.230	0.202
Best accuracy available within an energy budget per question						
Available at the end of	2024	the study	2024	the study	2024	the study
Within 0.5 kJ	28.5%	28.5%	none	none	26.6%	26.6%
Within 1 kJ	46.5%	47.0%	43.5%	43.5%	33.1%	33.1%
Within 2 kJ	46.5%	61.0%	60.8%	65.3%	40.9%	40.9%
Within 5 kJ	46.5%	73.7%	60.8%	79.3%	67.2%	72.0%

Table A.5: Instruct runs of each size category released until the end of 2024 and afterwards: the typical model of each period, and the best accuracy available within each energy budget.

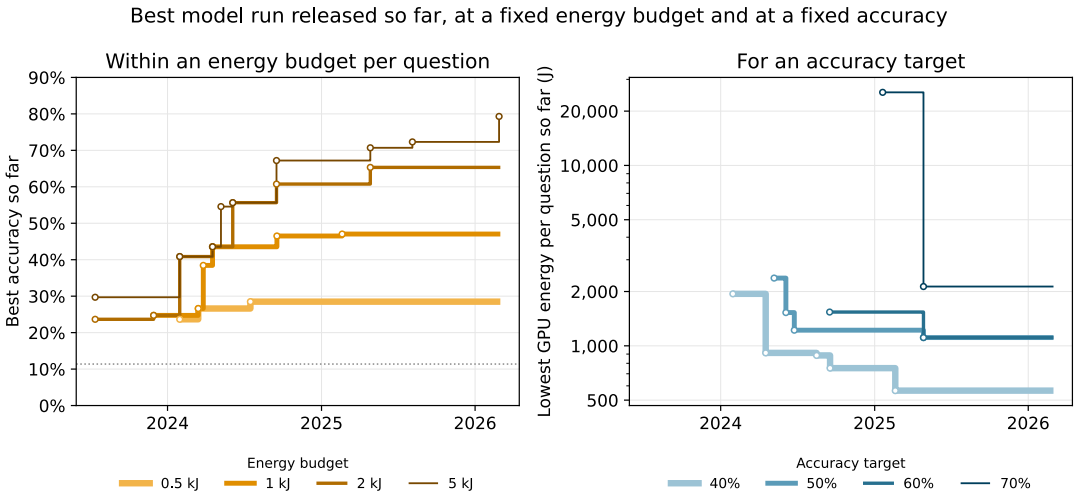


Figure A.10: Best accuracy within each energy budget (left) and lowest energy per question for each accuracy target (right), among the model runs released so far.

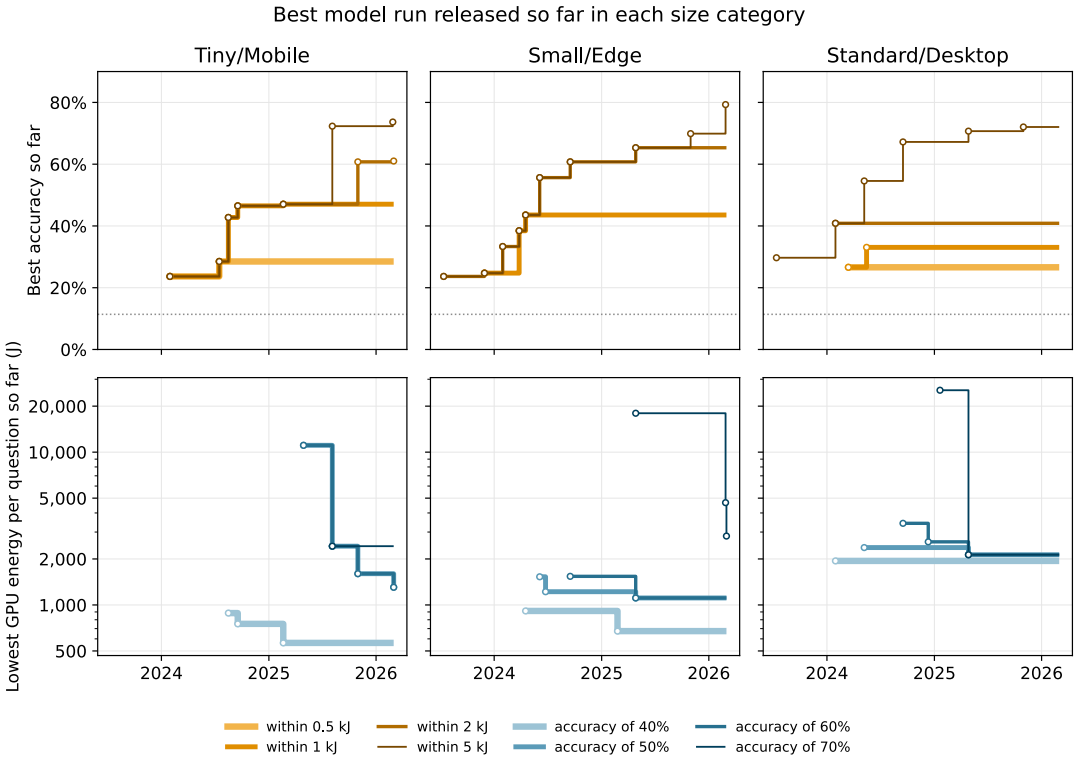


Figure A.11: Best accuracy within each energy budget (top) and lowest energy per question for each accuracy target (bottom), among the model runs of each size category released so far.

Figure A.11 repeats Figure A.10 with the model runs of each size category: the best accuracy within each energy budget in the top row, and the lowest energy per question for each accuracy target in the bottom row.

Table A.5 compares, in each size category, the instruct runs released until the end of 2024 with those released afterwards. The upper part describes the typical model of each period, through the medians.

The lower part gives the best accuracy that the instruct runs of the category offered within each energy budget at the end of 2024, and the best that they offer at the end of the study. In the three categories, the later models are much more accurate and use more energy, and their median intelligence efficiency is not higher in any of them, while the best accuracy available within 5 kJ per question rose in all three.

The Influence of Design Choices

Figure A.12 compares the energy per token of each model with the value predicted from its active parameters by a power law fitted to the other 46 models. The models whose prediction is more than 15% away from the measurement are labelled. The largest error, of 36%, is that of Phi-4-multimodal. This model has 5.6 billion parameters, but its language model is Phi-4-mini, to which encoders and adapters for images and speech were added [227]. These components are not used to answer the questions of the benchmark, and its energy per token is that of Phi-4-mini (3.62 J and 3.65 J).

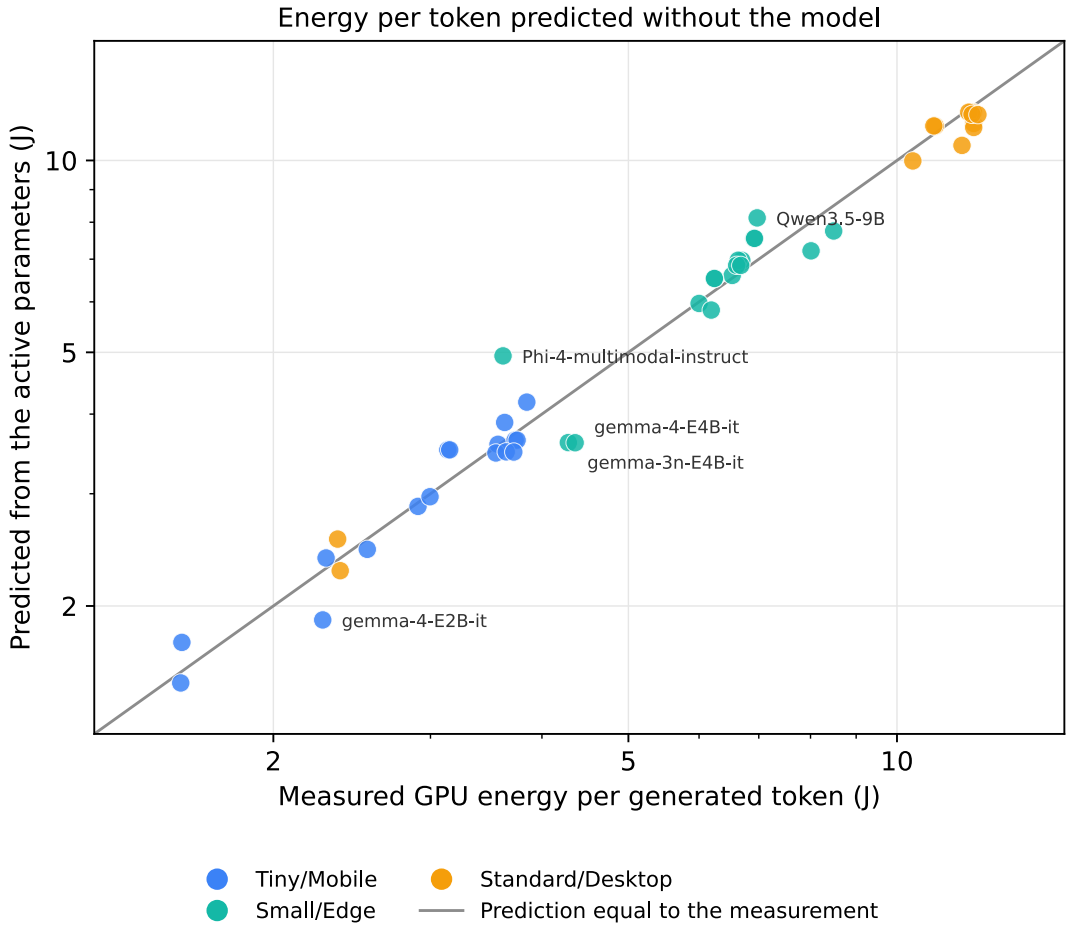


Figure A.12: Energy per token of each model, measured and predicted from its active parameters by a power law fitted without that model.

	Ministral 3 3B	Ministral 3 8B	Ministral 3 14B
Energy per token, 8-bit (J)	1.99	4.07	6.34
Energy per token, 16-bit (J)	3.14	6.91	11.04
Energy per token, 16-bit ÷ 8-bit	1.58	1.70	1.74
Tokens per question, 8-bit	587	509	418
Tokens per question, 16-bit	511	468	419
Accuracy, 8-bit	62.9%	69.4%	74.2%
Accuracy, 16-bit	60.8%	69.9%	72.0%
Difference of accuracy (p-value)	0.298	0.890	0.229

Table A.6: Instruct runs of the Ministral 3 models with 8-bit and with 16-bit weights.

Pair	**Accuracy** (%)		**As Recorded**		**Strict**	**Finished Questions**			**Energy per Question** (ratio)
	Instruct	Reasoning	Points	p	Points	Number	Points	p	
Same weights									
Qwen3-14B	70.7	78.5	+7.8	0.001	+4.6	328	+7.2	0.001	11.4
Qwen3-8B	65.3	75.5	+10.2	< 0.001	+4.8	317	+12.9	< 0.001	16.2
Qwen3.5-4B	73.7	72.8	−0.8	0.644	−5.9	262	+5.3	0.007	5.0
Qwen3.5-9B	79.3	73.9	−5.4	0.007	−11.0	263	+3.0	0.039	5.8
gemma-4-E2B-it	61.0	66.8	+5.8	0.015	+5.9	347	+5.0	0.126	3.2
gemma-4-E4B-it	70.7	75.8	+5.1	0.023	+7.3	366	+6.3	0.004	2.6
Separate variants									
Phi-4-mini	47.0	66.7	+19.6	< 0.001	+14.2	300	+23.4	< 0.001	19.6
Phi-4	64.5	74.2	+9.7	0.001	+7.3	341	+12.3	< 0.001	10.6
Qwen3-4B-2507	72.3	73.7	+1.3	0.519	+0.3	310	+5.8	0.004	5.6
Ministral-3-3B	60.8	58.6	−2.2	0.555	−4.8	288	+2.1	0.545	5.9
Ministral-3-8B	69.9	70.7	+0.8	0.832	−4.8	304	+3.0	0.321	5.7
Ministral-3-14B	72.0	71.8	−0.3	1.000	−9.1	280	+6.2	0.027	8.5

Table A.7: Reasoning run against the instruct run of each pair: difference of accuracy in the three views and ratio of the GPU energy per question.

Table A.6 compares the runs of the three Ministral 3 instruct models with 8-bit and with 16-bit weights. The p-value is that of the exact McNemar test on the accuracy of the two runs.

Table A.7 lists the values of Figure 36 for the 12 pairs of an instruct run and a reasoning run. The differences are those of the reasoning run minus the instruct run, and the p-values are those of the exact McNemar test. The finished questions of a pair are those whose reasoning response did not reach the output limit in any repetition.

Figure A.13 divides the difference of intelligence efficiency between the reasoning run and the instruct run of each pair into the contributions of the accuracy, of the response length and of the energy per token.

Figure A.14 gives the rank correlation (Spearman) between the characteristics of the 53 model runs and their results.

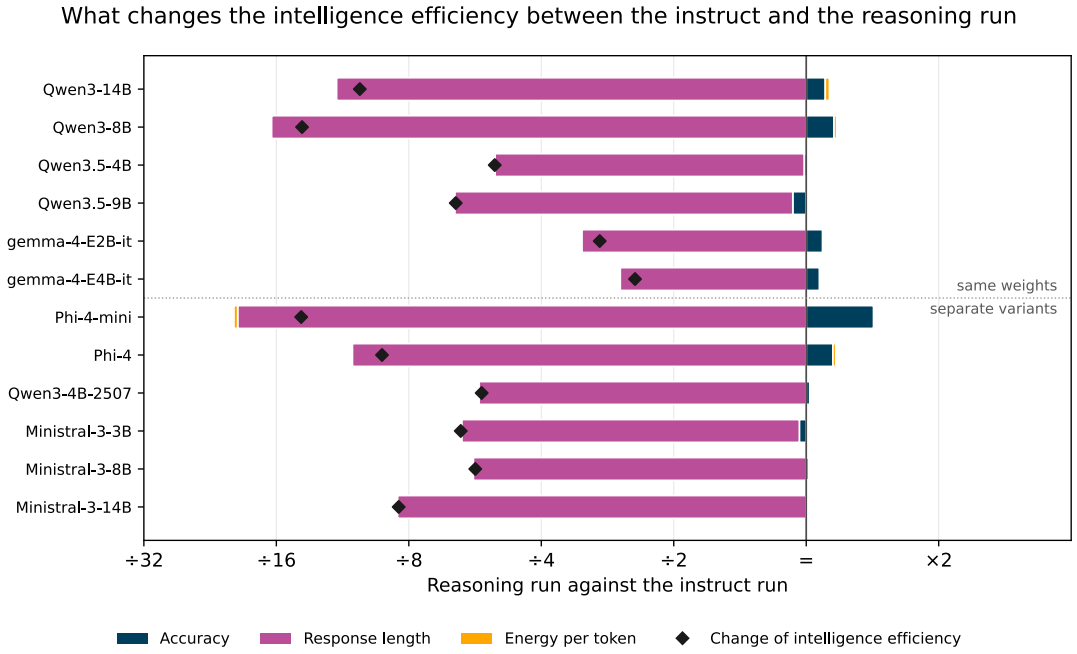


Figure A.13: Contribution of each factor to the difference of intelligence efficiency between the reasoning run and the instruct run of each pair.

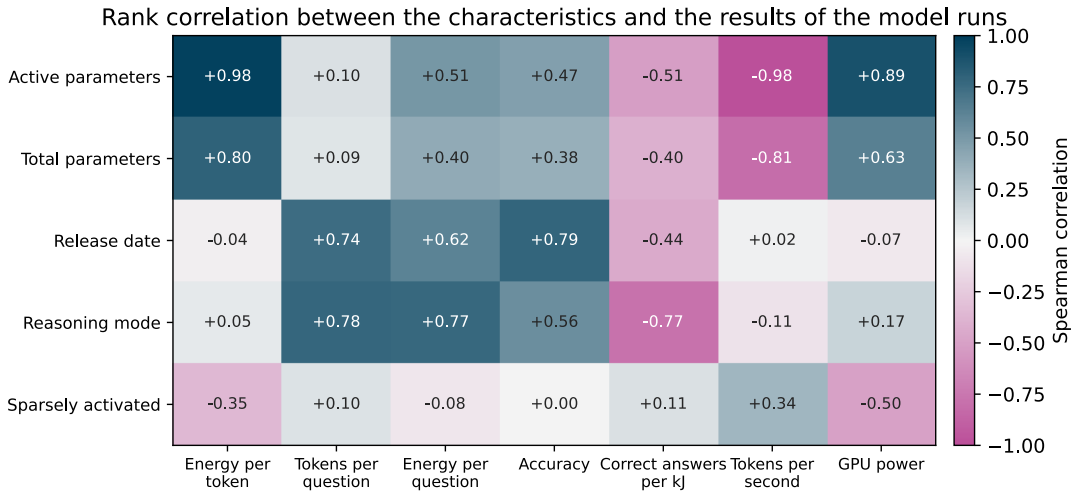


Figure A.14: Rank correlation between the characteristics and the results of the model runs.

Table A.8 lists the responses that reached the output limit in the 15 reasoning runs, and in the six instruct runs in which they are more than 5% of the responses. The strict accuracy counts every response that reached the limit as wrong. As in the other tables, each value is the median of the repetitions of the model run.

Model Run	Accuracy (%)	Strict Accuracy (%)	At the Limit (%)	Cut Off While Reasoning (%)	GPU Energy of the Responses at the Limit (%)
Reasoning runs					
Qwen3.5-4B	72.8	63.7	29.6	22.8	44.4
Qwen3.5-9B	73.9	65.1	29.3	22.8	46.0
Ministral-3-14B-Reasoning-2512	71.8	62.1	24.7	21.2	42.8
Ministral-3-3B-Reasoning-2512	58.6	53.0	22.6	22.6	47.0
DeepSeek-R1-Distill-Qwen-1.5B	39.2	33.1	20.7	19.1	50.9
Phi-4-mini-reasoning	66.7	61.3	19.4	18.0	41.3
DeepSeek-R1-0528-Qwen3-8B	69.9	65.6	19.1	18.0	37.6
Ministral-3-8B-Reasoning-2512	70.7	63.4	18.3	18.0	42.4
Qwen3-4B-Thinking-2507	73.7	69.1	16.7	13.4	28.9
Qwen3-8B	75.5	70.2	14.8	12.9	34.0
DeepSeek-R1-Distill-Qwen-14B	75.5	73.9	9.1	8.3	26.8
Qwen3-14B	78.5	75.3	8.9	8.1	27.0
Phi-4-reasoning	74.2	71.8	8.3	7.3	22.9
gemma-4-E2B-it	66.8	66.4	1.7	0.5	5.8
gemma-4-E4B-it	75.8	75.8	1.6	0.8	5.9
Instruct runs					
Qwen3.5-4B	73.7	69.6	15.1	0.0	37.9
Ministral-3-3B-Instruct-2512-BF16	60.8	57.8	11.0	0.0	44.3
Qwen3-4B-Instruct-2507	72.3	68.8	10.5	0.0	33.2
Qwen3.5-9B	79.3	76.1	10.2	0.0	31.2
Ministral-3-8B-Instruct-2512-BF16	69.9	68.3	6.7	0.0	29.4
gemma-4-E4B-it	70.7	68.5	5.9	0.0	18.8

Table A.8: Responses at the output limit in the reasoning runs, and in the instruct runs with more than 5% of them.

Trajectories of the Model Families

Table A.9 gives the releases and the model runs of each family in the final selection.

Table A.10 lists the values of Figure 38. The last four columns are factors: a value above 1 improves the intelligence efficiency, and the first three multiply to give the last one.

Figure A.15 compares the families that released an instruct model in the same year and size category, using the last release of each family in that year. The vertical line of each year spans the families that are compared. Between the most and the least accurate of them, the difference goes from 1 to 34 points, and the ratio of their energies per question from 1.4 to 5.8, with no consistent narrowing or widening over the years.

Figure A.16 shows the difference of each family from the average of the families, at equal release date and size category, with 95% confidence intervals. Of the 24 estimated effects, only that of Qwen in accuracy has an interval that excludes zero (5.4 points above the average), and one such case among 24 intervals is what chance alone would be expected to produce. In the same models, the release date

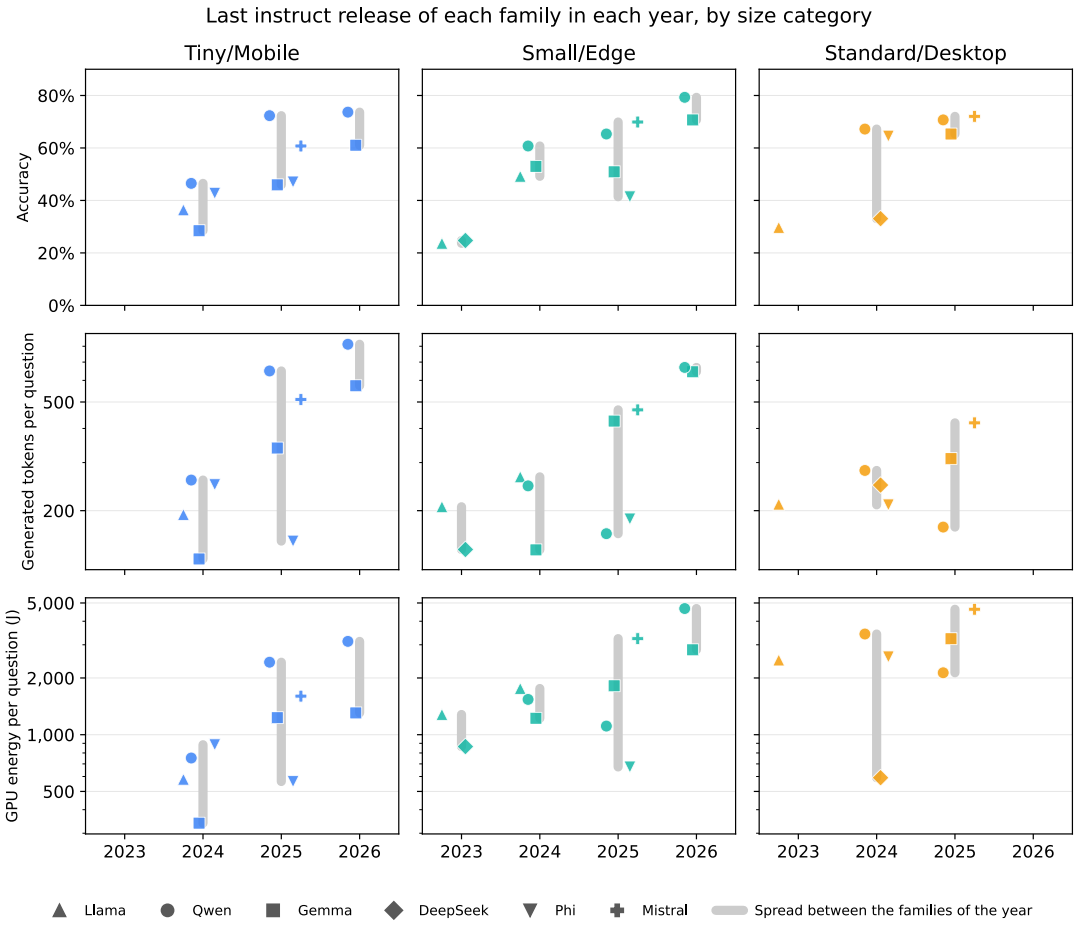


Figure A.15: Last instruct release of each family in each year, by size category: accuracy, tokens per question and GPU energy per question.

Family	Releases	First	Last	Instruct Runs			Reasoning Runs
				Tiny/ Mobile	Small/ Edge	Standard/ Desktop	
Llama	4	2023-07	2024-09	1	3	1	0
Qwen	5	2024-01	2026-02	5	5	4	5
Gemma	5	2024-03	2026-03	4	4	1	2
DeepSeek	3	2023-11	2025-05	0	1	1	3
Phi	2	2024-05	2025-04	2	1	2	2
Mistral	1	2025-10	2025-10	1	1	1	3

Table A.9: Releases and model runs of each family in the final selection.

has a clear effect: 21 points of accuracy per year, and twice the tokens and twice the energy per question per year.

First and Last Release	Releases	Months	Accuracy (points)	Energy per Question (ratio)	Accuracy	Response Length	Energy per Token	Intelligence Efficiency
Tiny/Mobile								
Qwen1.5-4B-Chat → Qwen3.5-4B	5	25	+50.0	6.72	3.07	0.16	0.93	0.47
gemma-1.1-2b-it → gemma-4-E2B-it	4	23	+43.0	7.09	3.38	0.14	1.00	0.47
Phi-3.5-mini-instruct → Phi-4-mini-instruct	2	6	+4.3	0.64	1.10	1.64	0.97	1.75
Small/Edge								
Llama-2-7b-chat-hf → Llama-3.1-8B-Instruct	3	12	+25.5	1.37	2.00	0.78	0.94	1.47
Qwen1.5-7B-Chat → Qwen3.5-9B	5	25	+46.0	2.98	2.38	0.36	0.94	0.80
gemma-1.1-7b-it → gemma-4-E4B-it	4	23	+32.3	3.88	1.87	0.14	1.83	0.48
Standard/Desktop								
Qwen1.5-14B-Chat → Qwen3-14B	3	15	+29.8	1.10	1.73	0.91	1.00	1.58
Phi-3-medium-4k-instruct → phi-4	2	7	+9.9	1.09	1.18	0.93	0.99	1.08

Table A.10: Change between the first and the last instruct release of each trajectory, and contribution of each factor to the change of intelligence efficiency.

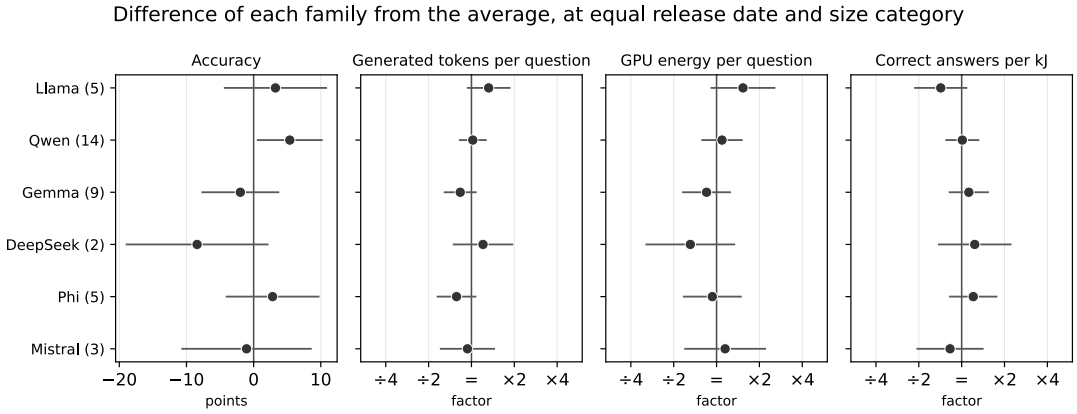


Figure A.16: Difference of each family from the average of the families, at equal release date and size category, with 95% confidence intervals (number of instruct runs in parentheses).

Robustness of the Results

Table A.11 repeats the main results with three changes: the strict accuracy, in which every response that reached the output limit counts as wrong, the GPU energy above the consumption of the idle GPU (Section 4.1.3), and the accuracy above the chance level, in which the accuracy that random answers would obtain is subtracted from the accuracy of every model run. The first two rows compare the ranking of the 53 model runs by intelligence efficiency with the ranking as reported. The last three give the yearly change of the intelligence efficiency of the instruct runs of each size category, as a factor. The Pareto front is not affected by the chance level, since it depends only on the order of the accuracies.

With the accuracy above the chance level, the intelligence efficiency of the instruct runs is not significantly associated with the release date in any size category: the rank correlation is −0.31 in Tiny/Mobile

	As Reported	Strict Accuracy	Energy Above Idle	Both	Accuracy Above Chance
Rank correlation with the ranking as reported	1.000	0.999	1.000	0.999	0.982
Largest change of position in the ranking	0	2	1	2	9
Model runs on the Pareto front	9	8	9	8	9
Of which, also on the front as reported	9	7	9	7	9
Reasoning runs on the Pareto front	0	0	0	0	0
Yearly change, Tiny/Mobile	×0.622	×0.612	×0.621	×0.611	×0.815
Yearly change, Small/Edge	×0.931	×0.923	×0.933	×0.925	×1.076
Yearly change, Standard/Desktop	×0.962	×0.963	×0.958	×0.960	×1.133

Table A.11: Main results with the strict accuracy, with the GPU energy above the idle consumption and with the accuracy above the chance level.

($p = 0.306$), 0.15 in Small/Edge ($p = 0.594$) and 0.13 in Standard/Desktop ($p = 0.726$), against −0.74 ($p = 0.004$), −0.08 and −0.02 as reported. Its highest value in each category then belongs to Phi-4-mini, Qwen3 8B and DeepSeek-V2-Lite, in place of Qwen2 1.5B, Phi-4-multimodal and Qwen1.5-MoE-A2.7B. The adjusted ratio nevertheless remains closely associated with the energy per question, with a rank correlation of −0.91, against −0.96 as reported.

Subjects of the Benchmark

Figure A.17 gives, for each of the 14 subjects of the benchmark, the median accuracy and the median GPU energy per question, relative to that of the whole run, among the 38 instruct runs and among the 15 reasoning runs, two groups that do not contain the same models. Figure A.18 shows this relative energy for every model run. The model runs agree on the order of the subjects by energy much more than by accuracy: Kendall's coefficient of concordance is 0.87 for the energy and 0.55 for the accuracy among the instruct runs, and 0.80 and 0.56 among the reasoning runs.

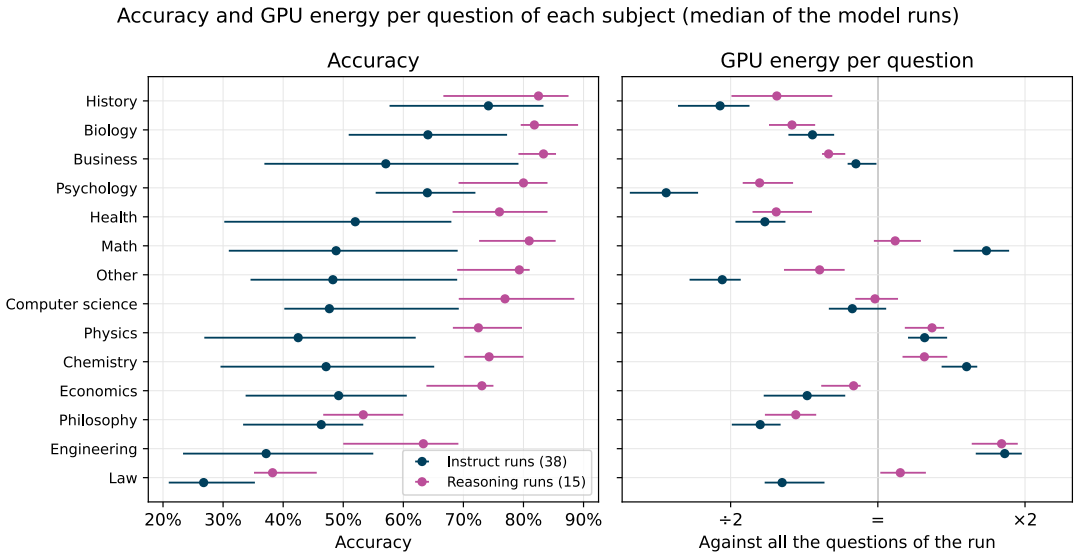


Figure A.17: Accuracy and GPU energy per question of each subject, relative to that of all the questions, as the median of the instruct runs and of the reasoning runs.

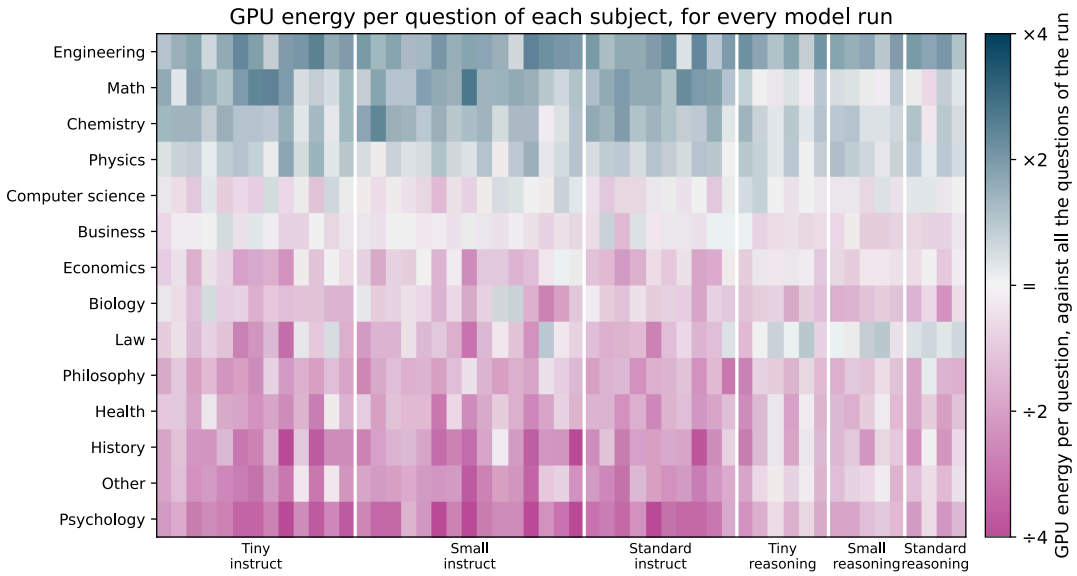


Figure A.18: GPU energy per question of each subject relative to that of all the questions, for every model run.

Appendix B

Implementation Reference

This appendix complements Chapter 5 with reference material: the listings that show a data format, a configuration or a call to a library, and the complete lists that the chapter only summarizes.

Model Selection Notebook

Table B.1 lists the models excluded through the explicit list of the initial filtering, with the reason for each exclusion. Listing B.1 shows how the total parameter count of a model is obtained from its configuration, and Listing B.2 how the representative of each combination is selected.

Model	Reason for Exclusion
`Phi-3-small-8k-instruct` `Phi-3-small-128k-instruct`	Architecture no longer supported by vLLM after version 0.9.2.
`Phi-4-mini-flash-reasoning`	Architecture no longer supported by vLLM after version 0.10.2.
`phi-1` `phi-1_5`	Fail to load in vLLM 0.19.1, the version used in this study.
`gemma-4-12B` `gemma-4-12B-it`	Architecture requires a newer vLLM version (0.23.0 or later) and CUDA 13, which was not available on the server.
`Llama-2-7b-hf` `Llama-2-13b-hf`	Base models whose `-hf` suffix would cause them to be mistaken for tuned variants in the prioritization step (Section 4.2.4).

`Ministral-3-3B-Instruct-2512`
`Ministral-3-8B-Instruct-2512`
`Ministral-3-14B-Instruct-2512`
	Distributed with 8-bit (FP8) weights, which their names do not indicate. Their 16-bit versions, published in separate repositories, are used instead.

Table B.1: Models excluded through the explicit list, and the reason for each exclusion.

```
def get_total_parameters_from_config(model_id):
    try:
        config = AutoConfig.from_pretrained(model_id)
        with accelerate.init_empty_weights():
            model = AutoModelForCausalLM.from_config(config)

        return format_to_billions(model.num_parameters())
    except Exception:
        return "N/A"
```

Listing B.1: Retrieval of a model's total parameter count from its model configuration.

```
filtered_df = (
  df.sort_values(
      by=["category", "collection", "total_parameters", "release_date"],
      ascending=[True, True, False, False],
  )
  .drop_duplicates(
      subset=["category", "collection", "is_moe", "is_thinking"],
      keep="first",
  )
  .copy()
)
```

Listing B.2: Selection of a representative model for each combination of size category, collection, architectural paradigm and operating mode.

Experiment Orchestration

Table B.2 lists the corrections applied to the tokenizer configuration of the four models that could not be loaded, and Listing B.3 shows an excerpt of the file that holds the settings of every model.

Model	Declared Class	Patched Class

`DeepSeek-R1-Distill-Qwen-1.5B`
`DeepSeek-R1-Distill-Qwen-14B`
`DeepSeek-R1-0528-Qwen3-8B`
	`LlamaTokenizerFast`	`Qwen2TokenizerFast`

`deepseek-llm-7b-chat`	`LlamaTokenizerFast`	`PreTrainedTokenizerFast`

Table B.2: Patches applied to the `tokenizer_class` entry of models that could not be loaded.

```
{
  "default": { "max_new_tokens": 2048, "gpu_util": 0.9 },
  "models": {
    "Qwen/Qwen3-8B (Thinking Mode)": {
      "temperature": 0.6, "top_p": 0.95, "top_k": 20, ...
    },
    "Qwen/Qwen3-8B (Instruct Mode)": {
      "temperature": 0.7, "top_p": 0.8, "top_k": 20, ...
    },
    ...
  }
}
```

Listing B.3: Excerpt of the `model_configs.json` file, with shared defaults and per-model, per-mode sampling parameters.

Listing B.4 shows the arguments with which the evaluation script is called, and Listing B.5 how it is executed as a separate process and how its duration is measured.

```
python evaluate_from_local.py \
  --model <model> \
  --save_dir <run directory> \
  --global_record_file <global record file> \
  --config_path <model configuration file> \
  --max_model_length <maximum model length> \
  --max_thinking_tokens <maximum thinking tokens> \
  --enable_thinking <whether thinking is enabled in dual-mode models> \
  --ntrain <number of few-shot examples> \
  --sample
```

Listing B.4: Arguments passed to the model evaluation script.

```
def start_run(self, context: RunnerContext) -> None:
  ...
  log = open(context.run_dir / "experiment.log", "w")

  start = time.time()
  subprocess.run(command, shell=True, stdout=log, stderr=log, env=self.env)
  end = time.time()

  runtime = end - start
  with open(context.run_dir / "runtime.txt", "w") as f:
      f.write(str(runtime))
```

Listing B.5: Executing the model evaluation script as a separate process and measuring its runtime.

Evaluation Script

Listing B.6 shows the creation of the inference engine, Listing B.7 the system prompt that the documentation of the Ministral 3 reasoning models prescribes, and Listing B.8 the structure of the record that is stored for each question of a run (Section 5.4.6).

```
llm = LLM(
    model=str(model_path),
    gpu_memory_utilization=gpu_memory_utilization,
    tensor_parallel_size=1,
    max_model_len=max_model_length,
    trust_remote_code=True,
    max_num_seqs=1,
    enforce_eager=False,
    enable_prefix_caching=False,
    disable_log_stats=False,
)
```

Listing B.6: Initialization of the vLLM inference engine.

```
# HOW YOU SHOULD THINK AND ANSWER

First draft your thinking process (inner monologue) until you arrive at a response.
Format your response using Markdown, and use LaTeX for any mathematical equations. Write
both your thoughts and the response in the same language as the input.

Your thinking process must follow the template below:
[THINK]Your thoughts or/and draft, like working through an exercise on scratch paper. Be
as casual and as long as you want until you are confident to generate the response to the
user.[/THINK]
Here, provide a self-contained response.
```

Listing B.7: Custom system prompt required by the Ministral reasoning models.

```
{
  "question_id": 0, "category": "...", "question": "...",
  "options": ["...", ...], "answer": "A", "answer_index": 0,
  "model_outputs": "...", "pred": "A",
  "gpu_temperature_celsius": 0.0, "cpu_temperature_celsius": 0.0,
  "prefill_time_seconds": 0.0, "decode_time_seconds": 0.0,
  "inference_time_seconds": 0.0,
  "inference_gpu_joules": 0.0,
  "inference_cpu_joules": 0.0, "inference_dram_joules": 0.0,
  "total_inference_joules": 0.0,
  "prompt_tokens": 0, "generated_tokens": 0,
  "total_tokens": 0, "thinking_tokens": 0
}
```

Listing B.8: Structure of a single per-prompt record, with the values omitted.

Appendix C

Disclosure of the Use of Generative Artificial Intelligence

In accordance with the regulations of the University of Minho on academic integrity and the use of artificial intelligence [228], which require any use of these tools to be noted in the work, and with its recommendations on how to declare that use [229], I disclose below the generative AI tools that I used during the preparation of this dissertation.

Google's Gemini Notebook[16] was used during the literature review (Chapter 2) as an assistive tool to help extract and synthesize relevant content from the full texts of the screened articles. Anthropic's Claude Opus 5.5[17] was used mainly as an assistive writing tool, to draft text and to correct spelling, grammar and typography. It also helped implement the Python notebooks that perform the analysis of the results (Chapter 6). GitHub Copilot[18] provided code suggestions during the implementation of the evaluation script (Section 5.4).

No AI system is credited as an author. I reviewed, verified and revised all the text, code and analyses produced with the support of these tools, and I assume full responsibility for the content, originality and integrity of this dissertation, as stated in the Declaration on the Use of Artificial Intelligence (AI) Tools at the beginning of this document.

16 https://notebook.google

17 https://claude.com

18 https://github.com/features/copilot