

Large Language Model Inference Energy Efficiency Evaluation

Benchmark 1.5

Building a “Measure-Evaluate-Apply-Optimize” closed-loop system to advance sustainable AI

PREPARED BY



清华大学
清华四川能源互联网研究院
Sichuan Energy Internet Research Institute
Tsinghua University

This whitepaper was jointly prepared under the leadership of the Information Energy Laboratory, Department of Electrical Engineering and Applied Electronics Technology, Tsinghua University; the Institute of Information Energy and Green Intelligent Computing Technology, Sichuan Energy Internet Research Institute, Tsinghua University; and the Electrochemical Energy Storage Efficient Integration and Control Team of the State Key Laboratory of Operation and Control of New-Type Power Systems. Linux Foundation Research and the SODA Foundation, through its Aitra project, provided support through an open collaborative ecosystem. We express our sincere gratitude to the Linux Foundation, the SODA Foundation and the Aitra project.

WITH SUPPORT FROM



Research



soda foundation

AITRA

Contents

Contents.....	2
Preface.....	4
Chapter 1: Research Background and Evaluation Objectives.....	4
1.1 Rigid Growth of Energy Consumption in Large-Model Inference.....	4
1.2 The Absence of Energy-Efficiency Evaluation and the Selection Dilemma.....	5
1.3 Evaluation Objectives and the Positioning of Benchmark 1.5.....	5
Chapter 2: Energy-Efficiency Evaluation Framework.....	6
2.1 Model Selection.....	6
2.1.1 Selection Principles.....	6
2.1.2 Model Version Continuity and Group Composition.....	7
2.1.3 Representativeness of the Newly Added Models.....	8
2.2 Evaluation Metrics.....	9
2.3 Mapping Energy Efficiency to Electricity Cost.....	11
Chapter 3: Test Method and Experimental Design.....	12
3.1 Test Environment.....	12
3.2 Test Datasets.....	13
3.3 Test Parameters.....	14
3.4 Test Procedure.....	15
Chapter 4: Test Results.....	17
4.1 Energy Consumption Comparison.....	18
4.1.1 Steady-State Energy Consumption Across Scenarios.....	18
4.1.2 Full-Lifecycle Energy Consumption Heatmaps Across Scenarios.....	20
4.2 Structure of Lifecycle Energy Consumption.....	23
4.3 Throughput Comparison.....	25
4.4 Analysis of Representative Models.....	26
4.4.1 Cross-Scenario Deployment Behavior of Gemma 4 31B.....	26
4.4.2 Comparison of Multimodal Models.....	27
4.5 Deployment Behavior Under Different Inference Engines.....	28
4.6 Summary of Test Results.....	30
Chapter 5: Conclusions and Outlook.....	31
5.1 Core Conclusions.....	31
5.2 Future Outlook.....	32
5.3 Limitations.....	33

Appendix A: Evaluated Model Artifacts and Version Identifiers.....	34
Appendix B: Per-Run Configuration and Stage Energy Consumption.....	35

Preface

As large models move into production, the scale of inference-side energy consumption keeps rising. This whitepaper is not concerned with the overall split between training and inference energy, but with how energy is measured at the point of inference deployment. On the same hardware platform, energy per output token can differ several-fold across combinations of model, precision, and inference engine; if the measurement boundary is not unified, these differences cannot be used to compare options.

The Large Model Energy Efficiency Evaluation and Ranking Whitepaper (hereafter Benchmark 1.0) compared efficiency only during the steady running stage, and could not answer cost questions for on-demand startup, elastic scaling, and short-task scenarios. The Large Language Model Inference Energy Efficiency Evaluation Whitepaper (Benchmark 1.5) extends the measurement boundary from the running and steady stages to the full service lifecycle, including startup, the formal run, and resource release. It also expands the set of evaluated models from 8 to 13, adding large native multimodal models, MoE models with low active parameters, hybrid linear-attention models, and lightweight dense enterprise models.

This whitepaper continues the principle of "a unified test point, reproducible and traceable." All conclusions are limited to the model artifacts, inference engine versions, parallel configurations, and workloads listed here, and do not represent each model's efficiency ceiling after independent tuning.

Chapter 1: Research Background and Evaluation Objectives

1.1 Rigid Growth of Energy Consumption in Large-Model Inference

Transformer-based large language models and multimodal models continue to expand, and growth in both parameter scale and call volume is driving up compute demand. Unlike training, where investment is concentrated, inference energy accumulates for as long as a service runs; its total depends on request volume, request length, batch size, and service utilization. As models enter production, inference takes a growing share of data center power load.

The token is the basic unit of measurement for inference, and token consumption maps directly to inference-side energy demand. When token volumes grow rapidly while the energy to process each token can differ several-fold or even tens of times depending on model architecture, precision format, and system adaptation, the industry lacks a unified, token-level efficiency measure and cannot make comparable trade-offs among model capability, call cost, and energy constraints.

At the same time, the energy density of compute infrastructure keeps rising. Per-rack power has increased significantly, technologies such as liquid cooling have moved from optional to required, and data center siting is shifting from proximity to users toward proximity to energy. Energy cost and supply stability have become one of the core constraints on the economics of compute.

1.2 The Absence of Energy-Efficiency Evaluation and the Selection Dilemma

The large-model ecosystem already has fairly mature benchmarks for capability, but still lacks a unified, transparent, and reproducible way to evaluate energy efficiency. This creates three specific difficulties.

First, deployment choices lack a quantitative basis. In private deployments, users can usually only rely on a model's accuracy or output quality on specific tasks, and have no comparable data on the energy cost of completing the same task. The same task can differ several-fold across models, precision artifacts, and inference engines; without measured data, choices drift away from the lowest total cost of ownership.

Second, optimization lacks a common target. The effects of quantization, sparsification, operator fusion, and memory optimization can only be compared across models and configurations with a consistent measure. Without publicly comparable efficiency data, the benefits of different technical paths are hard to judge, and efficiency optimization lacks external competitive pull.

Third, the measurement basis itself is easily distorted. Energy figures depend heavily on the measurement boundary: whether model loading and service initialization are included, whether resource release is included, and whether an idle baseline is subtracted can all change the conclusions significantly. Without a clear boundary, steady-state efficiency data gets used directly as the full cost of a service, overstating real efficiency in short-task scenarios.

1.3 Evaluation Objectives and the Positioning of Benchmark 1.5

This whitepaper focuses on establishing a unified, transparent, and comparable basis for evaluating large-model inference energy efficiency, with the following four objectives.

- (1) Define metrics and measurement boundaries. Give definitions, applicable scenarios, and the relationship between steady-state energy per output token and full-lifecycle energy per output token, and state the start and end of each stage, the sampling method, and the statistical scope.
- (2) Cover the full service lifecycle. While retaining steady-state energy, throughput, and latency metrics, include the fixed costs of model loading, service startup, and service release in the measurement, so that on-demand startup and elastic scaling scenarios have comparable efficiency data.
- (3) Expand model and configuration coverage. Broaden the set of models under a unified test point to cover different parameter scales, architecture types, precision formats, and input modalities, providing like-for-like samples for cross-model comparison.
- (4) Define the scope of conclusions. State that all conclusions apply only to the model artifacts, inference engine versions, parallel configurations, and workloads listed here; disclose reproduction conditions and known limitations item by item; and avoid extrapolating results under a unified baseline into a model's efficiency ceiling.

The whitepaper is structured as follows: Chapter 2 describes the model selection principles and the evaluation metric framework; Chapter 3 describes the test environment, datasets, parameters, and procedure; Chapter 4 presents test results and analysis for three scenarios; Chapter 5 summarizes the core conclusions, future outlook, and limitations.

Chapter 2: Energy-Efficiency Evaluation Framework

2.1 Model Selection

2.1.1 Selection Principles

Benchmark 1.5 targets local inference deployment of large models and selects representative models for performance and efficiency evaluation. The evaluated set covers different parameter scales, architecture types, precision formats, model origins, and input modalities, and requires that model artifacts be obtainable, version-pinnable, and reproducibly deployable.

The specific selection principles are as follows:

- (1) **Deployability.** Model weights must be obtainable locally with a pinned version, and a complete inference service must be buildable with vLLM or SGLang. Models available only through a commercial API, whose weights cannot be obtained locally, or whose version cannot be pinned are not included.
- (2) **Coverage.** The evaluated models cover different parameter scales, architectures, precision formats, regions of origin, and input modalities. Architectures include Dense, MoE, and hybrid linear attention; precisions include BF16, FP8 variants, and MX low-precision formats; regions of origin include the United States, Europe, China, and Japan; input capabilities include text-only and image-text multimodal.
- (3) **Group comparison.** Models are divided by total parameter count into lightweight (no more than 50B), medium (above 50B and below 200B), and large (at least 200B). Each size group retains models with different architectures or precisions where possible, to avoid an overly uniform comparison set.
- (4) **Task applicability.** Text chat, code completion, and image-text question answering are run according to each model's actual capabilities. Unsupported tasks are marked "not applicable" and are not counted as test failures.

Table 2-1 Specifications of the Evaluated Models

Model	Total Parameters	Architecture	Model Capability			
			Text Generation		Code Completion	Image-Text QA
			MMLU-Pro	Arena Text Rating (Style Control)	LiveCodeBench v6	MMMU / MMMU-Pro
DeepSeek-V4-Flash-0731	284B	MoE	—	—	—	Not supported
Qwen3.5-397B-A17B-FP8	397B	MoE	87.8	1442	83.6	85.0 (MMMU) 79.0 (MMMU-Pro)
GLM-5.3-Flash	320B	MoE	—	1474	—	—

Model	Total Parameters	Architecture	Model Capability			
			Text Generation		Code Completion	Image-Text QA
			MMLU-Pro	Arena Text Rating (Style Control)	LiveCodeBench v6	MMMU / MMMU-Pro
MiniMax-M3-MXFP8	≈428B	MoE	—	1440	—	78.1 (MMMU-Pro)
MiMo-V2.5	310B	MoE	—	1434	—	77.9 (MMMU-Pro)
Hy3-FP8	295B	MoE	—	1457	—	Not supported
Llama-4-Maverick-17B-128E-Instruct-FP8	400B	MoE	80.5	—	—	73.4 (MMMU) 59.6 (MMMU-Pro)
gpt-oss-120b	117B	MoE	—	1352	—	Not supported
Gemma-4-31B-it	30.7B	Dense	85.2	—	80.0	76.9 (MMMU-Pro)
Mistral-Medium-3.5-128B	128B	Dense	—	—	—	—
Granite-4.2-30B-FP8	30B	Dense	77.6	1341	75.77	Not supported
Ling-3.0-Flash-FP8	124B	MoE	—	—	—	Not supported
Sarashina2.2-3B-Instruct-v0.1	3B	Dense	—	—	—	Not supported

Notes:

1. "—" means not published or no data on the same basis; "Not supported" means the model has no image input capability.
2. Model capability data were verified on September 28, 2026. MMLU-Pro, LiveCodeBench v6, and MMMU/MMMU-Pro are taken from each model's official model card or technical report; Arena Text Rating is taken from the Style Control view of the LMArena Text Overall leaderboard, published September 25, 2026.
3. For some quantized artifacts, capability data use the official published scores of the corresponding base model and do not represent an independent capability re-test of the quantized artifact. MMMU and MMMU-Pro are different evaluation bases and are labeled separately in the table.

2.1.2 Model Version Continuity and Group Composition

To maintain continuity with Benchmark 1.0 while covering new technical approaches, Benchmark 1.5 divides the evaluated models into continued, updated, and new categories. Continued models keep the original model line; updated models update the model version or deployment artifact

on the original line; new models add representative model categories not yet covered by Benchmark 1.0.

Benchmark 1.0 evaluated 8 models and Benchmark 1.5 evaluates 13: 5 continued, 3 updated, and 5 new.

Table 2-2 Model Evolution from Benchmark 1.0 to Benchmark 1.5

Category	Benchmark 1.0 Model	Benchmark 1.5 Model	Main Change	Precision Format	Weight Source
Continued	Qwen3.5-397B-A17B-FP8	Qwen3.5-397B-A17B-FP8	—	FP8	Qwen
Continued	Llama-4-Maverick-17B-128E-Instruct-FP8	Llama-4-Maverick-17B-128E-Instruct-FP8	—	FP8	RedHatAI
Continued	gpt-oss-120b	gpt-oss-120b	—	MXFP4	openai
Continued	Mistral-Medium-3.5-128B	Mistral-Medium-3.5-128B	—	FP8	mistralai
Continued	Gemma-4-31B-it	Gemma-4-31B-it	—	BF16	google
Updated	DeepSeek-V4-Flash	DeepSeek-V4-Flash-0731	Updated to the 0731 version, with the model revision and artifact pinned for this round	FP8 + FP4	deepseek-ai
Updated	MiniMax-M2.7-REAP-172B-A10B-BF16	MiniMax-M3-MXFP8	Iterated from M2.7 to M3, switched to an MXFP8 artifact, and added image-text input	MXFP8	MiniMaxAI
Updated	MiMo-V2-Flash	MiMo-V2.5	Iterated from V2-Flash to V2.5 and extended into a native multimodal model	FP8	XiaomiMiMo
New	—	GLM-5.3-Flash	Adds a large Chinese native multimodal model and a hybrid attention approach	FP8	zai-org
New	—	Hy3-FP8	Adds a large MoE model sample	FP8	tencent
New	—	Granite-4.2-30B-FP8	Adds a 30B-class dense enterprise model and an FP8 artifact approach	FP8	ibm-granite
New	—	Ling-3.0-Flash-FP8	Adds low-active-parameter MoE and hybrid linear attention	FP8	inclusionAI
New	—	Sarashina2.2-3B-Instruct-v0.1	Adds an official Japanese 3B dense model from Japan	BF16	sbintuitions

Notes:

1. Precision format and weight source both refer to the Benchmark 1.5 artifacts evaluated in this round. The weight source column shows the publishing organization of the Hugging Face repository; full repository IDs, revisions, and commit dates are given in Appendix A.
2. Precision format indicates the main weight format of the artifact at the pinned revision and does not mean that all modules or computations use the same precision. DeepSeek-V4-Flash-0731 uses a mixed FP8 and FP4 format.

2.1.3 Representativeness of the Newly Added Models

The new models fill gaps in Benchmark 1.0's coverage of architecture, scale, precision, region, and language. Their technical positioning is shown in Table 2-3.

Table 2-3 Technical Characteristics and Coverage Value of the New Models

New Model	Main Techniques Used by the Model	Reason for Inclusion
GLM-5.3-Flash	Large MoE structure combining sparse attention and linear attention, with image-text input support	To observe the deployment efficiency of a Chinese native multimodal model and a hybrid attention approach
Hy3-FP8	About 295B total parameters and about 21B active per token; MoE structure, FP8 artifact; the model itself supports expert-parallel deployment	To add measured samples of large MoE models outside the lightweight and other groups. This test uses TP8 throughout without expert parallelism, so it cannot serve as a measured sample under expert-parallel conditions
Granite-4.2-30B-FP8	30B dense Transformer with an FP8 artifact, aimed at enterprise deployment	To provide a scale and precision contrast with Sarashina and Gemma within the lightweight group, and to add a lightweight dense enterprise model sample
Ling-3.0-Flash-FP8	About 124B total parameters and about 5.1B active per token; hybrid linear-attention MoE	To observe low active parameters and linear attention in a medium-scale model
Sarashina2.2-3B-Instruct-v0.1	Official Japanese instruction model with a 3B dense decoder-only architecture; the tested artifact is BF16	To add a Japanese model sample and observe how a small official dense model deploys under the unified eight-GPU baseline

Across all 13 evaluated models, the lightweight models are Gemma, Granite, and Sarashina; the medium models are gpt-oss, Mistral, and Ling; and the large models are DeepSeek, Qwen, GLM, MiniMax, MiMo, Hy3, and Llama. The evaluated models cover Dense, MoE, and hybrid linear-attention architectures, as well as BF16, FP8 variants, and MX low-precision formats; 7 models support image-text multimodal input and 6 support text-only input.

The evaluated models come from the United States, China, Europe, and Japan. The United States and China each have several models, while Europe and Japan are represented by Mistral and Sarashina respectively.

2.2 Evaluation Metrics

Benchmark 1.5 retains the five core metrics of Benchmark 1.0 and adds three lifecycle metrics. The retained metrics are defined as follows:

Output throughput (tok/s): total output tokens generated during the formal request stage divided by the duration of that stage, in tokens per second.

Steady-state energy per output token (J/token): GPU energy within the steady-state window divided by the output tokens in the same window, in joules per token. Numerator and denominator are taken from the same window.

Average GPU power (W): GPU energy in the statistical window divided by the window duration. This whitepaper reports the combined power of all GPUs.

Time to first token (TTFT): time from sending a request to receiving the first output token, in seconds. This section reports P50.

Time per output token (TPOT): generation time after the first token divided by the output tokens in that stage, in seconds per token.

The new lifecycle metrics are as follows:

Startup energy: GPU energy from the start of the inference service to the start of formal requests, reflecting the fixed cost of model loading, service initialization, and warm-up.

Resource release energy: GPU energy from the end of formal requests until the inference process exits, GPU memory is released, and power returns to baseline.

Full-lifecycle energy per output token (J/token): the sum of GPU energy across the startup, formal request, and resource release stages divided by output tokens, used to evaluate the per-output cost of one complete service run.

Steady-state J/token and full-lifecycle J/token correspond to different measurement windows. The former suits analysis of continuous-operation efficiency; the latter suits analysis of cold starts, short tasks, and elastic services. Both are listed as parallel metrics under "energy per output token"; they share the J/token unit but have different measurement boundaries, so they must not be mixed or subtracted from each other.

The denominator of full-lifecycle energy per output token counts only the completion tokens returned by the API for requests that completed successfully during the formal request stage, excluding warm-up and failed requests. Models and inference engines differ in how they return and count thinking content: in some runs thinking content is mixed into ordinary output, while in others it is saved in a separate reasoning field, so the current count cannot uniformly be read as visible output tokens only. Because the tests record visible output and reasoning tokens differently, this whitepaper computes energy per output token from the completion token count returned by the API and does not split the two token types. The thinking setting and token counting basis for each test are given in Appendix B, Table B-1. The start and end of each stage are defined in Section 3.4.

For a service that processes many requests after a single startup, total GPU energy over the lifecycle can be estimated with the following formula, and the amortization of the fixed startup and release overhead calculated from it:

$$E(N) = E_{start} + E_{release} + N \cdot e_{ss} \quad (1)$$

where N is the number of requests in the lifecycle of the same instance; E_{start} and $E_{release}$ are the GPU energy of the startup and release stages, in J; e_{ss} is the average GPU energy per

request during steady operation, in J/request. The fixed startup and release overhead amortized to each request is $\frac{(E_{start} + E_{release})}{N}$.

To calculate e_{ss} from steady-state J/token, multiply by the average completion tokens per request under the same load conditions. When using the energy data in Appendix B, Table B-2, convert kJ to J by multiplying by 1000.

This estimate applies when request load and run configuration match the measurement conditions, and does not include idle energy from additional gaps between requests. Actual full-lifecycle energy should still be taken from measured results.

2.3 Mapping Energy Efficiency to Electricity Cost

To help turn technical metrics into deployment decisions, this section gives a method for estimating electricity cost from energy metrics and states the limits of its applicability.

All energy values reported in Section 2.2 and Chapter 4 of this whitepaper are GPU energy. Estimating data center electricity cost from GPU energy requires adding two unmeasured layers of loss in turn: first, the CPU, memory, storage, network, and power conversion losses within the same server; second, facility losses such as data center cooling, power distribution, and lighting. Total data center electricity cost therefore cannot be derived directly from GPU J/token.

The conversion for electricity cost per token is as follows:

$$C_{elec} = \frac{e_{GPU}}{3.6 \times 10^6} \times \eta_{sys} \times PUE \times P_{elec} \quad (2)$$

where: e_{GPU} is GPU energy per token, in J/token, which is the basis reported in the text and figures of this whitepaper; dividing by 3.6×10^6 converts it to kWh/token, and this intermediate result is written E_{GPU} (in kWh/token) to avoid using the same symbol as the J/token basis. η_{sys} is the ratio of whole-server energy to GPU energy in the same statistical window, equivalent to the ratio of average whole-server power to average GPU power in that window; PUE is data center power usage effectiveness; P_{elec} is the electricity price (CNY/kWh).

η_{sys} must be taken from the same statistical window as e_{GPU} . The η_{sys} of the steady operation stage cannot be applied directly to the startup or resource release stages: during those stages the GPUs may be loading or idle, so the server-to-GPU power ratio differs from steady state. Applying the steady-state η_{sys} to the full-lifecycle basis therefore introduces additional error; if electricity cost conversion is needed on the full-lifecycle basis, the η_{sys} values for the corresponding stages and their sources should be given separately.

The whole-server metering boundary is defined as the energy at the server power supply input, which includes CPU, memory, storage, network, and power conversion losses. Data center facility losses such as cooling, power distribution, and lighting are not counted again and are expressed uniformly through PUE . Neither η_{sys} nor PUE was measured in this test; their values depend on the specific server and facility conditions and are external assumptions rather than measured results. Until measured data are available, this section offers no electricity cost conclusions.

As a conversion example, take the steady-state median for text chat, $e_{GPU} = 0.747$ J/token, and assume $\eta_{sys} = 1.30$, $PUE = 1.40$, and $P_{elec} = 1.0$ CNY/kWh:

$$C_{elec} = \frac{0.747}{3.6 \times 10^6} \times 1.30 \times 1.40 \times 1.0 \approx 3.8 \times 10^{-7} \text{ CNY /token} \quad (3)$$

That is, the electricity cost per million tokens is about CNY 0.38. This value depends on all three external assumptions above and covers only the steady operation stage, excluding the fixed costs of model loading and service release; on a full-lifecycle basis, the unit cost in short-task scenarios would be noticeably higher. This example therefore only illustrates the conversion path and is not a basis for external quotes or cost commitments.

Note that if other values are used for η_{sys} and PUE , the result changes proportionally; the sources and applicable scope of those values should be disclosed at formal publication.

Chapter 3: Test Method and Experimental Design

3.1 Test Environment

All tests run on the same hardware platform and common test baseline (TP8, sequence scheduling limit 128) to eliminate the effect of platform differences on efficiency and performance. The evaluated models differ in software configuration, involving inference engine versions, images, and quantized artifacts, and some models use a specific development build of the inference engine. These model-specific software differences are retained in the tests and recorded with the formal results. On the hardware side, the test platform is a local server with 8 GPUs. On the software side, we built a unified, freezable benchmarking toolchain.

Table 3-1 summarizes the hardware and software environment of the test point in this section. To make results verifiable, the table lists every environment element needed to reproduce the same test point; items marked "to be supplied" are not yet disclosed in the current draft and need to be filled in from the test records.

Table 3-1 Test Platform and Reproducible Environment

Item	Configuration
GPU model and form factor	NVIDIA H100 80GB HBM3
Number of GPUs	8
Memory per GPU	80 GB
CPU configuration	Dual-socket CPU, 96 physical cores, 192 logical cores
System memory	About 1 TiB
GPU driver version	560.35.03
CUDA version	Container CUDA runtime 12.9.1 or 12.9.2, pinned per model image
Container image and tag	Pinned per model; image digest recorded with the formal results
Model revision	See Appendix A, which lists full repository IDs, revisions, and commit dates
Benchmark framework	ml-energy/benchmark v3.0
Energy measurement library	Zeus 0.13.1 (NVML)
Inference engines	vLLM and SGLang; specific versions pinned per model configuration and recorded with the formal results
Steady-state threshold	CV < 0.03 over 100 consecutive requests

The benchmark framework is ml-energy/benchmark v3.0, accepted as a NeurIPS 2025 Datasets & Benchmarks Spotlight. It drives the tests and collects energy and performance metrics.

The main inference engine is **vLLM**, with versions locked through Docker containers to avoid CUDA or PyTorch version drift. The tests used vLLM 0.29.x as the main line, with SGLang enabled as an alternative for compatibility issues with individual models. In the first round,

MiMo-V2.5 used SGLang and the other models mainly used vLLM; 6 models were then selected for SGLang tests to show the effect of the inference engine.

All energy values reported in this whitepaper are GPU energy. Energy is measured with the **Zeus 0.13.1** library. Where hardware and permissions allow, its monitoring interface can provide GPU, CPU, and memory energy readings; whether CPU and DRAM readings are available depends on the specific hardware and runtime environment and is not guaranteed. Only GPU-related readings are used here; CPU and memory readings are not included in any metric in this whitepaper. The measurement also uses two parallel observation windows. One covers the interval from the first formal request to the completion of the last formal request and is used to evaluate formal-request energy. The other is the interval after the system is statistically determined to have reached steady state; only energy in this interval is used to compute the key joules-per-token metric, excluding instability from cold starts and bursty queuing. Steady state is determined with a sliding window: candidate windows are formed in request completion order, sliding forward one request at a time, with a window length of 100 requests. The arithmetic mean of per-request J/token in the window is taken as the window's energy level, and the standard deviation of per-request J/token divided by the mean gives the coefficient of variation (CV). When consecutive non-overlapping windows all satisfy $CV < 0.03$, the system is deemed to have reached steady state, and the steady-state window runs from the start of the first qualifying window to the end of the formal requests. If no window meets the threshold by the end of the formal requests, the run is recorded as not reaching steady state; its steady-state J/token is excluded from cross-model comparison and flagged in the results. Energy for concurrent requests is attributed by each request's generation interval, and GPU energy in overlapping intervals is split in proportion to each request's output tokens in that interval.

We also deployed a Prometheus monitoring system on a side path to collect runtime data for later steady-state verification.

3.2 Test Datasets

The tests use three datasets corresponding to three typical production scenarios: text chat, code completion, and multimodal image question answering. They come from real human-model conversations on LMArena, real codebase context from Sourcegraph, and user-uploaded images and text from VisionArena. All are public real interaction or code corpora rather than synthetic templates, so the results are relevant to real deployments. Key parameters of the test datasets are shown in Table 3-2.

Table 3-2 Key Parameters of the Test Datasets

Scenario	Dataset Source	Requests	Max Output Tokens	Avg. Input Images	Avg. Input Text Tokens
Text chat	lmarena-ai/arena-human-preference-100k	256	1,024	—	About 580
Code completion	sourcegraph/context-aware-fim-code-completions	1,024	2,048	—	About 2k
Multimodal image-text QA	lmarena-ai/VisionArena-Chat	1,024	4,096	1	—

Notes:

1. The main test point for all three scenarios is TP=8 with a sequence scheduling limit of 128. The unified test point is used to compare deployment results under the same hardware configuration and does not represent each model's best configuration.
2. The general text chat data are real conversations between users and mainstream models collected on the LMArena platform, covering Chinese and English Q&A, programming, writing, reasoning, and other task types. We sampled 256 conversations with a fixed random seed, preserved the original prompt length distribution, and capped maximum output at 1024 tokens to simulate typical reply lengths in everyday conversation.
3. The code completion data use Sourcegraph's real codebase context completion dataset. Average input length is about 2k tokens; the test uses 1024 requests, with output capped at 2048 tokens.
4. The multimodal image question answering data come from user-uploaded images and text questions on VisionArena. Each request contains one image and a question, for 1024 requests in total. We also relaxed the maximum output length to 4096 tokens to measure the energy characteristics of joint vision-encoder inference and long generation.

The three datasets differ in input length, output length, and modality, and together cover the main working modes of large models in real applications.

3.3 Test Parameters

Benchmark 1.5 compares models at a unified test point, with tensor parallelism set to 8 and the sequence scheduling limit set to 128. Compared with the TP × BS multi-configuration sweep of Benchmark 1.0, this round focuses on expanding model coverage and measuring the full lifecycle. Workloads and request parameters are given in Table 3-2, and run parameters in Table 3-3.

Table 3-3 Unified Test Point and Run Parameters

Parameter	Setting	Description
Tensor parallelism (tensor-parallel-size)	8	8 GPUs used throughout
Sequence scheduling limit (max-num-seqs)	128	Upper limit on sequences the server can schedule per step; not equal to the actual batch size
Client concurrency limit	Not set	Requests enter the server queue and are scheduled by the inference engine
Warm-up requests	8	Max output 16 tokens, concurrency 1
Random seed	48105	Fixed data sampling

The driver sets no concurrency limit; requests are processed in batches by the inference engine according to the sequence scheduling limit. SGLang uses the configuration functionally equivalent

to vLLM. The unified test point is used to compare real deployment results under the same resource conditions and does not represent each model's best configuration.

Differences arising from model architecture, quantization, and inference engine follow the actual configuration.

The thinking settings and token counting basis used in the tests are given in Appendix B, Table B-1, as the basis for interpreting energy per output token results. Inference engine versions are given in Table B-3.

3.4 Test Procedure

Every test follows the same procedure:

Test preparation: Check the model ID, revision, precision, model snapshot, inference engine, image, parallel configuration, and request set, and confirm that GPUs, storage, leftover processes, and containers are in a normal state.

Start lifecycle monitoring: Collect 20 seconds of idle power before the test to confirm the GPUs' initial state. The startup stage uses driver-level GPU board power telemetry, sampling each GPU's instantaneous power at roughly 1-second intervals. The measurement runs from the first valid sample to the first formal request and includes service startup, model loading, service readiness, and warm-up. Power at stage boundaries is linearly interpolated, and each GPU's energy is integrated with the composite trapezoidal rule and then summed.

Start the service and verify its state: Start the inference service with the frozen configuration and monitor its state continuously. Once the service is ready, re-check model identity, revision, inference engine, context length, and parallel configuration to ensure the actual runtime environment matches the test plan.

Warm-up run: Tests use a cold start of the inference service. Model weights are stored in advance as a fixed local snapshot; each test creates a new inference instance with its own engine run directory and compile cache directory, while the operating system's file page cache is left in its existing state. After startup, 8 informal requests are run to trigger first-request handling, JIT and kernel compilation, and tokenizer processing. Warm-up requests are not counted in formal performance metrics; GPU energy during weight loading, compilation, and warm-up is counted in the startup stage, and network downloads are not counted in startup energy.

Formal requests: The driver submits all requests without limiting request rate or client concurrency; the inference service schedules according to the maximum concurrent sequences defined by `max-num-seqs=128`, and the number of in-flight requests on the driver side can exceed this value. During the test, each request's success, input and output token counts, time to first token (TTFT), time per output token (TPOT), inter-token latency (ITL), and end-to-end latency are recorded, and request throughput and output token throughput are computed.

Steady-state measurement: GPU energy during the formal request stage is measured by Zeus via NVML. Prometheus simultaneously collects queue, KV cache, and scheduling state from the inference engine's `/metrics` endpoint for run-state analysis only; it is not used to determine steady-state boundaries or compute GPU energy.

Service shutdown and resource release: After the last formal request completes, stop the inference service and continue collecting board power and GPU memory usage. The resource release stage starts at the completion of the last formal request and ends when the process exits, GPU memory is released, and GPU power returns to within the allowed range of the idle baseline. Release energy uses the same boundary handling and integration method as the startup stage; runs that do not recover resources within the specified time are excluded from the formal results.

Result verification and archiving: After all requests complete and the energy windows close, the framework generates a standard result file and saves the run configuration, lifecycle data, GPU telemetry, request records, and run logs for verification, aggregation, and traceability.

Benchmark 1.5 extends the test boundary from the running and steady stages of Benchmark 1.0 to the full service lifecycle:

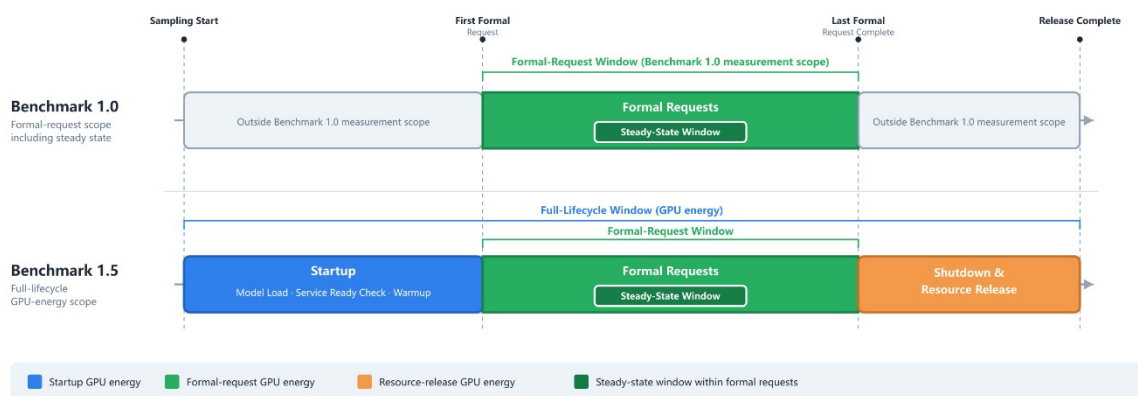


Figure 3-1 Measured Lifecycle in Benchmark 1.0 and Benchmark 1.5

Full-lifecycle GPU energy consists of three stages: startup, formal requests, and resource release. Formal-request and steady-state energy are measured by Zeus/NVML; the startup and resource release stages use GPU power sampling at roughly once per second, integrated with the trapezoidal rule over the intervals between adjacent samples. The stage boundaries are: the startup stage runs from the start of continuous GPU sampling to the start of formal requests, covering model loading, service readiness, and warm-up; the formal request stage runs from the first formal request to the completion of the last; the resource release stage runs from the end of formal requests until the inference process exits, GPU memory is released, and GPU power returns to within the allowed range of the idle baseline. Full-lifecycle energy uses the measured value of each stage without subtracting idle baseline power; the idle baselines before and after the test are used only to confirm the GPUs' idle state and resource recovery after the test. The allowed range of the idle baseline is defined as each GPU's power falling back to within $\pm 5\%$ of its pre-test baseline and holding for at least 30 seconds, at which point resource release is deemed complete. If power has not returned to this range within the maximum wait time, the run is recorded as timed out and its release stage interval is flagged as not closed. Prometheus is used only to explain inference engine run state and is not a source for energy accounting.

Although the overall procedure is the same, individual models need some special handling because of differences in architecture, quantized artifacts, and inference engine, as detailed in Table 3-4.

Table 3-4 Special Handling of Each Model in Testing

Model	Inference Engine	Main Adaptations
GLM-5.3-Flash	Specific development build of vLLM; SGLang	Special inference engine version and image identity
MiniMax-M3-MXFP8	vLLM	Uses MXFP8 model artifact
MiMo-V2.5	SGLang	First formal round run on the SGLang path
gpt-oss-120b	vLLM; SGLang	Uses MXFP4 model artifact
Gemma-4-31B-it	vLLM	Uses BF16 artifact
Sarashina2.2-3B-Instruct-v0.1	vLLM; SGLang	Uses the official 3B dense BF16 model artifact

After each test, the completion of formal requests, actual run configuration, lifecycle measurement boundaries, and GPU telemetry are verified, and it is confirmed that the inference process has exited, GPU memory has been released, and GPU power has recovered. Only after all checks pass are the results included in the formal results; tests that fail verification keep their original records, and re-tests generate separate records without overwriting existing data.

To make the steady-state determination verifiable, each formal run records the model name, test task, inference engine, and steady-state result. When steady state is reached, the start and end times of the steady-state window and the corresponding start and end request numbers are saved, together with GPU energy (J) and output tokens within the window. The computed steady-state energy per output token (J/token) is archived with them.

Total output tokens for the formal request stage are also saved, noting the token count used to compute full-lifecycle energy per output token. The result of whether GPU power returned to the allowed range of the idle baseline during resource release is archived as well.

Steady-state energy per output token is computed and used for cross-model comparison only for runs that reached steady state. Runs that did not reach steady state are flagged in the results table with the reason and are excluded from comparison on this metric.

Chapter 4: Test Results

This chapter analyzes the 48 formal test results of Benchmark 1.5. The first cross-model round covered 13 models and produced 33 results, using vLLM as the main inference engine, with MiMo-V2.5 on SGLang. Six of these models were then selected for supplementary SGLang tests, producing 15 results. Based on the first-round results, this chapter compares each model's energy consumption and throughput in text chat, code completion, and image-text QA, analyzes the composition of lifecycle energy and the behavior of representative models, and compares deployment behavior under vLLM and SGLang using the supplementary results.

This chapter divides models by total parameter count into three size groups, $\leq 50\text{B}$, $>50\text{B}$ and $<200\text{B}$, and $\geq 200\text{B}$, and compares them within each group under the same task scenario and evaluation metrics. Parameter scale is used for grouping and does not imply equal computation. Each model's results come from a single test under the corresponding deployment configuration; where a group has several applicable models they are ordered by the metric, and where it has only one its value is listed.

4.1 Energy Consumption Comparison

4.1.1 Steady-State Energy Consumption Across Scenarios

At the unified test point of TP8 and a sequence scheduling limit of 128, we compare each model's steady-state energy per output token in text chat, code completion, and image-text QA. A lower value means less GPU energy is needed per unit of output. Results are shown in Figures 4-1(a) to 4-1(c).

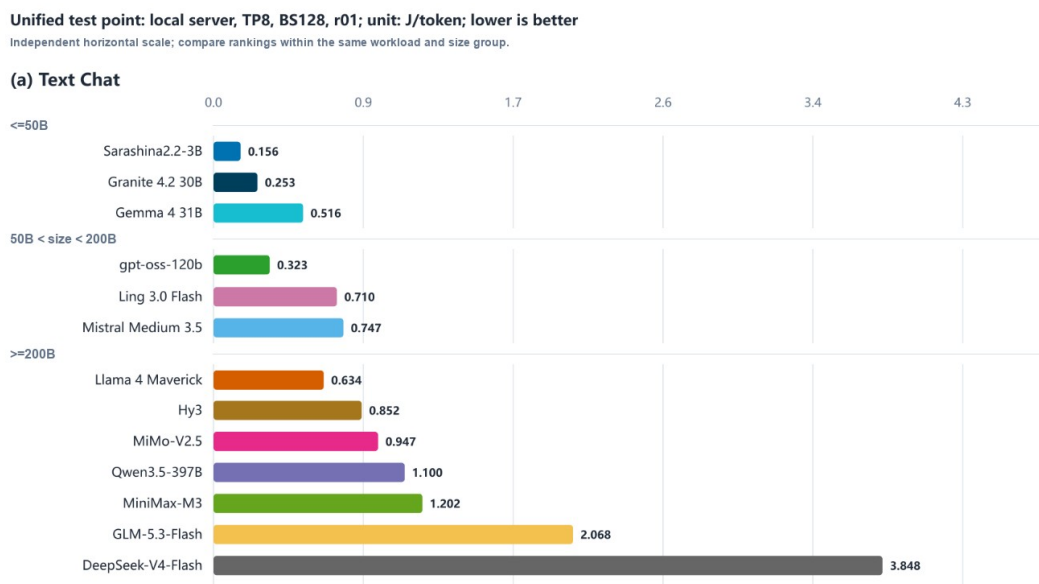


Figure 4-1(a) Steady-State Energy per Output Token, Text Chat

Text chat includes 13 models, with an overall median steady-state energy per output token of 0.747 J/token. In the lightweight, medium, and large groups, Sarashina 2.2, gpt-oss-120b, and

Llama 4 Maverick respectively have lower steady-state energy per output token than the other models in their groups, at 0.156, 0.323, and 0.634 J/token.

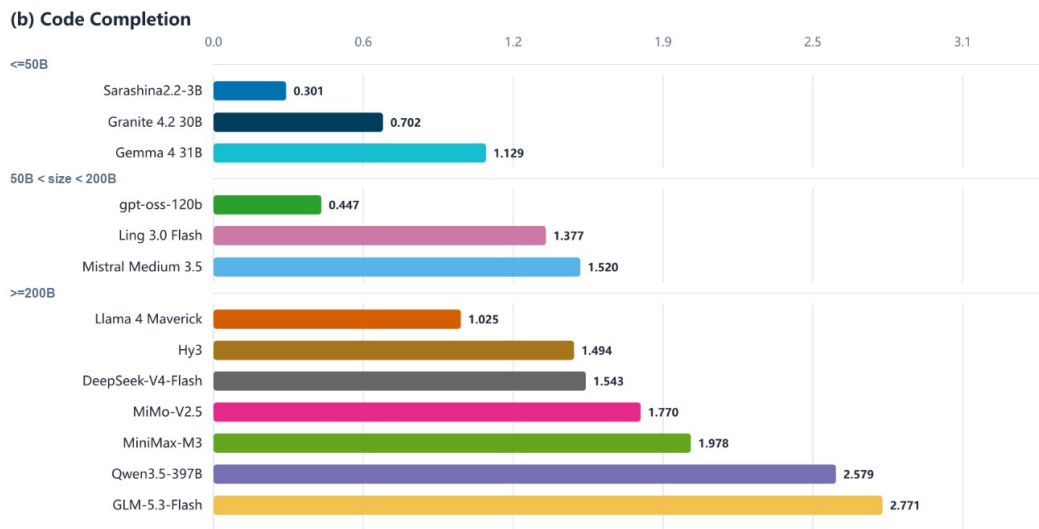


Figure 4-1(b) Steady-State Energy per Output Token, Code Completion

Code completion includes 13 models, with an overall median steady-state energy per output token of 1.494 J/token. In the lightweight, medium, and large groups, Sarashina 2.2, gpt-oss-120b, and Llama 4 Maverick respectively have lower steady-state energy per output token than the other models in their groups, at 0.301, 0.447, and 1.025 J/token.

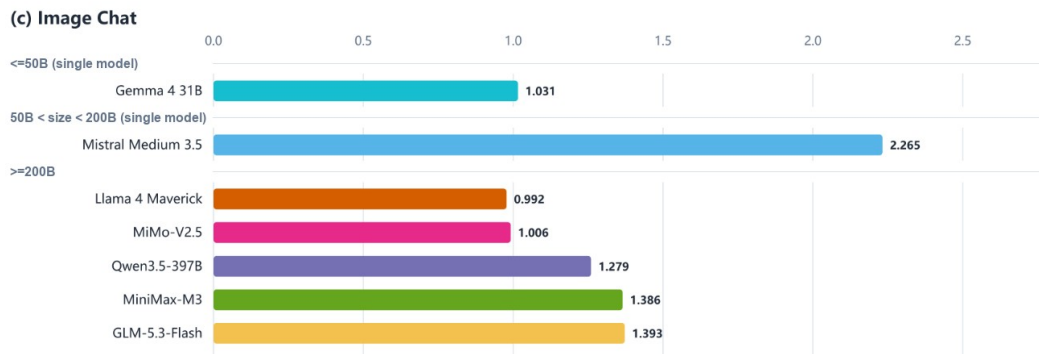


Figure 4-1(c) Steady-State Energy per Output Token, Image-Text QA

Image-text QA includes 7 models, with an overall median steady-state energy per output token of 1.279 J/token. In the large group, Llama 4 Maverick, MiMo-V2.5, and Qwen3.5 have steady-state energy per output token of 0.992, 1.006, and 1.279 J/token respectively, all lower than the other models in that group. The lightweight and medium groups each have one evaluated model, Gemma 4 31B and Mistral Medium 3.5, at 1.031 and 2.265 J/token respectively.

Cross-scenario comparison

To keep the evaluated set consistent, the cross-scenario comparison uses the 7 models that completed all three scenarios. Their median steady-state energy per output token is 0.947, 1.279, and 1.770 J/token for text chat, image-text QA, and code completion respectively; image-text QA and code completion are about 35% and 87% higher than text chat. For 5 of these models, steady-state energy per output token rises in the order text chat, image-text QA, code

completion. The scenario loads differ: text chat averages about 580 input tokens, code completion about 2k input tokens, and image-text QA involves image encoding.

Among the 7 common models, Llama 4 Maverick shows relatively small cross-scenario variation: the ratio of its highest to lowest steady-state energy per output token across the three scenarios is 1.62, lower than the other 6 models. MiMo-V2.5's image-text QA energy is close to its text chat energy, only about 6.3% higher, while its code completion energy is about 87% higher than text chat, so its cross-scenario variation is concentrated in code completion.

GLM-5.3-Flash and Mistral Medium 3.5 do not follow the same order as the 5 models above. GLM's image-text QA energy is lower than its text chat energy; Mistral's image-text QA energy is higher than its code completion energy. The overall order across the three scenarios therefore does not hold for every model, and deployment decisions should still look at the target model's results for the relevant task.

For multi-task deployment, models should be compared under the same task scenario and test configuration, and selection should combine each task's steady-state energy per output token, throughput, latency, and full-lifecycle energy per output token.

4.1.2 Full-Lifecycle Energy Consumption Heatmaps Across Scenarios

Figures 4-2 to 4-4 show the energy heatmaps for text chat, code completion, and image-text QA respectively. The first round of Benchmark 1.5 used the unified test point of TP8 and a sequence scheduling limit of 128, and each figure shows steady-state and full-lifecycle energy per output token side by side. Each figure is grouped by model size, with models within a group ordered from lowest to highest full-lifecycle energy per output token; the two columns compare steady-state and full-lifecycle energy per output token within the corresponding size group.

For ease of comparison, this section uses two statistics: the magnitude ratio and the Spearman rank correlation coefficient. The magnitude ratio is the median full-lifecycle J/token divided by the median steady-state J/token in the same scenario; a larger value means a larger overall gap between the two bases once startup and resource release are included. The Spearman rank correlation coefficient compares the model order under the two bases and ranges from -1 to 1; the closer to 1, the more consistent the order. For example, 0.74 means the two orders are broadly consistent but some models change position.

Unified test point: local server, TP8, BS128, r01; unit: J/token

Rows sorted within size groups by full-lifecycle J/token (ascending).

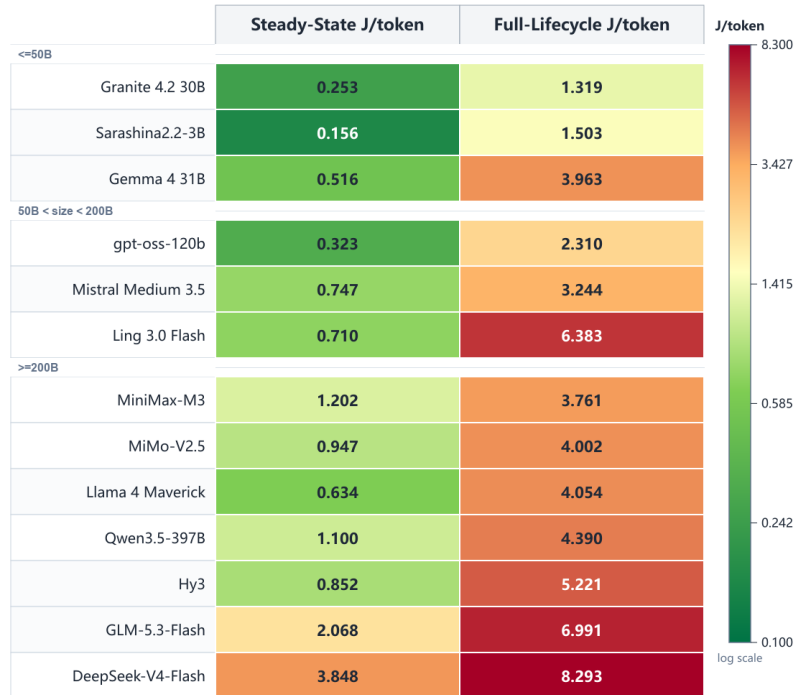


Figure 4-2 Energy Heatmap, Text Chat

In text chat, the median steady-state and full-lifecycle energy per output token are 0.747 and 4.002 J/token, a magnitude ratio of 5.36. The Spearman rank correlation between the model orders under the two bases is about 0.74; they are broadly similar, but the relative energy of some models changes. For example, Sarashina has lower steady-state energy per output token than Granite but higher full-lifecycle energy per output token; Ling has lower steady-state energy per output token than MiniMax but higher full-lifecycle energy per output token.

The formal run time in this text chat test is short, so startup and resource release overheads account for a large share of full-lifecycle energy per output token. The steady-state metric measures the GPU energy needed per unit of output while a service runs steadily; the full-lifecycle metric spreads the total GPU energy of the startup, formal run, and resource release stages over the output tokens of the formal request stage, measuring energy per output for one complete service run.

Unified test point: local server, TP8, BS128, r01; unit: J/token

Rows sorted within size groups by full-lifecycle J/token (ascending).

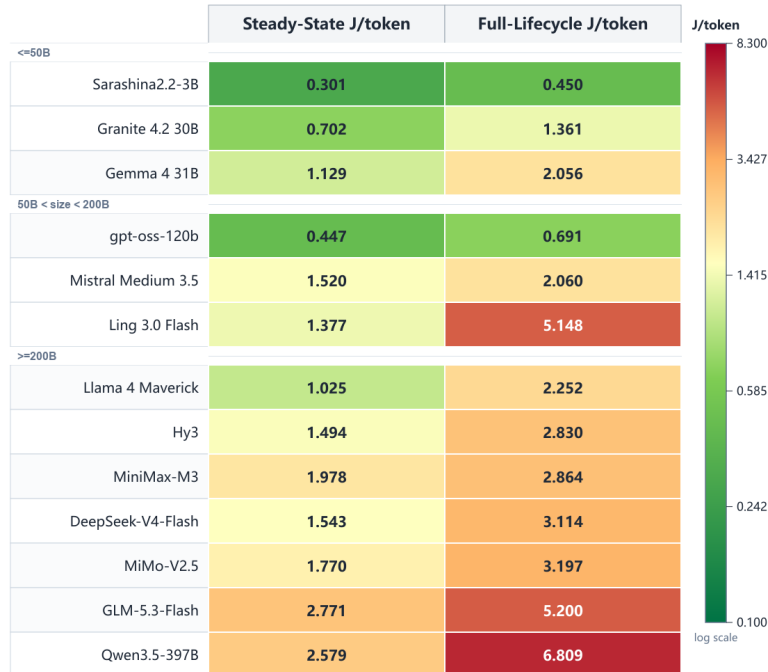


Figure 4-3 Energy Heatmap, Code Completion

In code completion, the median steady-state and full-lifecycle energy per output token are 1.494 and 2.830 J/token, a magnitude ratio of 1.89. The Spearman rank correlation between the model orders under the two bases is about 0.86, indicating broadly consistent orders. Compared with text chat, the steady-state and full-lifecycle orders are closer in code completion.

Unified test point: local server, TP8, BS128, r01; unit: J/token

Rows sorted within size groups by full-lifecycle J/token (ascending).

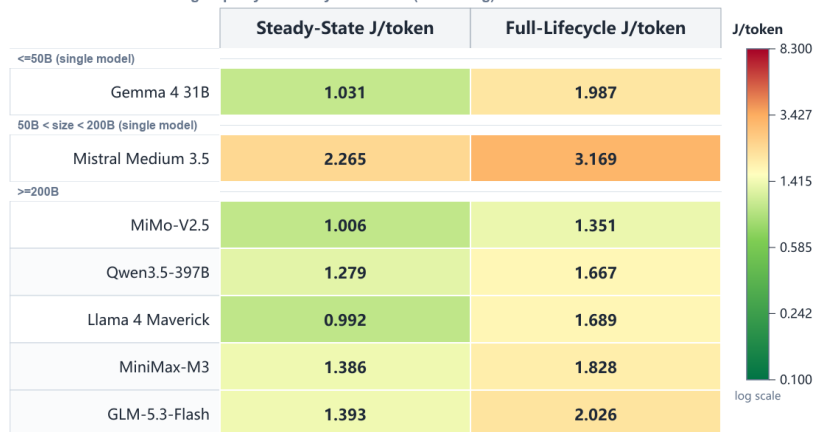


Figure 4-4 Energy Heatmap, Image-Text QA

In image-text QA, the median steady-state and full-lifecycle energy per output token are 1.279 and 1.828 J/token, a magnitude ratio of 1.43, the smallest of the three scenarios. The Spearman rank correlation between the model orders under the two bases is about 0.75. On the steady-state basis, Llama 4 Maverick has lower energy per output token than MiMo-V2.5 and Qwen3.5; on the full-lifecycle basis, MiMo-V2.5 and Qwen3.5 both have lower energy per output token

than Llama 4 Maverick. The overall gap between the two bases is small, but the relative energy of some models still changes.

Cross-scenario comparison

In terms of values, the median full-lifecycle energy per output token for text chat, code completion, and image-text QA is 5.36, 1.89, and 1.43 times the corresponding steady-state median. The gap between the two bases is therefore largest in text chat, followed by code completion, and relatively small in image-text QA.

In terms of overall order correlation, using all evaluated models in each scenario, we compare the consistency of the orders by steady-state and full-lifecycle energy per output token. The Spearman rank correlation for code completion is about 0.86, higher than 0.74 for text chat and 0.75 for image-text QA, indicating that in code completion the model orders by steady-state and full-lifecycle energy per output token are closer.

Building on steady-state measurement, Benchmark 1.5 adds full-lifecycle measurement covering the startup, formal run, and resource release stages, providing an energy evaluation basis for different service patterns. Long-running services that process requests continuously can rely mainly on steady-state J/token to evaluate energy per output during steady operation; services involving cold starts, short runs, or elastic scaling can add full-lifecycle J/token to evaluate the complete service energy including startup and resource release overhead.

4.2 Structure of Lifecycle Energy Consumption

Figure 4-5 shows the shares of startup, formal run, and resource release energy for the 13 models in text chat. Each model uses its main cross-model result: MiMo-V2.5 uses its SGLang result and the other 12 models use their vLLM results. Absolute GPU energy for the startup, formal run, resource release, and full lifecycle of each test is given in Appendix B, Table B-2, in kJ.

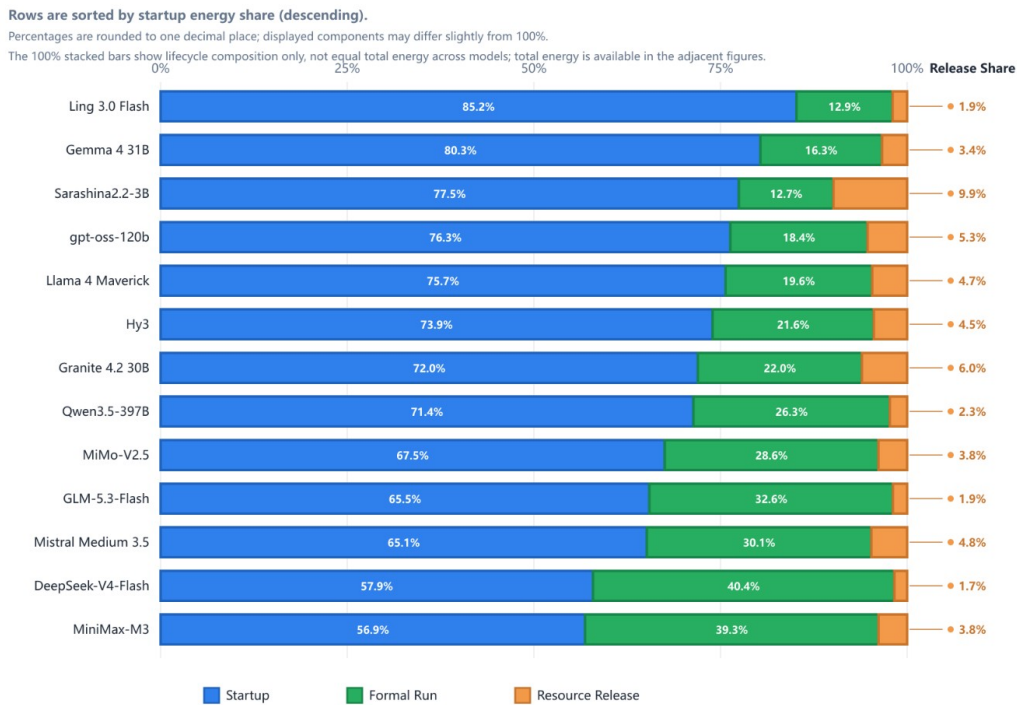


Figure 4-5 Lifecycle Energy Structure of Representative Models, Text Chat

In this text chat test, startup energy accounts for more than 50% of full-lifecycle GPU energy for all 13 models; the smallest startup share is MiniMax at 56.87%. The median shares of the startup, formal run, and resource release stages are 71.97%, 21.98%, and 3.85%. At the current request volume, startup is the main component of each model's complete service energy.

The composition of stage energy differs between models. Ling, Gemma, and Sarashina have startup shares of 85.18%, 80.34%, and 77.46%, and formal run shares of 12.88%, 16.30%, and 12.68%; DeepSeek and MiniMax have formal run shares of 40.37% and 39.30%, and startup shares of 57.92% and 56.87%. These shares reflect actual deployment results for the model artifacts, inference engines, and test load used here. In deployment optimization, reducing repeated startups and increasing instance reuse helps lower cumulative startup energy. For configurations such as DeepSeek and MiniMax, where the formal run takes a larger share, energy during request processing also deserves attention, and optimization effects should be evaluated together with steady-state energy per output token and throughput.

The resource release stage takes a small share overall, but varies between models. Sarashina's release share is 9.86%, the highest of the 13 models, while some models are below 2%. In services that frequently create and destroy instances, the energy of service shutdown, process exit, and GPU memory release accumulates with each start and stop, so resource release should also be included in deployment evaluation.

Sarashina's startup share is 77.46%, higher than DeepSeek's 57.92%, but its full-lifecycle energy per output token is 1.503 J/token, lower than DeepSeek's 8.293 J/token. The denominator of the stage shares is each model's own total GPU energy, so they reflect the distribution of energy within that model. When evaluating deployment options, full-lifecycle J/token compares energy per output for a complete service, while stage shares help analyze the main sources of energy and identify where to focus optimization.

4.3 Throughput Comparison

Figures 4-6(a) to 4-6(c) show each model's output throughput in text chat, code completion, and image-text QA. Each figure is grouped by size, with models within a group ordered from highest to lowest throughput.

Output throughput (tok/s); higher is better

Rows are sorted by throughput within each total-parameter size group.

(a) Text Chat



Figure 4-6(a) Output Throughput, Text Chat

(b) Code Completion

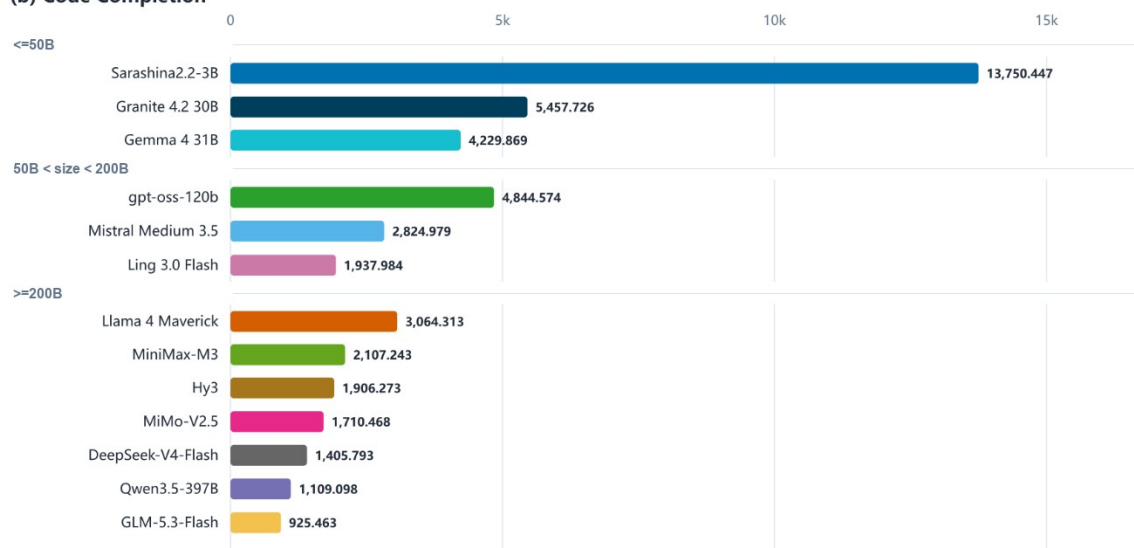


Figure 4-6(b) Output Throughput, Code Completion

Throughput characteristics of text chat and code completion

Model throughput orders in text chat and code completion are fairly consistent, with a Spearman rank correlation of about 0.79 across the 13 models. Under this test configuration, Sarashina 2.2's output throughput is 15,360 tok/s in text chat and 13,750 tok/s in code completion, the highest in the lightweight group in each scenario.

Qwen3.5 and MiniMax-M3 behave differently depending on the task. In text chat, Qwen3.5's output throughput is 3,924 tok/s, higher than MiniMax-M3's 2,738 tok/s; in code completion, MiniMax-M3's output throughput is 2,107 tok/s, higher than Qwen3.5's 1,109 tok/s.

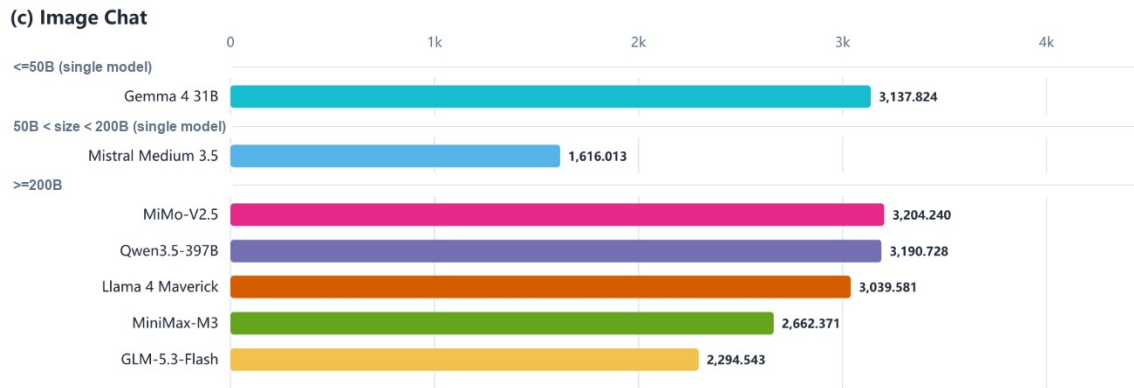


Figure 4-6(c) Output Throughput, Image-Text QA

Among the 7 models that completed all three tasks, the Spearman rank correlation of throughput is about 0.43 between text chat and image-text QA, and about -0.04 between code completion and image-text QA. The relative throughput of some models changes with the scenario. For example, in text chat Mistral Medium 3.5's output throughput is 3,581 tok/s, higher than GLM-5.3-Flash's 920 tok/s; in image-text QA GLM-5.3-Flash's output throughput is 2,295 tok/s, higher than Mistral Medium 3.5's 1,616 tok/s.

In image-text QA, MiMo-V2.5, Qwen3.5, Gemma 4 31B, and Llama 4 Maverick have similar output throughput, between 3,040 and 3,204 tok/s. MiMo-V2.5's output throughput is about 5.4% higher than Llama 4 Maverick's.

For deployment choices, evaluate a model's output capacity from measured throughput on the target task, combined with steady-state energy per output token, full-lifecycle energy per output token, time to first token (TTFT), and the quality requirements of the target task. The throughput results in this section correspond to the model artifacts, inference engines, and unified TP8 test configuration used here.

4.4 Analysis of Representative Models

4.4.1 Cross-Scenario Deployment Behavior of Gemma 4 31B

To analyze how the same model deploys under different workloads, this section uses Gemma 4 31B, which completed all three scenarios, as a representative case. Across its own three scenarios, its steady-state and full-lifecycle energy per output token rank differently, which makes it useful for showing how fixed startup cost affects deployment efficiency.

Gemma 4 31B uses a BF16 artifact, vLLM 0.29.0, TP8, a sequence scheduling limit of 128, and a 32K context. Figure 4-7 compares the model's throughput, steady-state energy, and full-lifecycle energy across the three scenarios.

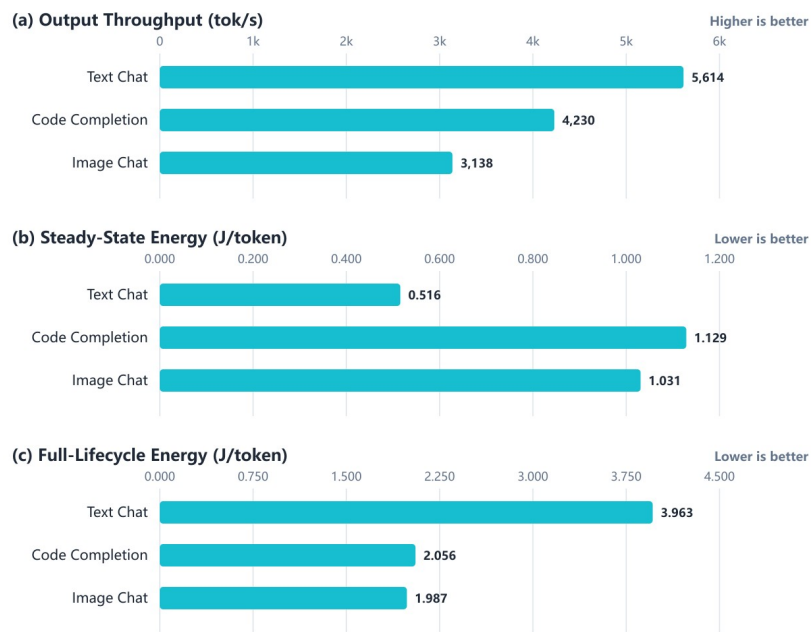


Figure 4-7 Throughput, Steady-State and Full-Lifecycle Energy per Output Token of Gemma 4 31B Across Three Scenarios

Across Gemma's three scenarios, text chat has the highest throughput at 5,614 tok/s and the lowest steady-state energy per output token at 0.516 J/token, but the highest full-lifecycle energy per output token at 3.963 J/token. Code completion and image-text QA have throughput of 4,230 and 3,138 tok/s, steady-state energy per output token of 1.129 and 1.031 J/token, and full-lifecycle energy per output token of 2.056 and 1.987 J/token. These comparisons refer only to Gemma's three scenarios.

The difference comes mainly from how fixed startup cost is spread. Startup times are similar across the three scenarios, 362–369 seconds; the formal run in text chat lasts only 24.0 seconds, much shorter than 113.8 seconds for code completion and 169.2 seconds for image-text QA, and produces fewer output tokens. Once fixed startup energy is spread over each output token, text chat ends up with the higher full-lifecycle energy per output token. The running cost of resident services can be evaluated mainly from steady-state results; services that start and stop frequently or run briefly also need the full-lifecycle results.

4.4.2 Comparison of Multimodal Models

Across the 7 image-text models, median throughput is 3,040 tok/s, and median steady-state and full-lifecycle energy are 1.279 and 1.828 J/token. The top four by throughput fall between 3,040 and 3,204 tok/s, yet their energy still differs clearly, showing that the image-text scenario now has a "similar performance, divergent cost" pattern. For such results, throughput works better as an entry condition and energy metrics as the way to distinguish deployment options.

Considering both throughput and steady-state energy per output token, Llama 4 Maverick and MiMo-V2.5 stand out. Llama has the lowest steady-state energy in the large group at 0.992 J/token; MiMo's steady-state energy is 1.006 J/token, only 1.4% higher, but its throughput reaches 3,204 tok/s, 5.4% higher than Llama. In these results, Llama's advantage is lower continuous-operation energy, and MiMo's is higher output throughput at similar energy.

Comparing throughput and full-lifecycle energy per output token, MiMo has both the highest throughput and the lowest full-lifecycle energy per output token in the large group in this test. Its full-lifecycle energy is 1.351 J/token, about 20.0% lower than Llama's 1.689 J/token. Qwen's throughput is 3,191 tok/s, only about 0.4% below MiMo, but its steady-state and full-lifecycle energy are about 27.1% and 23.4% higher.

Across scenarios, Llama's advantage leans toward consistency. Its steady-state energy in text chat, code completion, and image-text QA ranges from 0.634 to 1.025 J/token, with the maximum about 1.62 times the minimum; MiMo's range is 0.947 to 1.770 J/token, with the maximum about 1.87 times the minimum and a more pronounced rise in code completion. So if a workload targets only the current image-text QA load and values elastic service cost, MiMo's current deployment option has the advantage; if one model needs to cover text, code, and image-text tasks, Llama's steadier cross-scenario behavior has extra value. These judgments apply only to the model artifacts, inference engines, TP8, sequence scheduling limit of 128, and workloads used here; they do not concern answer quality, and the differences cannot be attributed to model architecture alone.

4.5 Deployment Behavior Under Different Inference Engines

vLLM. Uses PagedAttention, dividing the attention key-value cache (KV cache) into memory blocks allocated on demand, which reduces GPU memory fragmentation and over-reservation; uses continuous batching to adjust batches dynamically while requests execute. Its prefix caching reuses computation for identical input prefixes through hash matching of cache blocks.

SGLang. Also supports continuous batching and prefix caching. Its RadixAttention organizes request prefixes and their KV cache in a radix tree to identify and reuse shared-prefix caches; its overlap scheduling lets the CPU prepare the next batch while the GPU executes the current one, reducing scheduling waits.

The two engines differ in implementation across cache organization, request scheduling, memory configuration, operator adaptation, and output parsing. Based on the inference engine version and run configuration each model actually used, this section compares the overall throughput, latency, and energy of the two deployment options.

To balance service capacity, user response, steady-operation efficiency, and full-lifecycle energy per output token, this section uses four metrics: output throughput, TTFT P50, steady-state J/token, and full-lifecycle J/token. There are 15 formal SGLang results in total: 14 can be paired with vLLM results for the same model and task, and the other is a supplementary DeepSeek test with reasoning mode disabled, which is not part of the main comparison. Of the 14 pairs, Figure 4-8 includes only the 11 pairs where the relative difference in output tokens between the two engines is no more than 20%.

In the DeepSeek code completion, GLM-5.3-Flash image-text QA, and Granite 4.2 code completion tests, the relative difference in output tokens between vLLM and SGLang exceeds 20%. Those engine comparisons are not included in Figure 4-8 or in this section's deployment analysis. GLM-5.3-Flash image-text QA also has a difference in output parsing path.

The relative difference in output tokens here is defined as the absolute difference between the two output token totals divided by the larger of the two, calculated as in Equation (4).

$$\delta_N = \frac{|N_{\text{vLLM}} - N_{\text{SGLang}}|}{\max(N_{\text{vLLM}}, N_{\text{SGLang}})}, \delta_N \leq 0.20 \quad (4)$$

where N_{vLLM} and N_{SGLang} are the total output tokens in the formal request stage of the two deployment options for the same model and task.

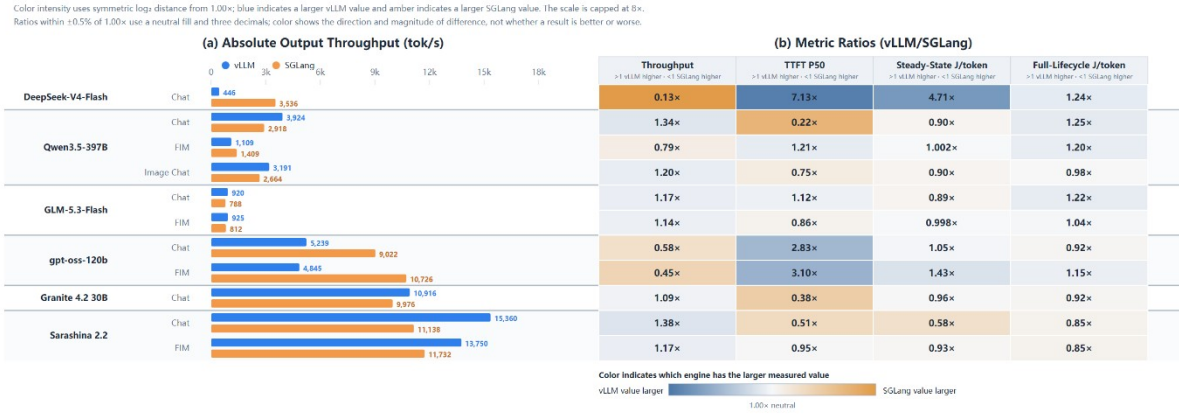


Figure 4-8 Changes in Key Metrics Between vLLM and SGLang

Note: All multiples in the figure are calculated as the vLLM measured value divided by the corresponding SGLang measured value. A multiple above 1 means vLLM's measured value is higher, and below 1 means SGLang's is higher; which is better depends on the direction of the metric. A value close to 1.00 means the two deployment options measured similarly on that metric.

DeepSeek-V4-Flash-0731 text chat has the largest relative throughput difference of the 11 pairs. Throughput is 446 tok/s on vLLM and 3,536 tok/s on SGLang, about 7.9 times higher on SGLang, with median time to first token about 86% lower. SGLang's energy advantage narrows across the complete service: steady-state energy per output token is about 79% lower than vLLM, but full-lifecycle energy per output token, calculated over startup, formal run, and resource release, is about 19% lower.

Among the other pairs, gpt-oss-120b code completion reaches 10,726 tok/s on SGLang, higher than 4,845 tok/s on vLLM, with lower TTFT P50 and lower energy per output token on both bases. For Qwen3.5 image-text QA, Granite 4.2 text chat, and both text tasks of Sarashina 2.2, vLLM has higher throughput, lower TTFT P50, and lower energy per output token. The size of the difference also varies by task for the same model: in Sarashina text chat, vLLM's steady-state energy per output token is about 42% lower than SGLang, while in code completion it is about 7% lower.

For Qwen3.5 and GLM-5.3-Flash code completion, the steady-state energy per output token ratios are 1.002 and 0.998, essentially equal; full-lifecycle energy per output token is lower on SGLang for both. On throughput and TTFT P50, Qwen3.5 does better on SGLang and GLM-5.3-Flash does better on vLLM.

In text chat, Qwen3.5 on vLLM has higher throughput and lower TTFT P50 and steady-state energy per output token, but 25% higher full-lifecycle energy per output token than on SGLang. gpt-oss-120b does better on SGLang on those three metrics, but its full-lifecycle energy per output token is about 8% lower on vLLM. GLM-5.3-Flash on vLLM has higher throughput and

lower steady-state energy per output token, while on SGLang it has lower TTFT P50 and lower full-lifecycle energy per output token.

These paired results show that the relative performance of vLLM and SGLang varies by model and task; deployment options should be evaluated on throughput, median time to first token, and both steady-state and full-lifecycle energy per output token.

4.6 Summary of Test Results

Taken together, the results support the following conclusions:

- 1. Different task scenarios have different levels of energy per output token.** Among the 7 models that completed all three scenarios, median steady-state energy per output token is 0.947, 1.279, and 1.770 J/token for text chat, image-text QA, and code completion, rising in that order, but not every model keeps the same cross-scenario order. Text chat, code completion, and image-text QA therefore need to be compared separately, and results in one scenario cannot be extrapolated directly to other tasks.
- 2. Steady-state and full-lifecycle energy reflect efficiency at different stages.** The magnitude ratios of full-lifecycle to steady-state median energy are about 5.36, 1.89, and 1.43 for text chat, code completion, and image-text QA, and all three scenarios show some change in model order. Steady-state J/token suits evaluating sustained efficiency once a service is running; full-lifecycle J/token suits evaluating one complete service run including startup, formal run, and resource release.
- 3. Startup accounts for a large share of full-lifecycle energy in text chat.** The median shares of startup, formal run, and resource release across the 13 models are 71.97%, 21.98%, and 3.85%, and even the model with the lowest startup share reaches 56.87%. Stage structure differs between models; lifecycle shares identify which stage energy is concentrated in, while full-lifecycle J/token compares energy per output for a complete service run. The two kinds of information cannot substitute for each other.
- 4. Throughput should be compared within the same task scenario and judged together with energy and latency.** Throughput orders in text chat and code completion are strongly correlated, while image-text QA changes models' throughput order significantly. The top four in image-text QA throughput fall between 3,040 and 3,204 tok/s; at that point, steady-state energy, full-lifecycle energy, and latency are more meaningful differentiators than small throughput differences.
- 5. Model selection should consider task scenario and evaluation metrics.** In text chat and code completion, Sarashina 2.2 in the lightweight group combines high throughput with low steady-state energy per output token, but its full-lifecycle energy per output token in text chat is higher than Granite 4.2's; gpt-oss-120b in the medium group is better than the other models in its group on throughput, steady-state, and full-lifecycle energy per output token. In the large group, Llama 4 Maverick has lower steady-state energy per output token than the other models in its group in all three scenarios, while MiMo-V2.5 combines high throughput with low full-lifecycle energy per output token in image-text QA.
- 6. Measured performance of vLLM and SGLang varies by model, task, and metric.** Among the 11 vLLM/SGLang pairs with output token differences of no more than 20%, some show SGLang

improving all four metrics, some show vLLM leading on all four, and some show throughput, TTFT, steady-state energy, and full-lifecycle energy moving in different directions. Choosing an inference engine requires comparing output throughput, TTFT, steady-state energy per output token, and full-lifecycle energy per output token together.

Chapter 5: Conclusions and Outlook

5.1 Core Conclusions

This test evaluated the energy and performance of 13 large models in text chat, code completion, and image-text QA. Building on the steady-state evaluation basis, Benchmark 1.5 extends the measurement scope to service startup, formal run, and resource release, and expands the evaluated models from 8 to 13: 5 continued, 3 updated, and 5 new. The results show:

1. **Full-lifecycle measurement reflects model service cost more completely.** Steady-state energy per output token mainly reflects generation efficiency during continuous operation, while full-lifecycle energy per output token also includes the costs of model loading, service startup, and resource release. Text chat runs are short, and Figure 4-5 shows that startup takes a large share of energy for most models. This section uses text chat to illustrate the lifecycle energy structure; stage shares for code completion and image-text QA have not yet been computed item by item, and their main energy stage cannot be judged from the ratio of full-lifecycle to steady-state J/token alone. Steady-state J/token alone is therefore insufficient to evaluate cold starts, short tasks, and elastic services.
2. **The efficient models differ clearly by scenario.** In text chat and code completion, Sarashina 2.2 and Granite 4.2 in the lightweight group and gpt-oss-120b in the medium group each form good combinations of energy and throughput; in the large group for image-text QA, Llama 4 Maverick has the lowest steady-state energy per output token, while MiMo-V2.5 has the highest throughput and the lowest full-lifecycle energy per output token. Model selection should be based on the specific workload, and results from a single scenario should not stand in for overall performance.
3. **Model scale is not the only factor determining efficiency.** Granite 4.2 and Sarashina 2.2 stand out under the unified eight-GPU baseline, while Gemma 4 31B did not retain its efficiency advantage from Benchmark 1.0. Model artifact, quantization, inference engine, context length, and parallel configuration all affect the final result, and small models may incur extra communication and synchronization overhead when tensor parallelism splits them too finely.
4. **A unified test configuration does not represent each model's optimal configuration.** Benchmark 1.5 uses TP8 and a sequence scheduling limit of 128 as a common test point, which helps compare deployment results under the same resources but cannot represent each model's optimal configuration. Gemma's results differ between Benchmark 1.0 and 1.5, but model version, inference engine, precision artifact, workload, and parallelism may all have changed between the two rounds, so the cross-round difference cannot be taken directly as evidence of mismatched parallelism. That parallelism needs to match model scale is only one possible explanation and must be verified with controlled experiments.
5. **Inference engines and model artifacts are an important part of efficiency evaluation.** In the first cross-model round, MiMo-V2.5 used SGLang and the other models mainly used vLLM; the evaluated models cover BF16, FP8 variants, MXFP4, and MXFP8 precision formats. The results

reflect actual deployment options made up of model, precision, inference engine, and configuration, and the differences cannot all be attributed to model architecture.

6. Expanding the model range makes the evaluation more representative. Benchmark 1.5 increases the evaluated models from 8 to 13: 5 continued, 3 updated, and 5 new, with the new lines being GLM-5.3-Flash, Hy3-FP8, Granite-4.2-30B-FP8, Ling-3.0-Flash-FP8, and Sarashina2.2-3B-Instruct-v0.1. The number of models completing image-text QA rose from 3 to 7, further broadening coverage of origins, parameter scales, and input modalities.

Overall, Benchmark 1.5 extends efficiency comparison from steady operation to evaluation of the complete service lifecycle. Model deployment should follow the principle that "the scenario sets the requirements, configuration affects efficiency, the inference engine unlocks capability, and the lifecycle reflects cost," weighing energy, throughput, latency, and service pattern together.

5.2 Future Outlook

For the long-term development of large-model technology and the intelligent computing industry, the evaluation framework and industry practice will continue to deepen in the following directions:

- 1. Extending to a full-stack evaluation framework.** The scope can extend from the current model inference layer up to agents, retrieval augmentation, tool calling, memory management, and multi-model collaboration, forming an end-to-end evaluation chain from underlying compute and inference services to intelligent applications. This helps reveal the real distribution of energy in complex intelligent systems, avoiding optimization of a single model while ignoring the cost of upper-layer orchestration and long-chain interactions.
- 2. Broadening the range of model architectures.** Current mainstream evaluation still focuses on Transformers and their MoE variants. Future work can cover state space models, linear attention, hybrid recurrent architectures, and other non-Transformer approaches, studying the efficiency of different computing paradigms in long-context, sparse-activation, and continuous-inference scenarios to provide a unified empirical basis for new model architectures.
- 3. Combining system energy with algorithmic capability.** Link system-side J/token, throughput, and latency metrics with algorithm-side accuracy, task completion rate, reasoning quality, and safety, building combined evaluation logic such as "effective intelligent output per unit of energy" under capability constraints. Drawing on established academic research, the quality-performance-energy frontier can be studied further to avoid apparent efficiency gains achieved at the expense of output quality.
- 4. Coordinating heterogeneous compute with energy systems.** The scope can extend from a single server to different GPUs, NPUs, CPUs, and domestic accelerators, and further link to data center PUE, renewable energy use, carbon emission factors, and power load characteristics. This direction connects model efficiency research with green data centers, compute scheduling, and new-type power systems, providing a data foundation for joint optimization of compute, power, and carbon.
- 5. Building standards and the industry ecosystem.** Promote reproducible, mutually recognized industry norms for efficiency metrics, test boundaries, data disclosure, and result interpretation, gradually serving cloud service pricing, enterprise model selection, green compute certification,

and policy evaluation. A public, transparent evaluation framework can drive continuous feedback among models, software, and hardware, shifting industry competition from pursuing capability and scale alone toward balanced development of capability, efficiency, and sustainability.

5.3 Limitations

This whitepaper has the following limitations, and we welcome comments and corrections from readers and industry colleagues:

1. **Limited repetitions.** Each configuration completed one formal test; means, standard deviations, and statistical significance from three repetitions are not yet available.
2. **The unified test point does not represent each model's best configuration.** This round mainly uses TP8 and a sequence scheduling limit of 128. Models of different scale, architecture, and parallelism may have different efficiency optima, so the results are mainly for deployment comparison under a common baseline.
3. **Deployment options are heterogeneous.** Models use different precisions, model versions, inference engine versions, and model-specific configurations. This round produced 15 formal SGLang results: 14 can be paired with vLLM results for the same model and task, and 1 is a supplementary test. Of the 14 pairs, 11 are included in Figure 4-8; DeepSeek FIM, Granite FIM, and GLM Image Chat are not used to compare the two inference engines because their output token counts differ by more than 20%.
4. **Energy boundary limited to GPUs.** All energy values in this whitepaper are GPU energy. Current results exclude CPU, memory, storage, network, cooling, and whole-server power supply losses; startup and resource release energy come from integrated GPU power samples and cannot be equated with data center wall-plug energy.
5. **Token counting affects cross-model comparison.** Different tokenizers, output lengths, and reasoning modes affect J/token and throughput results. J/token and output throughput are both calculated from the completion token count returned by the API; the thinking settings and token counting basis used in the tests are given in Appendix B, Table B-1. Cross-model judgments should also consider J/request, requests/s, and actual task completion.
6. **Workload and capability evaluation remain limited.** Current coverage is text chat, code completion, and image-text QA; long-horizon agent tasks, retrieval augmentation, video, multi-model collaboration, and specialized domain applications are not yet covered. This test also does not directly evaluate answer accuracy or task quality.

Appendix A: Evaluated Model Artifacts and Version Identifiers

This appendix lists the full weight repository IDs, pinned revisions, and commit dates for the 13 evaluated models in Table 2-2. Repository ID links point to the repository page for that revision.

Evaluated Model	Weight Repository ID	Revision	Commit Date
Qwen3.5-397B-A17B-FP8	Qwen/Qwen3.5-397B-A17B-FP8	9f1f3de9a3a48cfd340fd73ca98c02625b7afb3b	2026-02-26
Llama-4-Maverick-17B-128E-Instruct-FP8	RedHatAI/Llama-4-Maverick-17B-128E-Instruct-FP8	ad751fe44a9a92326ce966d980e325fbc2c832c3	2025-09-22
gpt-oss-120b	openai/gpt-oss-120b	b5c939de8f754692c1647ca79fbf85e8c1e70f8a	2025-08-26
Mistral-Medium-3.5-128B	mistralai/Mistral-Medium-3.5-128B	ed0c85631cf20929c29bc36d7cb37519ed0cdd28	2026-05-04
Gemma-4-31B-it	google/gemma-4-31B-it	439edf5652646a0d1bd8b46bfcd1d3645761a445	2026-04-10
DeepSeek-V4-Flash-0731	deepseek-ai/DeepSeek-V4-Flash-0731	7872f01b1d1fe23eabc4c98b48bffcef5a386062	2026-08-01
MiniMax-M3-MXFP8	MiniMaxAI/MiniMax-M3-MXFP8	c5454eb03678d8710e54a4e0fc681b9f3b4a3dba	2026-07-11
MiMo-V2.5	XiaomiMiMo/MiMo-V2.5	63651580ca774f8504f676040460aed3e1244ac1	2026-07-09
GLM-5.3-Flash	zai-org/GLM-5.3-Flash	eb9eb208eb0d988989d07a6a12d0fdeb5f52574a	2026-09-07
Hy3-FP8	tencent/Hy3-FP8	ecc1d8e194e093f33177f2f0ef7ce8f397b2d68b	2026-07-20
Granite-4.2-30B-FP8	ibm-granite/granite-4.2-30b-fp8	bee6d7463fd7064455f59ac480a3fcdc ae5cb801	2026-09-02
Ling-3.0-Flash-FP8	inclusionAI/Ling-3.0-flash-fp8	88d48c5a46e1b2f609144e83d95b8ac47528b12e	2026-09-04
Sarashina2.2-3B-Instruct-v0.1	sbintuitions/sarashina2.2-3b-instruct-v0.1	4f3626fb1b64b3e97c908e67f27b2d627ba2a999	2026-08-25

Note: The commit date is the date the listed revision was committed in the weight repository, given as a UTC date; it does not represent the first release date of the model or weights. In FP8, FP4, and MX low-precision artifacts, some modules may retain BF16 or other precisions; the specific configuration is as given in the repository files for the corresponding revision.

Appendix B: Per-Run Configuration and Stage Energy Consumption

This appendix gives the configuration and stage energy of each test, so the test conditions behind the results in the main text can be looked up. The 48 tests are divided into the first-round model tests, supplementary SGLang engine tests, and a supplementary DeepSeek test with thinking disabled. Tables B-1 and B-2 are linked by test number, and all energy values are totals across 8 GPUs. In the first round, MiMo-V2.5 used SGLang and the other models used vLLM.

Common cache conditions: Tests use a cold start of the inference service; weights are a fixed local snapshot prepared in advance; run and compile cache directories are set separately for each test; the operating system's file page cache is left in its existing state. Startup energy includes weight loading, compilation, and warm-up, and excludes network downloads.

Table B-1 Test Configuration and Thinking Settings

First-round model tests (33)

Corresponds to the cross-model comparison in Chapter 4.

No.	Model	Scenario	Inference Engine	Weight Precision	Thinking Setting
0003	Sarashina 2.2	Text chat	vLLM	BF16	Default setting
0004	Sarashina 2.2	Code completion	vLLM	BF16	Default setting
0012	Granite 4.2	Text chat	vLLM	FP8	Default setting
0013	Granite 4.2	Code completion	vLLM	FP8	Default setting
0035	Gemma 4 31B	Text chat	vLLM	BF16	Default setting
0036	Gemma 4 31B	Code completion	vLLM	BF16	Default setting
0034	Gemma 4 31B	Image-text QA	vLLM	BF16	Default setting
0024	gpt-oss-120b	Text chat	vLLM	MXFP4	Default setting
0025	gpt-oss-120b	Code completion	vLLM	MXFP4	Default setting
0014	Ling 3.0 Flash	Text chat	vLLM	FP8 E4M3	Default setting
0015	Ling 3.0 Flash	Code completion	vLLM	FP8 E4M3	Default setting
0020	Mistral Medium 3.5	Text chat	vLLM	FP8	Default setting
0021	Mistral Medium 3.5	Code completion	vLLM	FP8	Default setting
0019	Mistral Medium 3.5	Image-text QA	vLLM	FP8	Default setting
0008	DeepSeek-V4-Flash	Text chat	vLLM	FP8 + FP4	On by default
0009	DeepSeek-V4-Flash	Code completion	vLLM	FP8 + FP4	Default setting
0040	Hy3	Text chat	vLLM	FP8	Default setting

No.	Model	Scenario	Inference Engine	Weight Precision	Thinking Setting
0041	Hy3	Code completion	vLLM	FP8	Default setting
0043	MiMo-V2.5	Text chat	SGLang	FP8 E4M3	Default setting
0044	MiMo-V2.5	Code completion	SGLang	FP8 E4M3	Default setting
0042	MiMo-V2.5	Image-text QA	SGLang	FP8 E4M3	Default setting
0049	GLM-5.3-Flash	Text chat	vLLM	FP8 E4M3	Default setting
0050	GLM-5.3-Flash	Code completion	vLLM	FP8 E4M3	Default setting
0048	GLM-5.3-Flash	Image-text QA	vLLM	FP8 E4M3	Default setting
0032	Qwen3.5-397B	Text chat	vLLM	FP8	Default setting
0033	Qwen3.5-397B	Code completion	vLLM	FP8	Default setting
0031	Qwen3.5-397B	Image-text QA	vLLM	FP8	Default setting
0038	Llama 4 Maverick	Text chat	vLLM	FP8	Default setting
0039	Llama 4 Maverick	Code completion	vLLM	FP8	Default setting
0037	Llama 4 Maverick	Image-text QA	vLLM	FP8	Default setting
0017	MiniMax-M3	Text chat	vLLM	MXFP8	Default setting
0018	MiniMax-M3	Code completion	vLLM	MXFP8	Default setting
0016	MiniMax-M3	Image-text QA	vLLM	MXFP8	Default setting

Supplementary SGLang engine tests (14)

For comparison with vLLM results for the same model and task.

No.	Model	Scenario	Inference Engine	Weight Precision	Thinking Setting
0001	Sarashina 2.2	Text chat	SGLang	BF16	Default setting
0002	Sarashina 2.2	Code completion	SGLang	BF16	Default setting
0010	Granite 4.2	Text chat	SGLang	FP8	Default setting
0011	Granite 4.2	Code completion	SGLang	FP8	Default setting
0022	gpt-oss-120b	Text chat	SGLang	MXFP4	Default setting
0023	gpt-oss-120b	Code completion	SGLang	MXFP4	Default setting
0005	DeepSeek-V4-Flash	Text chat	SGLang	FP8 + FP4	Explicitly on

No.	Model	Scenario	Inference Engine	Weight Precision	Thinking Setting
0007	DeepSeek-V4-Flash	Code completion	SGLang	FP8 + FP4	Default setting
0046	GLM-5.3-Flash	Text chat	SGLang	FP8 E4M3	Default setting
0047	GLM-5.3-Flash	Code completion	SGLang	FP8 E4M3	Default setting
0045	GLM-5.3-Flash	Image-text QA	SGLang	FP8 E4M3	Default setting
0029	Qwen3.5-397B	Text chat	SGLang	FP8	Explicitly on
0030	Qwen3.5-397B	Code completion	SGLang	FP8	Default setting
0028	Qwen3.5-397B	Image-text QA	SGLang	FP8	Default setting

DeepSeek test with thinking disabled (1)

Lists the text chat result with thinking disabled separately.

No.	Model	Scenario	Inference Engine	Weight Precision	Thinking Setting
0006	DeepSeek-V4-Flash	Text chat	SGLang	FP8 + FP4	Explicitly off

Note: "Default setting" means the test parameters do not separately specify a thinking switch, and the behavior is determined by the model and inference engine; "On by default" is labeled according to the configuration record. J/token is calculated from the completion token count returned by the API.

Table B-2 GPU Energy by Test Stage (kJ)

First-round model tests (33)

No.	Model	Scenario	Startup Energy	Formal Run Energy	Release Energy	Total Energy
0003	Sarashina 2.2	Text chat	150.002	24.561	19.096	193.660
0004	Sarashina 2.2	Code completion	147.889	379.138	25.757	552.784
0012	Granite 4.2	Text chat	209.915	64.119	17.645	291.679
0013	Granite 4.2	Code completion	209.201	273.269	23.723	506.193
0035	Gemma 4 31B	Text chat	428.682	87.000	17.919	533.601
0036	Gemma 4 31B	Code completion	432.114	532.620	24.641	989.376
0034	Gemma 4 31B	Image-text QA	442.615	584.812	27.187	1054.614
0024	gpt-oss-120b	Text chat	305.774	73.658	21.220	400.652
0025	gpt-oss-120b	Code completion	320.174	657.298	28.030	1005.501
0014	Ling 3.0 Flash	Text chat	1054.382	159.398	24.070	1237.850

No.	Model	Scenario	Startup Energy	Formal Run Energy	Release Energy	Total Energy
0015	Ling 3.0 Flash	Code completion	1072.905	415.851	24.784	1513.540
0020	Mistral Medium 3.5	Text chat	279.065	128.877	20.508	428.450
0021	Mistral Medium 3.5	Code completion	273.513	995.478	26.070	1295.061
0019	Mistral Medium 3.5	Image-text QA	277.942	754.756	27.128	1059.826
0008	DeepSeek-V4-Flash	Text chat	885.103	616.880	26.198	1528.180
0009	DeepSeek-V4-Flash	Code completion	1054.118	1074.524	27.118	2155.760
0040	Hy3	Text chat	400.277	116.888	24.170	541.336
0041	Hy3	Code completion	388.812	586.474	27.249	1002.534
0043	MiMo-V2.5	Text chat	441.068	186.918	25.141	653.128
0044	MiMo-V2.5	Code completion	447.075	614.935	28.809	1090.820
0042	MiMo-V2.5	Image-text QA	449.530	1994.613	38.601	2482.743
0049	GLM-5.3-Flash	Text chat	923.078	460.091	26.702	1409.872
0050	GLM-5.3-Flash	Code completion	959.978	1028.126	27.025	2015.128
0048	GLM-5.3-Flash	Image-text QA	962.563	2529.878	32.350	3524.792
0032	Qwen3.5-397B	Text chat	785.070	289.457	25.389	1099.917
0033	Qwen3.5-397B	Code completion	783.167	494.229	27.509	1304.905
0031	Qwen3.5-397B	Image-text QA	795.462	3069.200	34.089	3898.750
0038	Llama 4 Maverick	Text chat	290.231	75.334	17.965	383.530
0039	Llama 4 Maverick	Code completion	274.728	293.973	24.271	592.973
0037	Llama 4 Maverick	Image-text QA	267.711	524.834	27.706	820.251
0017	MiniMax-M3	Text chat	353.892	244.546	23.868	622.307
0018	MiniMax-M3	Code completion	361.205	990.942	26.564	1378.711
0016	MiniMax-M3	Image-text QA	353.723	1759.398	29.337	2142.458

Supplementary SGLang engine tests (14)

No.	Model	Scenario	Startup Energy	Formal Run Energy	Release Energy	Total Energy
0001	Sarashina 2.2	Text chat	178.384	29.908	17.166	225.458

No.	Model	Scenario	Startup Energy	Formal Run Energy	Release Energy	Total Energy
0002	Sarashina 2.2	Code completion	180.677	367.744	28.592	577.013
0010	Granite 4.2	Text chat	221.829	73.890	23.548	319.268
0011	Granite 4.2	Code completion	224.904	261.292	27.320	513.515
0022	gpt-oss-120b	Text chat	354.201	62.286	19.260	435.746
0023	gpt-oss-120b	Code completion	350.967	471.525	31.828	854.320
0005	DeepSeek-V4-Flash	Text chat	1193.996	191.471	24.625	1410.092
0007	DeepSeek-V4-Flash	Code completion	1182.147	712.763	29.714	1924.624
0046	GLM-5.3-Flash	Text chat	655.851	470.434	25.304	1151.588
0047	GLM-5.3-Flash	Code completion	764.482	959.256	25.778	1749.516
0045	GLM-5.3-Flash	Image-text QA	609.840	1698.709	29.026	2337.576
0029	Qwen3.5-397B	Text chat	516.960	335.126	25.799	877.884
0030	Qwen3.5-397B	Code completion	509.992	476.035	28.686	1014.713
0028	Qwen3.5-397B	Image-text QA	513.598	3385.570	36.832	3936.001

DeepSeek test with thinking disabled (1)

No.	Model	Scenario	Startup Energy	Formal Run Energy	Release Energy	Total Energy
0006	DeepSeek-V4-Flash	Text chat	1173.428	163.273	24.568	1361.269

Note: Total energy is the sum of GPU energy across the startup, formal run, and resource release stages. Each stage is given to three decimal places, so the sum of the components may differ from the total in the last digit due to rounding. Amortization of fixed overhead by request volume is given by Equation (1).

Table B-3 Inference Engine Versions

Inference Engine	Version	Applicable Tests
vLLM	0.29.0	First-round tests; vLLM tests other than Ling 3.0 Flash, Mistral Medium 3.5, and GLM-5.3-Flash
vLLM	0.29.0-cu129	Ling 3.0 Flash first-round test
vLLM	0.20.1	Mistral Medium 3.5 first-round test
vLLM	0.28.1rc1.dev580+g385dce36b.cu129	GLM-5.3-Flash first-round test
SGLang	0.5.19-cu129	MiMo-V2.5 first-round test; SGLang supplementary tests other than GLM-5.3-Flash; DeepSeek test with thinking disabled
SGLang	nightly-dev-cu12-20260827-20621aa1	GLM-5.3-Flash SGLang supplementary test

Thank you for reading

Large Language Model Inference Energy Efficiency Evaluation Whitepaper (Benchmark 1.5)

Building a "Measure-Evaluate-Apply-Optimize" closed-loop system to advance sustainable AI