The Silent Hyperparameter: Quantifying the Impact of Inference Backends on LLM Reproducibility

David Pape **Jonathan Evertz** **Lea Schönherr**
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
{david.pape, jonathan.evertz, schoenherr}@cispa.de

Abstract

Progress in LLMs is increasingly measured through standardized benchmarks, where state-of-the-art improvements are often separated by fractions of a percentage point. At the same time, the computational cost of evaluating modern LLMs has driven widespread adoption of specialized inference backends, software systems that execute trained models efficiently at inference time. While critical for scalability, system-level optimizations, such as custom CUDA kernels and reduced-precision arithmetic, can alter token probabilities and introduce non-determinism, possibly cascading into divergent generation. In this work, we first survey the inference landscape, identifying 200 distinct engines, and analyze 35,000 ML publications, finding that the specific inference stack is rarely reported despite this widespread diversity. We then present a systematic empirical study of how inference backends affect LLM benchmark results. Holding model weights, decoding parameters, and hardware constant, we evaluate five widely used inference engines, including vLLM, SGLang, and llama.cpp, across multiple open-weight models and established benchmarks. We show that the choice of backend alone can shift benchmark scores by up to 16.6 percentage points and induce high rates of output disagreement. By isolating backend optimizations and tracing the execution pipeline, we find this divergence is driven by system-level optimizations like prefix caching and CUDA graphs, custom kernels, and engine-specific defaults in logit processing. Our findings identify the inference backend as a previously unreported but consequential hyperparameter in the evaluation of LLM and advocate standardized reporting of inference stacks to improve the reproducibility and interpretability of benchmark comparisons.

1 Introduction

The rapid advancement of Large Language Models (LLMs) has established a new standard in artificial intelligence. However, running these evaluations on powerful models introduces significant computational challenges, demanding immense memory and processing power. To address this, a rich ecosystem of specialized inference engines has emerged, with tools like vLLM [1], SGLang [2], and llama.cpp [3] becoming essential for efficient model serving. These *backends* employ sophisticated optimization techniques, such as paged attention [1], custom CUDA kernels, and optimized memory management, to reduce latency and increase throughput. Consequently, they are widely adopted not only in production but also by researchers for resource-efficient experimentation.

While essential for performance, these engines are complex systems, and *their internal optimizations can potentially alter model outputs.* Differences in floating-point accumulation, non-deterministic behavior in custom CUDA kernels, or varying implementations of attention mechanisms could lead to subtle numerical differences in token log-probabilities. In the context of autoregressive generation, where the selection of the next token depends on the previous sequence, a single flipped token early in the generation can cascade into a completely divergent output.

Preprint.

This potential source of variance has critical implications. Different inference engines can result in varying benchmark scores, even when the underlying model is identical. Consequently, this backend-induced variance can dethrone a model or falsely elevate a weaker one. A model's superior performance might not stem from a better architecture or improved training paradigm, but from the specific numerical properties of the inference engine used during testing. Beyond academic rankings, this instability also poses risks in real-world deployments. A model trained for safety alignment or medical accuracy on a reference implementation (e. g., HuggingFace transformers [4]) may exhibit different, potentially unsafe behaviors when deployed on a high-throughput engine like vLLM. This creates a dangerous "deployment gap" between research validation and production reality, where backend-induced discrepancies can undermine not only performance claims but also safety guarantees, potentially leading to harmful or non-compliant behavior in real-world use.

While prior work has examined sources of variability such as prompt sensitivity [5], quantization [6], and decoding strategies [7], the role of the inference backend itself has remained largely unexplored. In this work, we address this gap through a systematic empirical study and a large-scale survey of over 35,000 papers published at top machine learning venues. Our survey reveals that the specific inference stack is rarely reported, despite its widespread use in evaluation and deployment. Complementing this analysis, our controlled experiments demonstrate that the choice of inference backend alone can induce substantial variation in benchmark outcomes, shifting reported performance by up to 16 percentage points, even when model weights, prompts, and decoding parameters are held constant.

In summary, our contributions are as follows:

- **Landscape Survey.** We survey the landscape of modern inference engines and categorize them.
- **Controlled Evaluation.** We conduct a unified evaluation of open-weight models across a diverse set of popular backends (including vLLM, SGLang, and llama.cpp). We quantify their differences on standard benchmarks demonstrating that the choice of backend is a significant hyperparameter.
- **Reproducibility in ML Research.** We analyze over 35,000 recent publications from top ML conferences to quantify how frequently the inference stack is reported.
- **Root Cause Analysis.** By isolating specific optimizations, we trace backend-induced variance to two primary sources: correctable systematic defaults and optimization-induced numerical drift.

By quantifying this variability, we aim to establish new reporting standards that ensure scientific reproducibility in the era of optimized inference. To support reproducibility and future research, we will release all code and experimental artifacts upon publication.

2 Related Work

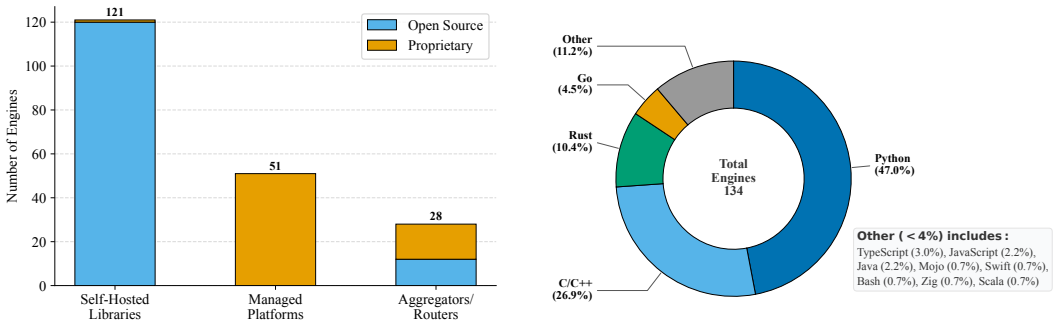
Our work connects to a broad literature on LLM inference and reproducibility.

Reproducibility for LLM evaluations. Recent studies highlight varying LLM results due to floating-point non-associativity [8], ambiguous semantic benchmarks [9], and model versioning [10]. We extend this literature by isolating the inference engine itself as a key, previously undocumented source of evaluation variance.

Assessing inference performance and determinism. Prior work evaluating inference engines primarily focuses on hardware-level optimizations, absolute inference speed, and energy efficiency across various platforms [11–13]. Other research analyzes the impact of general system design, such as caching and decoding strategies [14], expanding earlier findings on CNNs and RNNs [15], or evaluates differences between plain quantization formats [16]. Despite optimizations for determinism [17] and guidelines recommending fixed seeds and low temperatures [18], LLMs retain inherent randomness. We build on these performance-centric studies to quantify how bare backend design choices fundamentally alter generation trajectories.

3 Landscape of Modern Inference Engines

LLM inference is resource-intensive and latency-sensitive, requiring careful management of memory, parallelism, and hardware utilization. Inference engines encapsulate these optimizations behind standardized execution interfaces. To quantify the diversity of this ecosystem, we first conduct a systematic survey.



(a) Engine Categories

(b) Programming Languages

Figure 1: **Landscape of Inference Engines. (a)** Distribution of 200 surveyed inference engines across the three categories, colored to distinguish between open-source and proprietary systems. **(b)** The distribution of primary programming languages used across the open-source engines.

3.1 Survey Methodology and Scope

We define an *inference engine* as standalone software capable of loading a transformer-based model and generating completions. We surveyed the ecosystem as of January 2026, identified via GitHub, Google, and community discussions.

Inclusion & Exclusion Criteria. We include software that (1) supports open-weight or local models, (2) possesses a verifiable open-source repository or API, and (3) demonstrates active usage ($\geq$ 100 GitHub stars or active API availability). We exclude foundational libraries (e. g., PyTorch, JAX), pure training/application-layer frameworks (e. g., LangChain) and engines serving VLMs or diffusion models exclusively.

3.2 Taxonomy of Inference Systems

We classify these engines based on the level of control a user holds over the hardware and software environment. We categorize the ecosystem into three distinct categories:

Category 1: Self-Hosted Inference Libraries. The user manages the full stack/hardware. *Examples:* vLLM, llama.cpp, SGLang, HuggingFace transformers.

Category 2: Managed Inference Platforms. Abstracted compute accessed via API. *Examples:* Fireworks AI, Together AI.

Category 3: Aggregators and Routers. Unified APIs routing to third parties. *Examples:* OpenRouter, LiteLLM.

3.3 Landscape Analysis

Through our systematic search, we identified a total of **200** inference engines. Figure 1a illustrates the distribution of these systems. We find that the landscape is dominated by self-hosted libraries, which account for around 61 % of the total ecosystem. Managed platforms and aggregators comprise 26 % and 14 % respectively. This variety of available engines shows that the choice of software is a significant variable in experiments. However, we find that 44 engines (22% of the total) are inactive (no main branch commits in six months); 43 of these are self-hosted projects, with only one belonging to Category 3. This highlights that maintaining complex, low-level LLM optimizations quickly outpaces individual developer capacity. Finally, we analyzed the primary programming languages for the open-source subset of our survey (Figure 1b). While Python remains the dominant engine language, many backends are implemented in C/C++ or Rust, reflecting the necessity of low-level languages for efficient model serving.

4 Methodology

To quantify the impact of inference backends on model reproducibility, we designed a controlled experimental framework that specifically isolates the inference engine.

4.1 Experimental Scope

Inference Backends. We selected a set of five inference engines based on the popularity metrics from our landscape survey (cf. Section 3), namely: vLLM [1], SGLang [2], llama.cpp [3], LMDeploy [19, 20], and Ollama [21]. We use HuggingFace transformers [4] as the reference implementation.

Models. We selected five open-source models across different architectures and scales. *Standard:* Llama3.1 8B [22], Qwen3 4B, Qwen3 30B [23]. *Reasoning:* DeepSeek R1 Distill Qwen 7B [24], Qwen3 Thinking 30B [23]).

Benchmarks. We employ four widely adopted datasets to evaluate distinct model capabilities: GSM8K (Math) [25], GPQA Diamond (Science) [26], SimpleQA Verified (Factuality) [27], and LiveCodeBench v6 (Code) [28].

4.2 Standardization

To attribute performance differences strictly to the backend implementation, we enforce the following constraints:

Decoding Strategy. We enforce greedy decoding (temperature $= 0$) for all generations.

Model Precision. All models are loaded in FP16 (GGUF-FP16 for llama.cpp/Ollama).[1]

Prompting. To avoid discrepancies in how backends apply chat templates, we extract the Jinja2 chat template directly from the model tokenizer and apply it externally before generation.

Batch Size. Evaluations use a batch size of one to prevent batching-induced instability [8, 29].

Generation Parameters. We set the maximum number of output tokens to 2048 (Standard) and 32768 (Reasoning), with context windows of 4096 and 34816 respectively.

Seeds. To account for other sources of non-determinism, we report averages across twelve unique seeds for every configuration.

5 Evaluation & Results

All evaluations ran within a unified Docker container (Ubuntu 22.04, Python 3.12) on a single NVIDIA H100 GPU using fixed software versions for all backends (Appendix A).

5.1 Outcome Consistency

To measure macro-level agreement, we evaluate Benchmark Accuracy, Disagreement Rate, and Length Error.

Benchmark Accuracy (Table 1). Our results indicate that the inference engine is a significant source of variance, with accuracy discrepancies often exceeding several percentage points, enough to alter leaderboard ranking. Furthermore, significant outliers emerge: Llama 3.1 8B on Ollama exhibits a sharp performance drop on GSM8K, falling ten percentage points below the reference. The impact is most pronounced in reasoning models. DeepSeek R1 Distill Qwen 7B displays a 16.60 percentage-point spread between the best- and worst-performing backends on GSM8K. Ultimately, no backend perfectly matches the transformers reference across all benchmarks.

Disagreement Rate (Figure 3 in Appendix B.1). We define this as the frequency with which a backend's prediction y differs from the reference y_{ref} for the same input, regardless of ground-truth correctness: $D = \frac{1}{N} \sum \mathbb{1}(y \neq y_{ref})$. While in some settings the disagreement is small, in many cases we observe that disagreement rates consistently exceed the absolute differences in accuracy, indicating that the backend alters the model's decision boundary. For instance, the 27.37 % disagreement rate for DeepSeek R1 Distill Qwen 7B on Ollama means that out of GSM8K's 1,319 questions, the backend

[1] Not all engines support FP32 and FP16 serves as a baseline across all evaluated engines.

Table 1: **Backend Performance Variance across Benchmarks.** Performance comparison of five backends across selected models and benchmarks. Metrics are reported as accuracy (%) for GPQA and GSM8K, F1 for SimpleQA, and pass@1 for LiveCodeBench. The last column shows the difference between the maximum and minimum scores for each benchmark. Highest scores are **bold**, lowest scores are underlined.

Model	Benchmark	transformers (reference)	llama.cpp	LMDeploy	Ollama	SGLang	vLLM	Max - Min
Qwen3 4B	GPQA	35.86 ± 00.00	**38.89** ± 00.00	35.35 ± 00.00	37.88 ± 00.00	34.85 ± 00.00	34.81 ± 00.55	04.08
	GSM8K	**90.83** ± 00.00	90.45 ± 00.00	90.45 ± 00.00	90.75 ± 00.00	90.45 ± 00.00	90.47 ± 00.15	00.38
	SimpleQA	**05.91** ± 00.09	05.79 ± 00.07	05.89 ± 00.10	05.81 ± 00.09	05.52 ± 00.06	05.83 ± 00.09	00.40
	LiveCodeBench	29.71 ± 00.00	30.29 ± 00.00	30.86 ± 00.00	29.14 ± 00.00	30.29 ± 00.00	**32.33** ± 00.38	03.19
Llama 3.1 8B	GPQA	24.75 ± 00.00	23.74 ± 00.00	**25.76** ± 00.00	22.22 ± 00.00	23.23 ± 00.00	**25.76** ± 00.00	03.54
	GSM8K	84.23 ± 00.00	84.15 ± 00.00	**84.53** ± 00.00	74.30 ± 00.00	84.23 ± 00.00	83.85 ± 00.00	10.24
	SimpleQA	01.81 ± 00.12	01.97 ± 00.15	01.89 ± 00.23	**02.64** ± 00.12	01.84 ± 00.13	02.06 ± 00.11	00.83
	LiveCodeBench	17.71 ± 00.00	17.71 ± 00.00	**18.29** ± 00.00	13.14 ± 00.00	**18.29** ± 00.00	**18.29** ± 00.00	05.14
Qwen3 30B	GPQA	**45.45** ± 00.00	42.42 ± 00.00	40.40 ± 00.53	**45.45** ± 00.00	41.41 ± 00.00	40.15 ± 01.44	05.30
	GSM8K	91.74 ± 00.00	91.74 ± 00.00	91.79 ± 00.10	**91.81** ± 00.00	91.74 ± 00.00	91.72 ± 00.10	00.09
	SimpleQA	22.63 ± 00.10	22.98 ± 00.13	22.82 ± 00.11	22.91 ± 00.11	**23.00** ± 00.08	22.66 ± 00.18	00.38
	LiveCodeBench	38.29 ± 00.00	37.14 ± 00.00	36.57 ± 00.84	38.86 ± 00.00	37.71 ± 00.00	**39.14** ± 01.55	02.57
Deepseek R1 Distill Qwen 7B	GPQA	33.33 ± 00.00	31.82 ± 00.00	**35.86** ± 00.00	27.27 ± 00.00	31.31 ± 00.00	33.84 ± 00.00	08.59
	GSM8K	**78.47** ± 00.00	78.24 ± 00.00	73.54 ± 00.00	61.87 ± 00.00	78.24 ± 00.00	78.32 ± 00.00	16.60
	SimpleQA	**04.88** ± 00.14	03.88 ± 00.24	04.36 ± 00.08	04.86 ± 00.19	04.36 ± 00.18	04.74 ± 00.19	01.00
	LiveCodeBench	20.57 ± 00.00	22.86 ± 00.00	22.29 ± 00.00	19.43 ± 00.00	22.29 ± 00.00	**25.14** ± 00.00	05.71
Qwen3 Thinking 30B	GPQA	71.72 ± 00.00	70.71 ± 00.00	69.70 ± 00.00	72.22 ± 00.00	**72.73** ± 00.00	69.28 ± 02.56	03.45
	GSM8K	**94.39** ± 00.00	94.24 ± 00.00	94.01 ± 00.11	94.01 ± 00.00	94.31 ± 00.00	94.16 ± 00.15	00.38
	SimpleQA	**28.71** ± 00.20	28.62 ± 00.28	27.65 ± 00.32	27.33 ± 00.34	27.99 ± 00.24	28.43 ± 00.36	01.37
	LiveCodeBench	**61.71** ± 00.00	57.71 ± 00.00	60.52 ± 01.47	56.00 ± 00.00	61.14 ± 00.00	60.33 ± 01.15	05.71

generates a different final answer than the transformers reference over 360 times. For reasoning models evaluated on GPQA and LiveCodeBench, this divergence is similarly pronounced.

Length Error (Figure 4 in Appendix B.2). We detect systematic biases in verbosity via the signed difference (bias) and absolute difference (magnitude) in output token counts. Standard models usually stay near a stable "ideal zone" of $\leq$25 absolute tokens, a threshold indicating the text length varies by no more than 1-2 sentences, preserving structural consistency. However, the lengths for GPQA and LiveCodeBench differ significantly for all tested models. Even more distinct are the differences for reasoning models where the length differs significantly across all the tested datasets. For instance, the Ollama backend produces DeepSeek R1 outputs that are, on average, over 9,000 tokens shorter than the reference on GPQA, fundamentally altering the chain-of-thought process.

5.2 Token-Level Divergence

To pinpoint *when* the generation differences occur, we define the ***Divergence Index*** as the position k of the first mismatched token between y and y_{ref}. We report both the averaged raw index and a Normalized Divergence Score, calculated as $S = \frac{k}{max(|y|,|y_{ref}|)} \in [0, 1]$, where 1.0 indicates a perfect match, while values approaching 0.0 denote immediate divergence. Analyzing this index (Figures 5–8 in Appendix B.3), we observe that reasoning models consistently diverge from the reference generation, much earlier than standard architectures. This is amplified on difficult benchmarks like GPQA, where, for example, Llama 3.1 served via Ollama diverges as early as the 12th output token.

5.3 Numerical Precision

To evaluate floating-point stability on the matching prefix (tokens generated prior to divergence), we compute two probability metrics. ***Logprob Root Mean Squared Error (RMSE)*** of the top-1 token measures absolute floating-point drift, while ***Top-5 Token Jaccard Similarity*** assesses preservation of the distribution's overall shape, even when the top-selected token remains identical. The LogProb RMSE quantifies the floating-point variance of the top token. Even small differences are consequential; an RMSE of 0.1 corresponds to a roughly 10 % relative change in the raw probability assignment ($e^{0.1} \approx 1.1$), which is sufficient to flip the greedy selection. While most backends maintain similar precision with the reference (RMSE$<$0.01), we observe that reasoning models systematically exhibit higher drift and specific configurations display distinct error spikes. Despite these numerical fluctuations, the Top-5 Token Jaccard similarity remains high across most configurations, suggesting that the general *shape* of the probability distribution remains intact, breaking down only in the most extreme failure cases.

5.4 Robustness and Real-World Implications

To ensure our findings generalize beyond our strictly standardized setup, we conducted targeted ablation studies (full details in Appendix C).

Safety implications (Section C.1). Using JailbreakBench [30], we found that DeepSeek R1's vulnerability to adversarial prompts fluctuated by 8.9 percentage points based solely on the inference engine.

Batching (Section C.2). Evaluating batched generation (batch size = 4) revealed that performance differences persist, and slight numerical shifts within vLLM and SGLang verify that batching actively influences generation.

Hardware Independence (Section C.3). Evaluating on NVIDIA L40 GPUs yielded consistent, slightly increased backend-induced variance (up to 17.2% Max-Min difference), confirming divergence is rooted in software implementations rather than specific GPU architectures.

Stochastic Sampling (Section C.4). Relaxing our greedy decoding constraint to use temperature sampling ($T = 0.7$) preserves the backend-induced variance, proving this variance is not an artifact of greedy decoding.

6 Root Cause Analysis of Backend Variance

The results in Section 5 demonstrate that the choice of inference backend can shift benchmark performance by up to 16.6 percentage points. To understand the origin of this variance, we isolate specific optimizations and trace the model execution pipeline. For these targeted ablations, we evaluate Llama 3.1 8B and DeepSeek R1 on GSM8K. We categorize root causes into two distinct groups: (1) systematic, but correctable, engine defaults that explain the massive outliers observed in Table 1, and (2) optimization-induced numerical drift that inherently alters the mathematical execution of the model (detailed ablation results are provided in Appendix D).

6.1 Systematic Engine Defaults

The most extreme divergences, such as DeepSeek R1 scoring only 61.87 % on Ollama compared to 78.47 % on the transformers reference, are caused by engine-specific defaults applied prior to generation.

Hidden Prompt Mutation: Modern chat templates are highly sensitive to formatting. Even when passing the exact Jinja2 chat templates to Ollama via the `raw=True` parameter, the engine forcefully prepends a Begin-Of-Sequence (BOS) token. Similarly, LMDeploy automatically injects a BOS token specifically for DeepSeek R1. Removing these BOS tokens to strictly pass the raw prompt recovers 4.7 percentage points for DeepSeek on LMDeploy, and improves accuracy by 8.34 and 7.35 points for Llama 3.1 and DeepSeek on Ollama, respectively.

Default Penalty Parameters: Ollama enforces a hidden default repetition penalty of 1.1, severely degrading chain-of-thought generation. Disabling this penalty (setting to 1.0) increases DeepSeek R1's accuracy by 11.67 percentage points and Llama 3.1's by 1.67, effectively closing the most extreme performance gaps observed in Table 1.

6.2 Optimization-Induced Numerical Variance

Even after completely aligning all defaults and pre-processing steps, smaller, random fluctuations persist. This is driven by high-throughput optimizations that inherently alter the mathematical execution of the model.

Prefix Caching & CUDA Graphs: Engines like vLLM, SGLang, llama.cpp, and Ollama enable prefix caching by default. Processing a prompt in fragmented chunks fundamentally alters the reduction trees. Disabling prefix caching caused random accuracy shifts (e. g., +0.46 % on Llama 3.1 for vLLM, and -0.47 % on DeepSeek for Ollama). Similarly, disabling CUDA graphs shifted performance across engines by up to +0.15 %.

Kernel-Level Tie-Breaking & Accumulation Precision: When tokens share identical logit values in FP16, PyTorch deterministically selects the lowest ID. Conversely, LMDeploy computes greedy decoding via a multi-threaded Top-K kernel ($K = 1$) creating a hardware race condition that picks

arbitrarily. Patching this kernel to match PyTorch tie-breaking caused random fluctuations for Llama 3.1 (-0.45 %) and DeepSeek (+0.06 %). Furthermore, llama.cpp and Ollama accumulate intermediate matrix multiplications in FP32. This preserves higher precision but inherently guarantees rounding differences compared to pure FP16 execution.

Layer-wise Error Propagation: To verify if a specific architectural layer was responsible for these divergences, we implemented a custom tracking pipeline to measure similarity at every layer boundary during the forward pass. We found no single failing layer. Instead, slight numerical drifts caused by custom kernels compound continuously across the model's depth, eventually cascading into different Top-1 token predictions.

7 Inference Reproducibility in ML Research

Given the substantial impact of backend choice on benchmark performance, we conducted a systematic survey of recent ML publications to contextualize this variance and measure the prevalence of inference stack reporting.

7.1 Methodology

We analyze papers published between 2023 and 2025 in top-tier ML and NLP conferences (NeurIPS, ICML, ICLR, ACL, and EMNLP). To categorize reproducibility artifacts, we classified papers into four Reproducibility Tiers: **Tier 0** (neither a code repository nor inference backend is mentioned), **Tier 1** (backend is documented textually, but no code is provided), **Tier 2** (code is available, but specific environment dependencies are absent), and **Tier 3** (code is provided alongside explicit dependency management, e. g., `requirements.txt`)

These tiers categorize the level of reproducibility artifacts, distinguishing between verifiable and machine-readable environment definitions (Tier 3) and ambiguous textual descriptions that fail to capture implementation details (Tier 1).

Filtering and Extraction Pipeline. A keyword pre-filter first flagged potentially relevant papers. An LLM-as-a-judge then strictly isolated 9,018 papers running local LLM inference. For these confirmed papers, we ran two parallel extraction processes using the LLM judge: a *Text Analysis* scan to determine if specific inference engines were explicitly named, and a *Code Analysis* scan to extract repository URLs. The full methodology, including the exact keyword list, pipeline logic, and complete LLM prompts, is detailed in Appendix E.

Finally, via the GitHub API, we checked validated repositories for dependency specifications (e. g., `requirements.txt`; see Appendix E.4 for the full list of file patterns) to distinguish between Tier 2 and 3. While dependency files do not guarantee strict reproducibility (e. g., unpinned versions), their presence serves as a proxy for reproducibility intent, distinguishing raw code dumps from deliberate standardization efforts.

7.2 Results

Setup. We extracted text from PDF files using `pymupdf` and utilized Qwen3-235B-A22B-Instruct-2507-AWQ [31] as the LLM judge.[2]

We present the distribution of reproducibility tiers across the 9,018 relevant papers in Figure 2.

Prevalence of Code Sharing. Among the 9,018 papers that we extracted after the filtering (Step 1), 75 % (6,761) include a URL to a code repository. The distribution of hosting platforms is heavily skewed toward GitHub with 90.2 % (6,098), with minor representation from github.io with 3.7 % (247), HuggingFace with 1.1 % (74) and other platforms with 5.0 % (342).

Artifact Availability. To automate verification, we restricted our analysis to GitHub and attempted to access the 6,098 identified repositories. However, we found that nearly 8 % (460) of these repositories were either deleted, empty, or contained only documentation (License/Readme) with no source code.

Backend Reporting Frequency. For papers without code artifacts, we analyzed how frequently the inference stacks were disclosed. Among the 2,257 papers offering no code, 820 (36 %) explicitly

[2]Using the vLLM backend (version 0.13.0) with greedy decoding

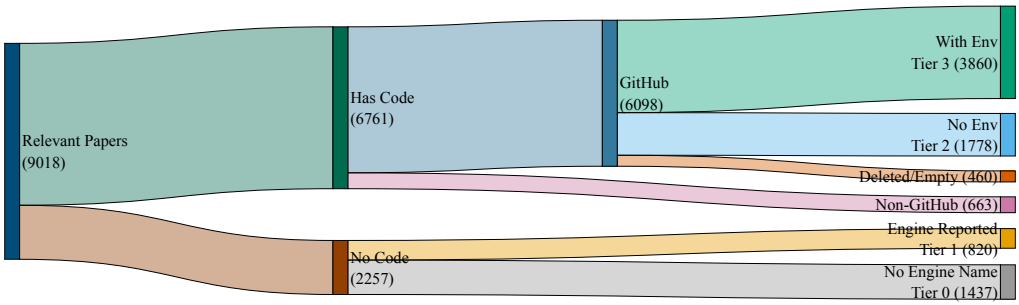


Figure 2: **Prevalence of Reproducibility Artifacts in ML Research.** A breakdown of 9,018 relevant papers categorized by their reproducibility tier.

named the backend. This subset is dominated by transformers (322; 39 %) and vLLM (150; 18 %), followed by custom PyTorch implementations (98; 12 %). We extended this analysis to the 460 papers with empty or deleted repositories, and found a similar trend: only 180 (39.1 %) reported the engine textually. In this group, reliance on transformers was even higher (90; 50 %), with vLLM and custom PyTorch both at 14 % (25).

Environment Reproducibility. We found that only 3,860 of the accessible GitHub repositories (approx. 63 %) contained explicit dependency specifications (e. g., `requirements.txt`, `Dockerfile`). This leaves over a third (1,778) of released repositories without a defined execution environment. For these papers, it becomes impossible to reconstruct the specific inference stack used during evaluation.

Manual Verification. To ensure the reliability of our automated pipeline, we conducted a manual verification on a random subset of papers at each filtering and classification stage (details in Appendix F.

Ultimately, these findings confirm that while code sharing is becoming standard, the inference backend remains a largely undocumented source of experimental variance.

8 Discussion

The findings above demonstrate that inference backends are not a trivial implementation detail, but an active, influential element of the LLM evaluation process.

Root Causes and Preventability. Backend divergence stems from preventable defaults and fundamental hardware optimizations. While users can standardize overrides like repetition penalties, optimization-induced variance (e. g., non-associative reductions, hardware race conditions) is deeply tied to system architectures. Completely mitigating this variance is impractical, as these optimizations are required for high-throughput serving.

Implications for Leaderboards. Backend choice alone can shift performance by up to 16 percentage points. While the most extreme deviations stem from correctable hidden defaults, researchers are largely unaware of them. Even after correcting these, optimization-induced numerical drift still shifts scores by margins larger than the fractions of a percent frequently used to claim SOTA. This suggests that comparative benchmarking is fundamentally flawed, as victories may reflect backend artifacts rather than architectural superiority.

Security & Robustness. Backend variance also introduces a critical "deployment gap." An identical model's vulnerability to jailbreaks fluctuates by nearly 9 % simply by switching the backend, highlighting backend selection as a previously unrecognized security variable.

8.1 Recommendations

Our findings suggest that inference reproducibility is a systemic issue and thus we propose the following recommendations for researchers, evaluators, and system developers.

Researchers and Practitioners. The following guidelines aim to improve experimental rigor and ensure more reliable and reproducible findings in practice. ***Reporting Standards:*** We advocate for publishing exact environment specifications (e. g., Docker containers), or at a minimum, the specific backend library and version. ***Account for Non-Determinism:*** Our results show that optimized backends can exhibit variance even with greedy decoding. Consequently, researchers should avoid relying on single evaluation runs. We recommend averaging results across multiple seeds. ***Ensure Fair Comparisons:*** If the inference backend of a state-of-the-art method cannot be determined due to missing documentation, the experiments should be rerun in a comparable setting, using the same inference backend for all experiments.

Benchmarks and Leaderboards. To ensure meaningful comparisons and trustworthy rankings, evaluation platforms must adopt stricter reporting and measurement practices. ***Standardize the Inference Stack:*** Leaderboards must explicitly state the engine and configurations used, as backend-induced variance can exceed margins separating SOTA models. ***Quantify Uncertainty:*** Where feasible, evaluations should report a "backend confidence interval" by testing on both a reference implementation and a high-throughput engine.

Inference Engine Developers & Providers. System-level improvements are necessary to enable reproducibility guarantees and better transparency for downstream users. ***Expose System Fingerprints:*** For API-based inference, fixing a random seed is insufficient for reproducibility. Providers should include a system fingerprint in response metadata to allow users to track backend changes over time. ***Deterministic Modes:*** We encourage developers to implement "strict reproducibility" flags. While high-performance non-deterministic kernels are essential for production, a slower, deterministic execution path is necessary for scientific debugging and validation.

8.2 Limitations

To isolate the numerical influence, we enforced a controlled environment. While necessary for fair comparison, this setup does not fully reflect production environment. Consequently, the divergence we observe likely represents a lower bound; in normal or high-load usage scenarios where advanced optimizations are active, the variance may be even more pronounced.

We utilized greedy decoding to minimize sampling randomness. However, this strategy is not optimal for all architectures, particularly reasoning models (e. g., DeepSeek R1). In some instances, we observed that greedy decoding led to repetitions or degradation in reasoning chains, potentially skewing the metrics for those specific models. Therefore, the individual benchmarks may reach better results if optimized for the respective setup. However, in this paper, we were interested in the relative differences of runs with varying inference backends.

Finally, ablating every custom kernel is infeasible, and fully disabling these features to achieve perfect determinism is often impossible without abandoning the engine entirely.

9 Conclusion

Our results reveal that inference backends are not a benign implementation detail, but an active and influential component of the LLM evaluation pipeline. Across models, benchmarks, and metrics, we find that backend-induced numerical differences can propagate into divergent generations, altered decision boundaries, and benchmark score shifts large enough to affect model rankings. These effects are particularly pronounced for reasoning-oriented models, where early token-level divergence can fundamentally reshape the generated chain of thought. We demonstrate that this behavior is driven by a combination of hidden engine defaults and compounding numerical drift caused by essential high-performance features. At the same time, our large-scale survey of recent ML publications shows that this source of variability is rarely documented, despite the widespread use of optimized inference engines in both research and deployment.

While the community has developed careful conventions around datasets, decoding strategies, and random seeds, the inference stack itself remains largely invisible, even though it can dominate other sources of variance. To address this gap, we advocate treating the inference backend as a first-class experimental parameter: explicitly reporting backend implementations, repeated evaluations, and supporting deterministic execution modes for scientific validation.

Acknowledgments and Disclosure of Funding

This work was supported by the Helmholtz Association's Initiative and Networking Fund on the HAICORE@FZJ partition and by the German Federal Ministry of Education and Research under the grant AIgenCY (16KIS2012) and SisWiss (16KIS2330). Moreover, this work was supported by the LCIS center VW-Vorab-2025, ZN4704 11-76251-2055, as well as the Daimler and Benz Foundation under the grant Ladenburger Kolleg, Project KonCheck.

References

[1] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the Symposium on Operating Systems Principles*, 2023.

[2] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[3] ggml org. llama.cpp. `https://github.com/ggml-org/llama.cpp`, 2023.

[4] Thomas Wolf et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics (ACL), 2020.

[5] Amirhossein Razavi, Mina Soltangheis, Negar Arabzadeh, Sara Salamat, Morteza Zihayat, and Ebrahim Bagheri. Benchmarking prompt sensitivity in large language models. In *Advances in Information Retrieval: European Conference on Information Retrieval (ECIR)*, 2025.

[6] Eldar Kurtic, Alexandre Noll Marques, Shubhra Pandit, Mark Kurtz, and Dan Alistarh. "give me BF16 or give me death"? accuracy-performance trade-offs in LLM quantization. In *Association for Computational Linguistics (ACL)*, 2025.

[7] Chufan Shi, Haoran Yang, Deng Cai, Zhisong Zhang, Yifan Wang, Yujiu Yang, and Wai Lam. A thorough examination of decoding methods in the era of LLMs. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

[8] Jiayi Yuan, Hao Li, Xinheng Ding, Wenya Xie, Yu-Jhe Li, Wentian Zhao, Kun Wan, Jing Shi, Xia Hu, and Zirui Liu. Understanding and mitigating numerical sources of nondeterminism in LLM inference. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[9] Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sid Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin A. Wang, Genta Indra Winata, Franccois Yvon, and Andy Zou. Lessons from the trenches on reproducible evaluation of language models. *arXiv preprint arXiv:2405.14782*, 2024.

[10] Jonathan Evertz, Niklas Risse, Nicolai Neuer, Andreas Müller, Philipp Normann, Gaetano Sapia, Srishti Gupta, David Pape, Soumya Shaw, Devansh Srivastav, Christian Wressnegger, Erwin Quiring, Thorsten Eisenhofer, Daniel Arp, and Lea Schönherr. Chasing shadows: Pitfalls in llm security research. In *Symposium on Network and Distributed System Security (NDSS)*, 2026.

[11] Krishna Teja Chitty-Venkata, Siddhisanket Raskar, Bharat Kale, Farah Ferdaus, Aditya Tanikanti, Ken Raffenetti, Valerie Taylor, Murali Emani, and Venkatram Vishwanath. Llm-inference-bench: Inference benchmarking of large language models on ai accelerators. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024.

[12] Jinhao Li, Jiaming Xu, Shan Huang, Yonghua Chen, Wen Li, Jun Liu, Yaoxiu Lian, Jiayi Pan, Li Ding, Hao Zhou, et al. Large language model inference acceleration: A comprehensive hardware perspective. *arXiv preprint arXiv:2410.04466*, 2024.

[13] Sihyeong Park, Sungryeol Jeon, Chaelyn Lee, Seokhun Jeon, Byung-Soo Kim, and Jemin Lee. A survey on inference engines for large language models: Perspectives on optimization and efficiency. *arXiv preprint arXiv:2505.01658*, 2025.

[14] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Hongyi Jin, Tianqi Chen, and Zhihao Jia. Towards efficient generative large language model serving: A survey from algorithms to systems. *ACM Comput. Surv.*, 2025.

[15] Bin Xu, Ayan Banerjee, and Sandeep Gupta. Hardware acceleration for neural networks: A comprehensive survey. *arXiv preprint arXiv:2512.23914*, 2026.

[16] Yifei Wang, Tianlin Li, Xiaohan Zhang, Xiaoyu Zhang, Wei Ma, Mingfei Cheng, and Li Pan. Hidden reliability risks in large language models: Systematic identification of precision-induced output disagreements. *arXiv preprint arXiv:2604.19790*, 2026.

[17] Lukas Heumos, Philipp Ehmele, Luis Kuhn Cuellar, Kevin Menden, Edmund Miller, Steffen Lemke, Gisela Gabernet, and Sven Nahnsen. mlf-core: a framework for deterministic machine learning. *Bioinformatics*, 2023.

[18] Robert E Blackwell, Jon Barry, and Anthony G. Cohn. Towards reproducible llm evaluation: Quantifying uncertainty in llm benchmark scores. *arXiv preprint arXiv:2410.03492*, 2024.

[19] LMDeploy Contributors. Lmdeploy: A toolkit for compressing, deploying, and serving llm. `https://github.com/InternLM/lmdeploy`, 2023.

[20] Li Zhang, Youhe Jiang, Guoliang He, Xin Chen, Han Lv, Qian Yao, Fangcheng Fu, and Kai Chen. Efficient mixed-precision large language model inference with turbomind. *arXiv preprint arXiv:2508.15601*, 2025.

[21] Ollama. ollama. `https://ollama.com/`, 2023.

[22] Aaron Grattafiori et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[23] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[24] Daya Guo et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature 645*, 2025.

[25] Karl Cobbe, Vineet Kosaraju, Mo Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[26] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *Conference on Language Modeling*, 2024.

[27] Lukas Haas, Gal Yona, Giovanni D'Antonio, Sasha Goldshtein, and Dipanjan Das. Simpleqa verified: A reliable factuality benchmark to measure parametric knowledge. *arXiv preprint arXiv:2509.07968*, 2025.

[28] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025.

[29] Horace He. Defeating nondeterminism in llm inference, 2025. URL https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/.

[30] Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwag, Edgar Dobriban, Nicolas Flammarion, George J. Pappas, Florian Tramèr, Hamed Hassani, and Eric Wong. Jailbreakbench: An open robustness benchmark for jailbreaking large language models. In *NeurIPS Workshop Datasets and Benchmarks Track*, 2024.

[31] AIDXteam. Qwen3-235b-a22b-instruct-2507-awq, 2025. URL https://huggingface.co/AIDXteam/Qwen3-235B-A22B-Instruct-2507-AWQ.

[32] OpenAI. Gpt-4o-mini. https://developers.openai.com/api/docs/models/gpt-4o-mini, 2024.

A Backend Versions

Table 2 details the specific versions of the inference backends and reference libraries utilized throughout all controlled experiments in Section 5. We enforced these fixed versions across all evaluation runs to ensure that any observed numerical variance is strictly attributable to the architectural differences between the backends.

Table 2: **Software Versioning.** Specific software versions for the inference backends and reference libraries used in our evaluation.

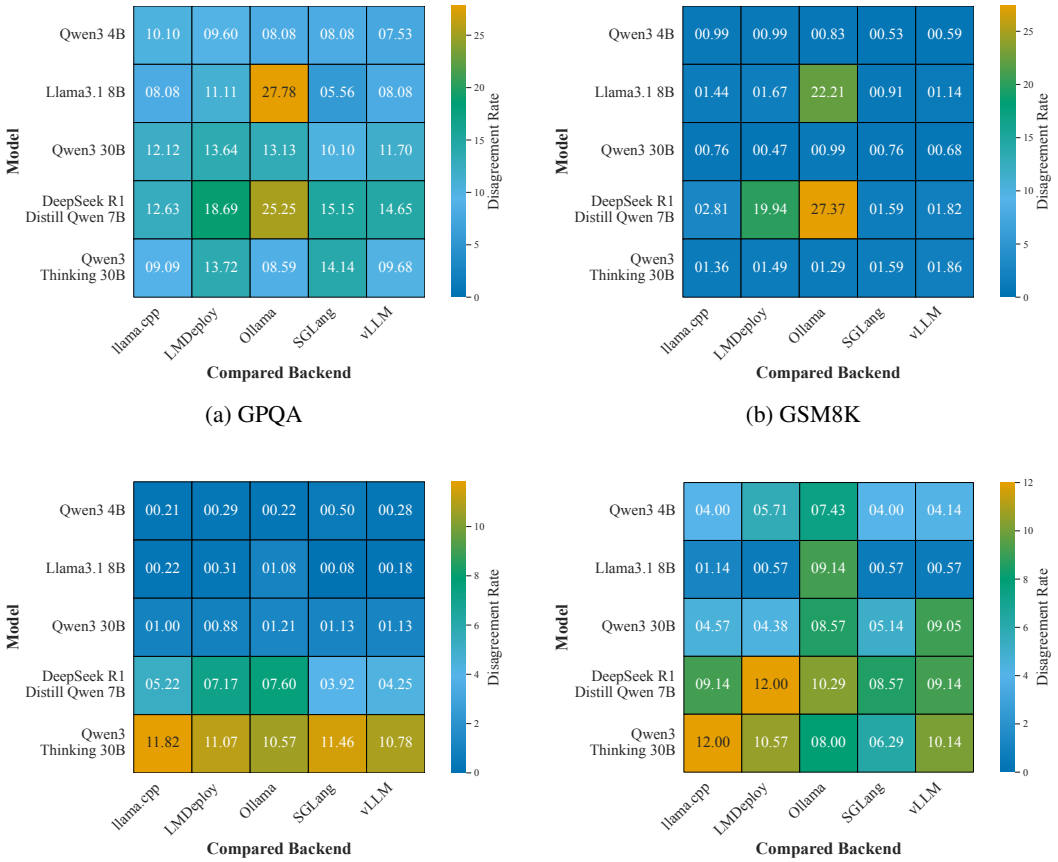
Library / Backend	**Version**
transformers	4.57.0
vLLM	0.10.2
SGLang	0.5.2
LMDeploy	0.10.1
llama_cpp_python	0.3.16
ollama	0.13.5 (python 0.6.1)

B Additional Metrics

This section provides detailed visualizations and breakdowns for the evaluation metrics introduced in Section 5. By observing these metrics across individual datasets and models, we can better understand how backend-induced variance disproportionately affects specific architectures (e. g., reasoning models) and tasks.

B.1 Disagreement Rate

As defined in Section 5.1, the Disagreement Rate measures the absolute frequency with which a backend produces a different final prediction than the transformers reference. Figure 3 illustrates these rates across our evaluated benchmarks. While aggregate scores (Table 1) might appear stable in certain configurations, the disagreement rate reveals underlying instability. Two backends can achieve the exact same overall accuracy score while correctly answering a completely different subset of questions, indicating that the backend numerical variance actively shifts the model's decision boundary.



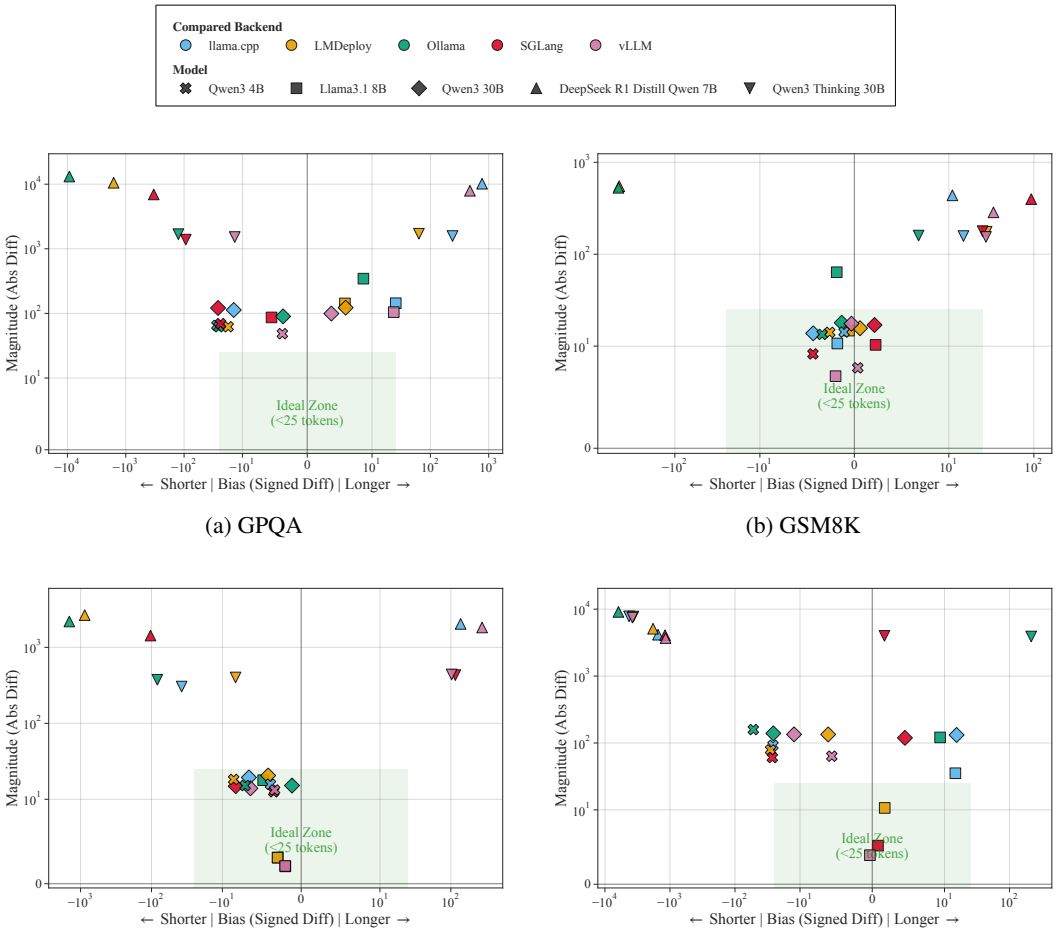
(a) GPQA (b) GSM8K

(c) SimpleQA (d) LiveCodeBench

Figure 3: **Output Disagreement Rates.** The frequency with which each backend's prediction differs from the transformers reference implementation for the same input. Higher values indicate a larger disagreement between the two backends.

B.2 Length Error

Beyond final accuracy, we analyze structural deviations in the generated responses by measuring Output Length consistency (Figure 4). We plot both the Signed Difference (Bias) on the X-axis and the Absolute Difference (Magnitude) on the Y-axis. The signed difference reveals whether a backend has a systematic bias toward verbosity (producing consistently longer or shorter sequences), while the absolute difference captures the scale of the deviation, preventing positive and negative length differences from canceling each other out. We define an "Ideal Zone" of ± 25 tokens (roughly 1-2 sentences), where the structural integrity of the answer is largely preserved. As shown, reasoning models frequently lie outside this zone, experiencing massive shifts in generation length that fundamentally alter their chain-of-thought process.



(a) GPQA

(b) GSM8K

(c) SimpleQA

(d) LiveCodeBench

Figure 4: **Analysis of Output Length Consistency against the transformers Reference.** This scatter plot visualizes the deviation in generation length for various backends. The **X-axis (Bias)** represents the average *Signed Difference*, where negative values indicate the backend generated fewer tokens than the reference (shorter), and positive values indicate more tokens (longer). The **Y-axis (Magnitude)** represents the average *Absolute Difference*, showing the total scale of the deviation regardless of direction. The shaded green region ("Ideal Zone") marks acceptable variance ($\pm$25 tokens), roughly equating to a 1-2 sentence difference.

B.3 Token Divergence

To pinpoint exactly *when* the generations begin to differ, we calculate the Token-Level Divergence Index. Figures 5 through 8 visualize the Normalized Divergence Score alongside the absolute token position of the first mismatch (labels above the bars). A normalized score closer to 1.0 indicates that the generations remain identical for the majority of the sequence, whereas lower scores indicate early divergence. Our results highlight that difficult benchmarks (such as GPQA) cause models to diverge much earlier in the generation process.

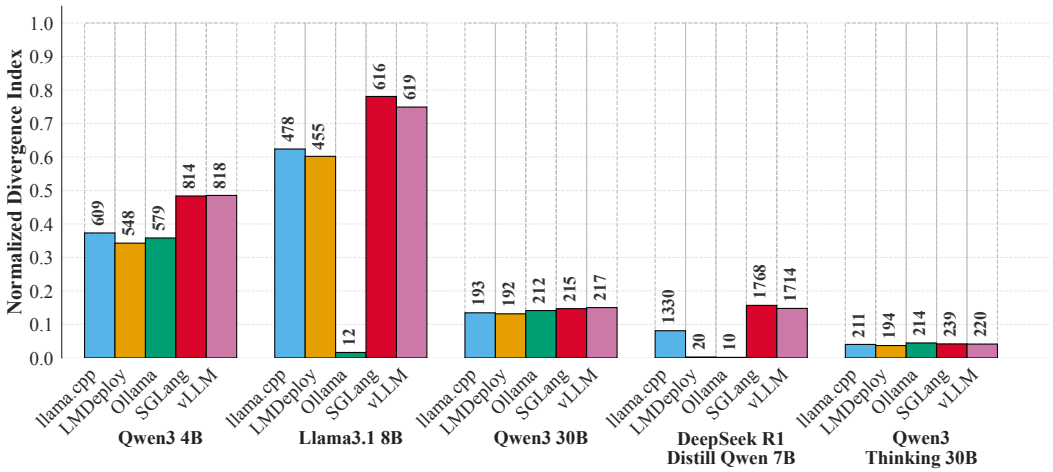


Figure 5: **Token-Level Divergence Analysis (GPQA).** Normalized divergence scores relative to the transformers reference. Larger values indicate high similarity (divergence happens late), while smaller values indicate early divergence. The labels above the bars indicate the average token position at which the generation first differs from the reference sequence.

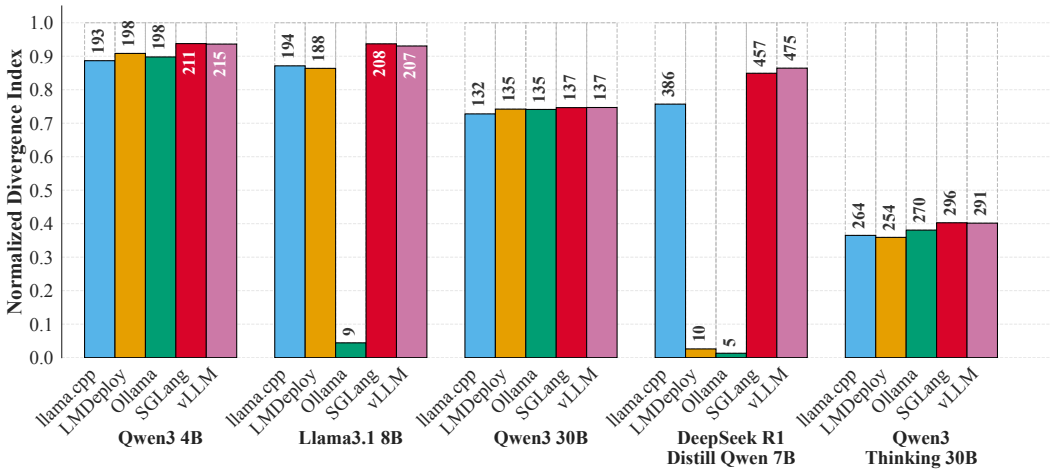


Figure 6: **Token-Level Divergence Analysis (GSM8K).** Normalized divergence scores relative to the transformers reference. Larger values indicate high similarity (divergence happens late), while smaller values indicate early divergence. The labels above the bars indicate the average token position at which the generation first differs from the reference sequence.

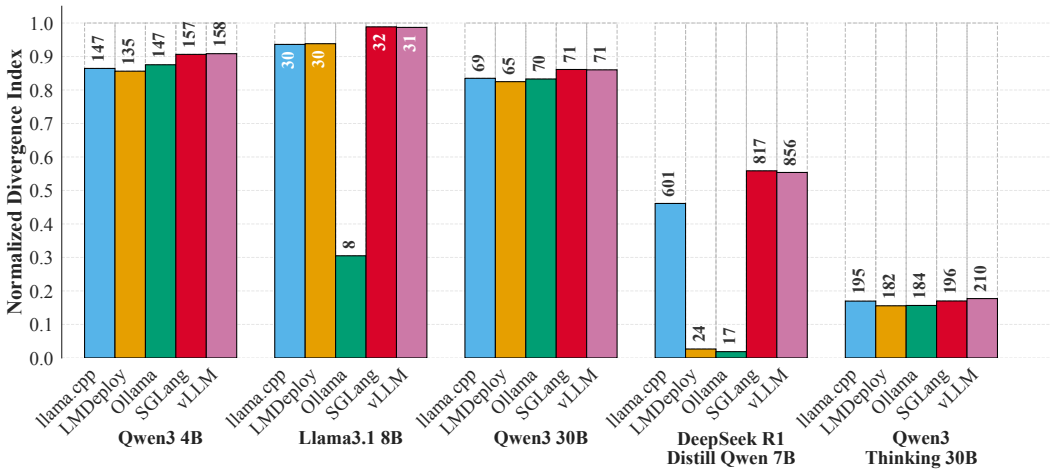


Figure 7: **Token-Level Divergence Analysis (SimpleQA).** Normalized divergence scores relative to the transformers reference. Larger values indicate high similarity (divergence happens late), while smaller values indicate early divergence. The labels above the bars indicate the average token position at which the generation first differs from the reference sequence.

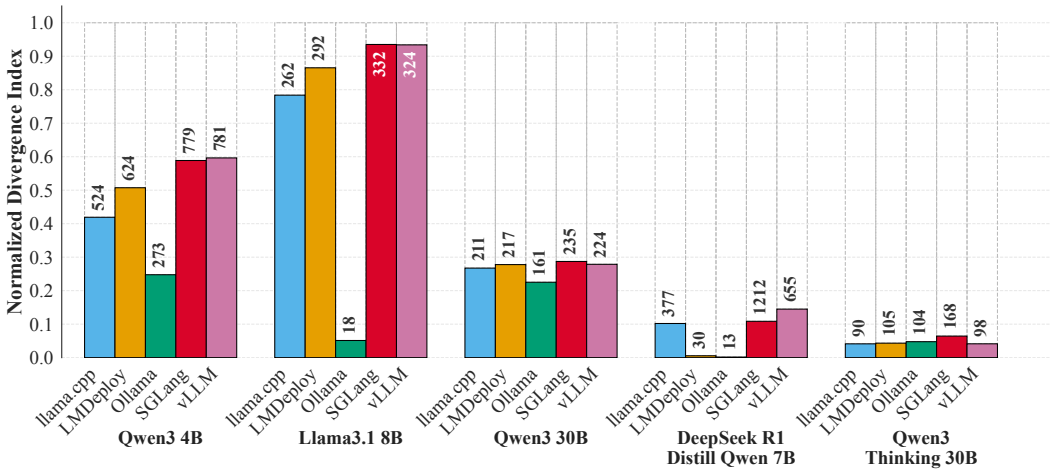
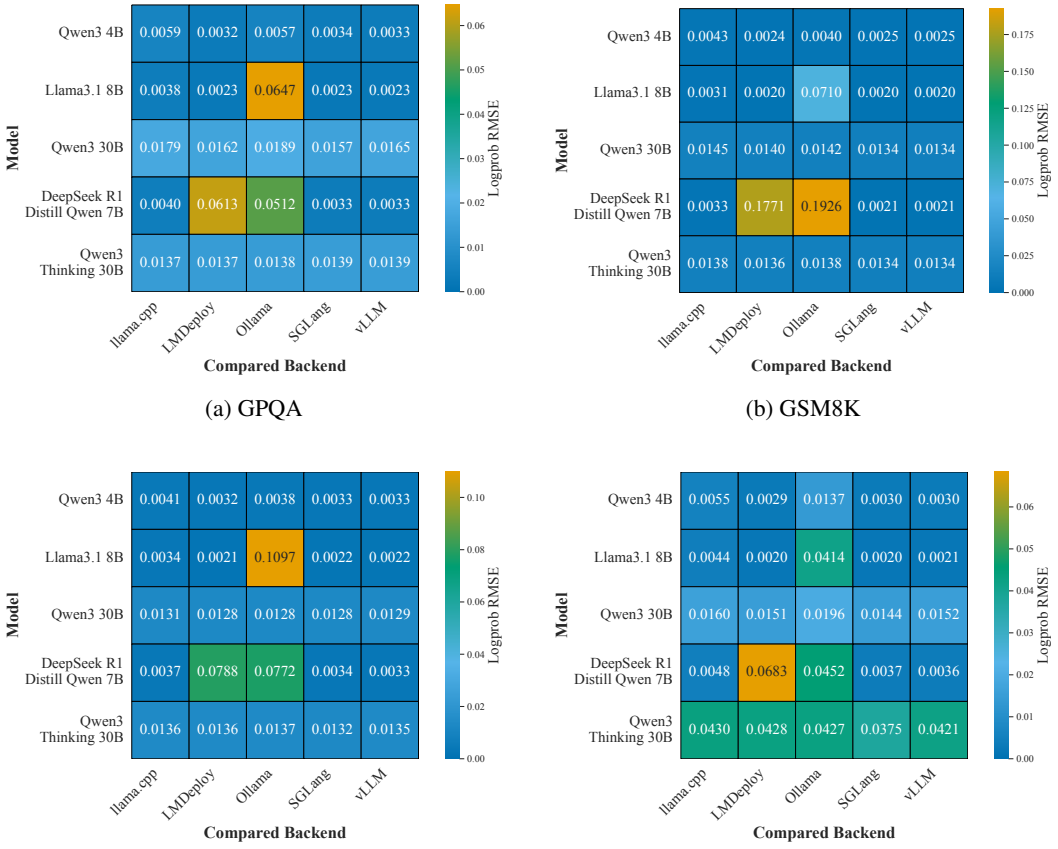


Figure 8: **Token-Level Divergence Analysis (LiveCodeBench).** Normalized divergence scores relative to the transformers reference. Larger values indicate high similarity (divergence happens late), while smaller values indicate early divergence. The labels above the bars indicate the average token position at which the generation first differs from the reference sequence.

B.4 LogProb Error

While the previous metrics evaluate the final generated text, we also investigate the underlying floating-point stability prior to any generation mismatch. Figure 9 presents the LogProb RMSE for the top-1 token calculated on the matching prefix (the tokens generated before the sequences diverge). These heatmaps confirm that numerical drift is present at the logit level even when the discrete greedy token selections remain identical. Reasoning models consistently exhibit higher baseline drift compared to standard models, foreshadowing their higher rates of downstream token divergence.



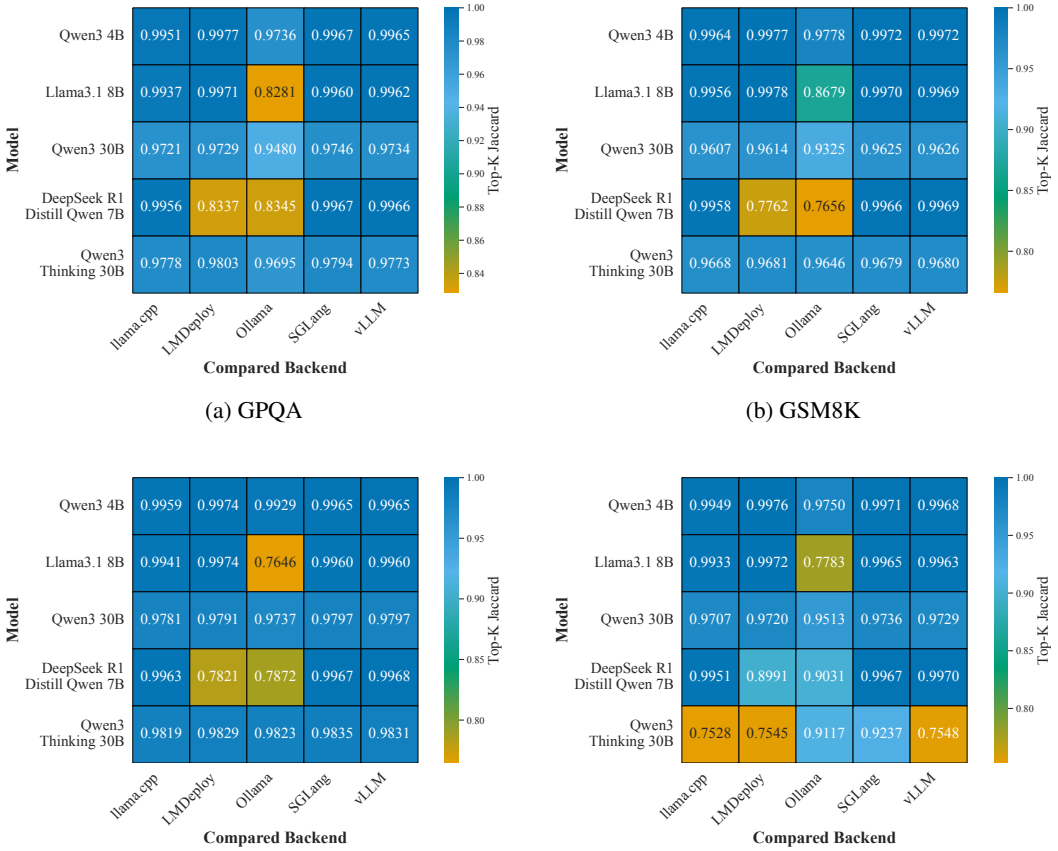
(a) GPQA (b) GSM8K

(c) SimpleQA (d) LiveCodeBench

Figure 9: **Numerical Precision (LogProb RMSE).** Root Mean Squared Error (RMSE) of the top-1 token log-probabilities compared to the transformers reference. In an ideal, deterministic setting, we expect an RMSE of exactly 0.0, indicating identical confidence in token selection. Higher values demonstrate numerical drift caused by the backend. This indicates that the underlying probability distribution is shifting, which can eventually cascade into divergent token selections.

B.5 Top-5 Token Jaccard Similarity

To determine if the backend variance shifts the entire probability distribution or just the absolute top prediction, we calculate the Top-5 Token Jaccard Similarity (Figure 10). This metric measures the overlap of the top-5 candidate tokens between the backend and the reference implementation. While most standard models maintain high similarity, sharp drops in this metrics (particularly in reasoning models) indicate that numerical instability is occasionally severe enough to completely reshape the distribution, pushing entire new tokens in the top-5 candidate pool.



(a) GPQA

(b) GSM8K

(c) SimpleQA

(d) LiveCodeBench

Figure 10: **Distribution Stability (Top-5 Jaccard Similarity).** The overlap of the top-5 most probable tokens between the backend and the transformers reference. We expect a similarity score of 1.0, meaning the set of the top-5 token candidates is perfectly identical across both implementations. Lower values indicate that the backend's numerical deviations fundamentally alter the model's candidate pool, bringing entirely different tokens into the top-5 predictions.

C Ablation Studies

To validate the robustness of our findings and address the real-world implications of backend-induced variance, we conducted four targeted ablation studies. Unless otherwise specified, all ablations evaluate the Llama 3.1 8B and DeepSeek R1 Distill Qwen 7B models across 12 seeds.

C.1 Safety and Jailbreak Vulnerability

To assess whether backend-induced variance impacts model alignment, we evaluated vulnerability to adversarial prompts using JailbreakBench. The metric reported is Attack Success Rate (ASR %), where lower is better. As shown in Table 3, while Llama 3.1 8B is consistently robust, DeepSeek R1's vulnerability fluctuates by nearly 9% depending on the inference engine. This demonstrates that

identical model weights can exhibit noticeable distinct safety profiles when deployed on different inference engines.

Table 3: **Ablation: Safety Vulnerability.** Attack Success Rate (%) on JailbreakBench (Lower is more robust).

Model	transformers	llama.cpp	LMDeploy	Ollama	SGLang	vLLM	Max - Min
Llama 3.1 8B	02.00 ± 00.00	02.00 ± 00.00	02.00 ± 00.00	02.00 ± 00.00	02.00 ± 00.00	02.00 ± 00.00	00.00
DeepSeek R1 7B	48.90 ± 00.80	48.20 ± 00.70	47.30 ± 00.70	**42.90** ± 00.30	47.10 ± 01.30	<u>51.80</u> ± 01.10	08.90

C.2 Batching

Inference engines like vLLM and SGLang are specifically designed for high-throughput, concurrent serving environments, relying heavily on mechanisms like continuous batching. To ensure our findings reflect these real-world usage patterns, we ran an ablation using a batch size of 4 on the GSM8K benchmark. Because llama.cpp and Ollama do not natively support batched generation, we report their batch size=1 numbers for comparison. As shown in Table 4, backend variance persists under batched generation. Furthermore, comparing these results to Table 1 reveals slight numerical shifts within vLLM and SGLang themselves, confirming that batching actively influences the final generation.

Table 4: **Ablation: Batched Generation (Batch Size = 4).** Accuracy (%) on GSM8K.

Model	transformers	llama.cpp*	LMDeploy	Ollama*	SGLang	vLLM	Max - Min
Llama 3.1 8B	84.00 ± 00.00	84.15 ± 00.00	84.15 ± 00.00	<u>74.30</u> ± 00.00	**84.22** ± 00.02	84.00 ± 00.09	09.93
DeepSeek R1 7B	**78.62** ± 00.00	78.24 ± 00.00	74.07 ± 00.00	<u>61.87</u> ± 00.00	**78.62** ± 00.00	78.38 ± 00.20	16.76

**Evaluated at batch size=1 due to lack of native batched generation support.*

C.3 Hardware Independence

To verify that our observations are not an artifact of our specific NVIDIA H100 (Hopper) setup, we re-ran evaluations on NVIDIA L40 GPUs (Ada Lovelace architecture). Table 5 shows that variance not only persists but slightly increases on the L40 GPUs. While absolute benchmark scores naturally shift when changing GPU architectures, the relative performance differences induced by the backends remain structurally consistent.

Table 5: **Ablation: Hardware Variation (NVIDIA L40).** Accuracy (%) on GSM8K.

Model	transformers	llama.cpp	LMDeploy	Ollama	SGLang	vLLM	Max - Min
Llama 3.1 8B	83.93 ± 00.00	84.00 ± 00.00	**84.76** ± 00.00	<u>73.31</u> ± 00.00	83.62 ± 00.00	84.08 ± 00.00	11.45
DeepSeek R1 7B	78.01 ± 00.00	78.24 ± 00.00	73.31 ± 00.00	<u>61.71</u> ± 00.00	**78.92** ± 00.00	78.54 ± 00.00	17.21

C.4 Stochastic Sampling

While we utilized greedy decoding ($T = 0$) to strictly isolate backend differences, real-world deployments often rely on temperature sampling. To confirm that the divergence established at $T = 0$ does not disappear when stochastic sampling is introduced, we evaluated GSM8K with temperature $T = 0.7$. As seen in Table 6, while absolute accuracies drop slightly and standard deviations naturally increase due to sampling randomness, the Max-Min differences remain nearly identical to the greedy decoding setup. Because engines alter token probabilities at the logit level, the underlying distributions sampled from remain fundamentally different, proving this variance is not an artifact of greedy decoding.

D Root Cause Analysis

As established in Section 6, the massive divergences in benchmark performance across inference engines are not random anomalies, but rather the direct result of specific system-level design choices. To isolate and quantify these sources of variance, we conducted a series of targeted ablation studies on the GSM8K benchmark using Llama 3.1 8B and DeepSeek R1 Distill Qwen 7B.

Table 6: **Ablation: Stochastic Sampling (Temperature = 0.7).** Accuracy (%) on GSM8K.

Model	transformers	llama.cpp	LMDeploy	Ollama	SGLang	vLLM	Max - Min
Llama 3.1 8B	**82.00** $\pm$ 00.70	81.00 $\pm$ 00.70	80.90 $\pm$ 01.00	73.30 $\pm$ 01.00	81.20 $\pm$ 00.80	81.10 $\pm$ 00.80	08.70
DeepSeek R1 7B	**78.10** $\pm$ 01.00	77.40 $\pm$ 01.70	70.40 $\pm$ 02.90	61.70 $\pm$ 01.10	77.90 $\pm$ 00.50	76.50 $\pm$ 03.90	16.30

Because modern inference backends possess distinct architectures, default configurations, and custom optimization kernels, not all ablations apply universally to every engine. For instance, Ollama enforces specific, hidden preprocessing defaults, whereas engines like vLLM and SGLang introduce variance strictly through high-throughput optimization techniques.

We broadly categorize these root causes into two groups:

1. **Systematic Engine Defaults:** These are correctable, engine-specific configurations applied prior to generation. As shown in Table 7, hidden prompt mutations (such as forceful BOS token injection) and hidden default repetition penalties fundamentally alter the prompt structure and token distributions. Correcting these defaults yields massive performance recoveries, particularly for reasoning models (e. g., DeepSeek R1 recovering up to 11.67 percentage points when Ollama's repetition penalty is disabled).
2. **Optimization-Induced Numerical Variance:** Even after aligning all generation parameters and prompt templates, subtle numerical drift persists due to the underlying mathematical execution. Features essential for high-throughput serving, such as Prefix Caching, CUDA Graphs, and custom kernels for greedy decoding, alter floating-point accumulation. While these shifts are smaller (typically $<1\,\%$), they are highly unpredictable and can arbitrarily increase or decrease model performance.

Table 7 details the exact performance shifts caused by isolating these individual optimizations and defaults. Original accuracies are provided as a baseline to demonstrate the relative impact of each ablation.

Table 7: **Root Cause Ablation Results on GSM8K.** Performance impact of isolating specific system-level defaults and hardware optimizations. Metrics are reported as Accuracy (%). The Δ column represents the absolute percentage point shift caused by the ablation.

Inference Engine	Ablated Feature / Optimization	Llama 3.1 8B			DeepSeek R1 Distill Qwen 7B		
		Original	Ablated	Δ	Original	Ablated	Δ
llama.cpp	Disable Prefix Caching	84.15	84.08	-0.07	78.24	78.70	+0.46
Ollama	Disable Prefix Caching	74.30	73.24	-1.06	61.87	61.41	-0.47
	Remove Hidden BOS Token	74.30	82.64	+8.34	61.87	69.22	+7.35
	Disable Repetition Penalty[†]	74.30	75.97	+1.67	61.87	73.54	+11.67
SGLang	Disable Prefix Caching	84.23	83.85	-0.48	78.24	78.54	+0.30
	Disable CUDA Graphs	84.23	84.23	0.00[*]	78.24	78.24	0.00[*]
vLLM	Disable Prefix Caching	83.85	84.31	+0.46	78.32	78.32	0.00
	Disable CUDA Graphs	83.85	84.00	+0.15	78.32	78.47	+0.15
LMDeploy	Patch Argmax Kernel tie-breaking	84.53	84.08	-0.45	73.54	73.62	+0.06
	Remove Hidden BOS Token	–	–	–	73.54	78.24	+4.70

[†] Ollama enforces a default repetition penalty of 1.1. Disabling it sets the penalty to 1.0.

[*] While overall accuracy remained identical, structural divergence was observed at the token and logit level.

E Paper Survey Methodology and Artifacts

To conduct the large-scale literature survey detailed in Section 7, we employed a multi-stage automated extraction pipeline utilizing an LLM-as-a-judge. This section documents the exact methodology and artifacts used to ensure the reproducibility of our survey. First, we outline the logic of our extraction pipeline. Then, we provide the specific Python lists of target keywords and dependency files used for the initial filtering and code repository validation. Finally, we include the exact prompts provided to the LLM judge.

E.1 Pipeline Methodology

To efficiently and accurately process the initial corpus of over 35,000 papers, our automated extraction pipeline was executed in sequential stages:

1. **Keyword Pre-Filtering:** We first applied a heuristic pre-filter, since running an LLM judge over the full corpus was computationally expensive. We scanned the extracted raw text (using `pymupdf` Python library) of all PDFs for specific terms related to open-weight models and local execution (see Section E.2). Only papers containing at least one of these keywords advanced to the LLM judge.
2. **LLM Prompting Strategy:** We utilized Qwen3-235B-A22B-Instruct-2507-AWQ (using vLLM and greedy decoding) as our LLM judge, processing the full text of each paper. To maximize accuracy and minimize hallucinations, the judge was prompted using a Chain-of-Thought (CoT) approach. As seen in the prompts, the model was instructed to first output a `thought_process` explaining its reasoning based on the provided inclusion/exclusion criteria, before outputting the final classification or extracted text.
3. **Structured Output:** To allow for automated parsing of the LLM's decisions, the judge was strictly prompted to return responses in a valid JSON format. This allowed our evaluation scripts to programmatically route papers through the subsequent *Code Extraction* and *Engine Extraction* stages based on the boolean flags generated during the *Relevance Filtering* stage.

E.2 Filtering Keywords

Relevance Filtering Keywords

```
KEYWORDS = [
    "large language model", "large language models",
    "LLM", "LLMs",
    "language model", "language models",

    "Transformer", "Transformers",
    "Transformer-based",

    "generative AI",
    "foundation model", "foundation models",
    "in-context learning",
    "chain-of-thought",

    "GPT", "Llama", "Mistral", "Falcon", "Qwen", "DeepSeek",
]
```

E.3 LLM Judge Prompts

Relevance Filtering Judge Prompt

```
You are an expert NLP researcher conducting a systematic survey on "LLM Inference Backends."
Your task is to analyze the text of a research paper and determine if it is RELEVANT for a study on how inference
engines (like vLLM, llama.cpp, etc.) affect the performance of Open-Source/Local **Text-Only** Large Language Models.
### STRICT EXCLUSION CRITERIA (CHECK THESE FIRST)
Mark as "relevant": false if ANY of the following apply, even if other criteria are met:
1.  **Multimodal Inputs (Vision/Audio)**:
   *   **The Paper uses Images, Video, or Audio as input.**
   *   **VLMs are EXCLUDED**, even if they use a Llama/Qwen backbone.
   *   **Diffusion/Generative image models are excluded**
   *   *Excluded Models*: LLaVA, Qwen-VL, GPT-4V, Phi-Vision, CLIP, MiniCPM-V, BakLLaVA, Yi-VL.
   *   *Reasoning*: The inference stack for VLMs involves visual encoders/projectors, which is outside the scope of
   text-only inference backends.
2.  **Non-Generative Architectures**:
   *   **Topic Models / Clustering**: Papers focusing on extracting topics (LDA, BERTopic, Autoencoders) without
   autoregressive generation.
   *   **Embeddings Only**: Papers that only use the model to generate vector embeddings (hidden states) for
   retrieval/search, without decoding text.
   *   **Encoder-Only / Autoencoders**: BERT, RoBERTa, DeBERTa, VAEs.
   *   **Non-Transformers**: RNNs, LSTMs, SSMs (Mamba/RWKV), etc. *unless* comparing against Transformer LLMs.
3. **Purely Proprietary/Black-Box**: The paper ONLY uses closed-source models without comparing them to local models.
   *   *Exclusion List*: GPT-3.5, GPT-4, GPT-4o, o1, GPT-5, OpenAI, Claude (Sonnet/Opus/Haiku), Gemini (Pro/Ultra),
   PaLM, Grok (proprietary versions) etc.
```

```
   *   *Exception*: If the paper compares GPT-4 vs. Llama 2, it is RELEVANT.
4.  **Secondary Analysis of Pre-Generated Data (PASSIVE USAGE)**:
   *   **CRITICAL EXCLUSION**: If the authors use an *existing dataset* (e.g., a corpus, a benchmark, or human-eval
   data) where the text was generated by LLMs in a *previous study*, this paper is **IRRELEVANT**.
   *   *Example of Exclusion*: "We analyze the *EMTeC corpus* (Smith et al.), which contains text generated by
   Llama-2." (Here, the *current* authors did not run Llama-2; Smith et al. did).
   *   *Reasoning*: We are studying the inference engine used *by these authors*. If they are analyzing downloaded
   data, they are not running an inference backend.
### INCLUSION CRITERIA (MUST MEET ALL)
To be marked as "relevant": true, the paper must meet these conditions:
1.  **Task = Autoregressive Text Generation**:
    *   The model must receive **Text** as input and generate **Text/Code** (or logits for text tokens) as output.
    *   The mechanism must be next-token prediction (Transformer Decoder).
2.  **Model = Open-Weights / Local**:
    *   The authors must utilize models where weights are publicly available or can be hosted locally.
    *   *Examples*: Llama (1, 2, 3), Mistral, Mixtral, Qwen (Text-only), DeepSeek (Text-only), Gemma, Phi, Yi,
    Falcon, OPT, Dolphin, Kimi, Vicuna, Alpaca, Pythia, BLOOM, OLMo, Solar, StarCoder.
3.  **Action = Running Inference**:
   *   The authors must **actively execute** the model themselves during the course of the study.
   *   This includes:
      *   Running the model to generate *new* responses.
      *   Running the model to calculate perplexity/logits on a dataset.
      *   Running the model to benchmark speed/latency.
   *   *Note*: Papers that Fine-Tune (SFT/RLHF/GRPO etc.) are RELEVANT if they subsequently evaluate the model
   using inference (calculating accuracy, perplexity, or generating responses).
   *   *Note*: Usage via APIs (e.g., Together AI, Anyscale) is RELEVANT if the underlying model is open-weights
   (e.g., calling Llama-3-70B via API).
   *   *Note:* Papers focused on "LLM-as-a-judge" or "Synthetic Data Generation" using open models ARE relevant.
### OPERATIONAL GUIDELINES
1. **Robustness to Artifacts**: The input text is extracted from PDFs and may contain OCR errors, headers/footers,
broken lines, or merged words (e.g., "Lla ma-2", "Hugging Face", "Q wen"). Look past these structural issues to
understand the semantic content.
2. **Model Family Inheritance**: Use the model's name to infer its nature.
   - If a model is unknown to you (e.g., "Llama-4" or "Mistral-Next") but shares a name with a known open-source
   family (Llama, Mistral, Qwen, etc.), **assume it is open-source**.
   - Conversely, if it shares a name with a proprietary family (e.g., "GPT-5", "Claude-Next"), assume it is excluded.
3. **Inference Engine Agnosticism**:
   - **Do not look for specific engine names** (like vLLM, llama.cpp, SGLang) to determine relevance.
   - Many authors fail to report their backend. If the paper *uses* a relevant model (e.g., Llama 2) for inference,
   it is **RELEVANT**, regardless of whether they mention the software stack used to run it.
4. **Non-Exclusive Examples**: The inclusion/exclusion model lists provided above are **representative samples**, not
exhaustive lists. If a paper uses a model not listed (e.g., "MiniCPM" or "XVerse"), use your judgment: if it is an
open-weights generative transformer, include it.
5. **Knowledge Cutoff & New Models**: You may encounter models released after your training data cutoff. **Do not
hallucinate**. Instead, look for context clues in the text to classify them.
   - *Clues for Relevance*: "weights released," "available on GitHub/HuggingFace," "reproduced locally," "7B
   parameters."
   - *Clues for Exclusion*: "proprietary model," "image generation," "diffusion process."
6. **Indirect Citations (Reference Lookup)**: If the authors refer to a model only by citation (e.g., "We utilize
the model proposed by Touvron et al. [15]" or "the model from [1]"), you **MUST** look at the
References/Bibliography section at the end of the text to identify the model. If citation [15] is the "Llama 2"
paper, then the paper is RELEVANT.
7. **Burden of Proof (Uncertainty = Reject)**: You must find **positive evidence** of the criteria above. If the
text is too vague, lacks sufficient information, or you are unsure, mark it as **"relevant": false**.
8.  **Dataset Origin vs. Experimentation (The "Created By" Check)**:
    - Pay close attention to grammar. If the text says: *"We use Data X (Author, Year), which was created using
    Model Y"*, the paper is **NOT RELEVANT** (unless they *also* run Model Y separately).
    - If the text says: *"We used Model Y to create Data X"*, the paper is **RELEVANT**.
---
### INPUT TEXT
<paper_text>
{full_pdf_text_extracted}
</paper_text>
---
### FINAL INSTRUCTION
Based on the text above, determine if this paper is RELEVANT.
Respond with valid JSON only:
{
  "thought_process": "Brief explanation. Did you find specific open model names? Did they run inference?",
  "relevant": boolean
}
```

Engine Extraction Judge Prompt

```
You are an expert Systems and Reproducibility Researcher analyzing a machine learning paper.
Your goal is to extract the **Inference Engine(s)** or **Backend(s)** used to actively execute **Open-Weight / Local
Text-Only Models**.
### 1. TARGET SCOPE (READ CAREFULLY)
You must filter your extraction based on the **Model** and **Task**.
*   **INCLUDE (Target Models)**: Open-Weights or Local models (e.g., Llama, Mistral, Qwen, DeepSeek, Gemma, Phi,
Falcon, Yi, OLMo).
    *   *Unknown Models*: If a model name is unfamiliar (e.g., "X-7B"), look for context clues ("weights on
    HuggingFace", "locally hosted"). If it looks like an open generative transformer, count it.
*   **INCLUDE (Target Task)**: Autoregressive Text Generation / Code Generation.
*   **EXCLUDE (Do not extract backends for these)**:
    *   **Proprietary Models**: If the paper runs GPT-4, Claude, or Gemini, **IGNORE** the API/Backend used for them.
    We only care about the open-source side.
    *   **Multimodal (VLMs)**: LLaVA, Qwen-VL, CLIP. (Inference stacks for vision differ from text-only stacks).
    *   **Non-Generative**: BERT, Encoders, Embeddings-only.
```

```
**Example of Mixed Usage:**
If a paper says: *"We compared GPT-4 (via OpenAI API) against Llama-3 (running on vLLM)."*
-> **Result**: Extract `vLLM`. Ignore `OpenAI API`.
### 2. DEFINITION: WHAT IS AN INFERENCE ENGINE?
An inference engine is the specific software stack that manages the model's weights and executes the forward pass
(generation). It is distinct from the model itself (e.g., "Llama-3" is a model; "vLLM" is the engine).
We categorize engines into three types:
1.  **Self-Hosted Libraries**: Software running on the user's hardware (e.g., `vLLM`, `llama.cpp`, `SGLang`,
`HuggingFace Transformers`, `TGI`, `LMDeploy`, `TensorRT-LLM`).
2.  **Managed Inference Platforms**: APIs serving open-weight models (e.g., `Together AI`, `Fireworks AI`,
`RunPod Serverless`).
3.  **Aggregators**: Routers that sit in front of providers (e.g., `OpenRouter`, `LiteLLM`).
### 3. KNOWN ENGINE LIST (Reference Only)
Use this list to help identify potential candidates, but **do not limit yourself to it**. Context matters more than
the list.
<known_engines>
{known_engines_list}
</known_engines>
### 4. CRITICAL LOGIC: ACTIVE EXECUTION vs. PASSIVE CITATION
Just because an engine is mentioned does not mean it was used.
*   **TRUE (Used)**:
    *   "We generated responses using **vLLM**."
    *   "Latency was measured on **llama.cpp**."
    *   "Models were deployed using **TGI**."
    *   "We use the standard **HuggingFace** implementation."
*   **FALSE (Reference/Comparison)**:
    *   "vLLM [15] is a popular system." (Background info).
    *   "We compare our method against the numbers reported by SGLang." (They didn't run SGLang; they just cited
    numbers).
    *   "We used the dataset from Smith et al., which was generated using vLLM." (Passive usage).
### 5. PRECISE NAMING & UNKNOWN LIBRARIES
**This is the most critical step for new or specialized tools.**
1.  **Do Not Over-Normalize**: Many libraries have similar names. Do not merge them unless they are aliases.
    *   *Example:* If the text says `FastTransformer`, do NOT map it to `transformers`. Report `FastTransformer`.
    *   *Rule:* Only map generic terms like "HuggingFace", "HF", or "AutoModel" to `transformers`. If a specific,
    distinct library name is used (even if it contains the word "Transformer"), **extract the exact name**.
2.  **Unknown/New Libraries**: The authors may use a library not in your known list or one released after your
knowledge cutoff.
    *   *Rule:* If the text explicitly states a software tool was used for inference/execution, **extract it**, even
    if you have never heard of it. Trust the text.
### 6. ROBUSTNESS & NORMALIZATION
*   **OCR Artifacts**: Fix broken text. `v LLM` -> `vLLM`, `llama . cpp` -> `llama.cpp`.
*   **Ambiguity**:
    *   "PyTorch" / "Native PyTorch": If they wrote a custom loader/engine, mark as `Custom/PyTorch`.
    *   "JAX": If they wrote a custom loader/engine, mark as `Custom/JAX`.
    *   "TensorFlow": If they wrote a custom loader/engine, mark as `Custom/TensorFlow`.
## SEARCH STRATEGY (Where to look)
The engine name is most likely found in one of these locations:
*   **Experimental Setup / Implementation Details**: The most likely location.
*   **Footnotes**: Authors often bury the backend version or name here (e.g., "We used vLLM v0.2.3").
*   **Appendix**: Look for "Compute Resources" or "Hyperparameters".
*   **Code Snippets**: Look for imports like `from vllm import LLM` or `import sglang`.
However, do not limit yourself to search for the engine only in those sections. Use the full paper.
### OUTPUT FORMAT
Respond with valid JSON only.
{
  "thought_process": "Step 1: Identify open-weight models used (e.g., 'They used Llama-2'). Step 2: Look for the
  execution software for THOSE models. Step 3: Verify active execution (not just citation). Step 4: Check if this
  software is a distinct library. Quote the relevant sentence.",
  "backend_reported": boolean, // true if they explicitly name the software stack used for open models
  "backends_found": [string, string] // List of clean names (e.g., ["vLLM", "transformers"]). Empty [] if none found.
}
### INPUT TEXT
<paper_text>
{full_pdf_text_extracted}
</paper_text>
```

Code Extraction Judge Prompt

```
You are an expert Reproducibility Reviewer for a top-tier Machine Learning conference.

Your sole objective is to analyze the full text of a research paper and locate the **official code repository**
provided by the authors.

### THE CHALLENGE
You are working with raw text extracted from a PDF. This text often contains errors, such as:
*   **Broken URLs**: `git hub . com / user / repo` or `https://github.com/ \n user/repo`.
*   **Merged Text**: `code is available atgithub.com/user/repo`.
*   **Hidden Links**: Links might be in footnotes, references, or the abstract.

**You must robustly scan the text, identify the link, clean it, and verify it.**

### SEARCH STRATEGY (Where to look)
The code link is almost always found in one of these locations. Check them mentally in this order:
1.  **Abstract**: specifically the very last sentence.
2.  **Introduction**: specifically in the "Contributions" list or the final paragraph.
3.  **Footnotes**: Look for text like "See footnote 1" or "[1]" near the mention of code.
4.  **Methodology header**: Sometimes listed as "Implementation Details".
5.  **Conclusion**: A section named "Reproducibility" or "Data Availability".
```

```
6.  **References/Bibliography**: Rarely, authors cite their own code as a bibliography entry (e.g., "Source Code
[25]").


### CRITICAL DECISION LOGIC

**1. Verification of Ownership (The "Author" Check)**
You must distinguish between **Own Work** and **Prior Work**.
*   **RELEVANT (True)**: "We release our code at...", "The official implementation is available at...", "Project
page: [URL]", "Code: [URL]", "Our code is open-sourced."
*   **IRRELEVANT (False)**: "We used the implementation from [Citation]", "Built upon the codebase of [Citation]",
"We use the HuggingFace library", "Model weights are available at [URL]" (if strictly weights only).

**2. Handling "Coming Soon" (The Promise Check)**
*   **False**: "Code will be released upon acceptance", "We plan to release code soon.", "Code available upon
request."
*   **True**: Anonymized repositories used for review (e.g., ‘anonymous.4open.science‘) ARE valid code reporting.
*   **True (Specific Link Provided)**: If the text provides a **SPECIFIC URL**, mark this as ‘true‘ **even if the
text uses future tense** (e.g., "Code *will be* released at...", "We *plan to* release code at...").

**3. Robustness to OCR/Parsing Artifacts (Reconstruction)**
*   PDF parsing often breaks URLs with spaces, hyphens, or newlines.
*   *Task:* If you find a broken URL in the text that points to the author’s code, **you must clean it** (remove
spaces, fix formatting) and return the corrected URL in the JSON output.

**4. Valid Targets**
*   **Repositories**: GitHub, GitLab, Bitbucket, etc.
*   **Project Pages**: (e.g., ‘github.io‘, ‘site.net‘) ARE valid if the text implies code is linked there.
*   **Anonymous Links**: ‘anonymous.4open.science‘, Dropbox, Google Drive (if explicitly stated as the code release).
*   **HuggingFace**: If the authors link to a HuggingFace repository that clearly contains the *code implementation*
and not just model weights/checkpoints, accept it. If unsure, prioritize GitHub.

### URL RECONSTRUCTION INSTRUCTIONS
The text extraction may insert spaces or newlines into URLs.
*   *Raw Text:* "g it hub . c om / my lab / my repo"
*   *Your Output:* "https://github.com/mylab/myrepo"
*   **Action**: You must intelligently remove whitespace and fix formatting to produce a valid URL string.

### OUTPUT FORMAT
Respond with valid JSON only.

{
  "thought_process": "Analyze the text. 1) Did they mention releasing code? 2) Is the link for THEIR code or a
  library? 3) Does the link look like a repository? Briefly explain your reasoning regarding ownership and text
  location.",
  "code_reported": boolean, // true if they linked their own code
  "official_url": string OR null // The best specific URL found. If null, return null. CLEAN THE URL (remove spaces
  /newlines) before returning.
}

---
### INPUT DATA
<paper_text>
{full_pdf_text_extracted}
</paper_text>
```

E.4 Dependency File Patterns

Dependency Files

```
DEPENDENCY_FILES = [
    "requirements.txt", "requirements.pip", "pyproject.toml", "poetry.lock",
    "pipfile", "pipfile.lock", "setup.py", "setup.cfg", "tox.ini",

    "environment.yml", "environment.yaml", "meta.yaml",

    "dockerfile", "docker-compose.yml", "docker-compose.yaml", "containerfile",
    "devcontainer.json",

    "cargo.toml", "cargo.lock", "cmakelists.txt", "makefile", "package.json",
    "yarn.lock", "pnpm-lock.yaml", "project.toml", "manifest.toml",

    "install.sh", "setup.sh", "build.sh"
]
```

F Manual Verification of Judge Results

To assess the reliability of our automated pipeline, we conducted a manual verification on a random subset of the corpus. We selected 50 papers for each of the three processing stages (Relevance Filtering, Code Extraction, and Engine Extraction) resulting in a total of 150 manually audited papers. We compared our manual classification against the LLM judge's output to calculate agreement rates and analyze failure modes:

- **Relevance Filtering (88 % Agreement):** The primary source of disagreement was papers utilizing unknown or unpopular open-weight models that the judge failed to recognize. Additionally, the judge occasionally missed relevant papers where LLM usage was mentioned exclusively in a specific subsection of the appendix rather than the main body.
- **Code Extraction (96 % Agreement):** The few discrepancies arose from two specific scenarios: cases where the judge interpreted a textual "promise to share code in the future" as an existing repository, and PDF parsing issues where valid repository links were embedded in a format our parser could not extract.
- **Engine Extraction (94 % Agreement):** Disagreements in this stage were primarily due to false positives where the judge flagged low-level kernel libraries as full inference engines. One error also stemmed from the usage of niche libraries not known to the judge.

G Impact Statement

This paper aims to improve the scientific quality and reproducibility of LLM evaluations. By quantifying the numerical instability introduced by different inference backends, we highlight a critical blind spot in current benchmarking practices. The primary positive impact of this work is to encourage more transparent reporting standards, ensuring that claims of "State-of-the-Art" performance are statistically significant rather than artifacts of system optimizations.

On a broader societal level, this work has implications for AI safety and reliability. As we demonstrate, optimization techniques can alter model behavior. A model aligned for safety in a development environment may exhibit divergent, potentially unsafe behaviors when deployed on high-throughput inference engines. Identifying and mitigating this source of variance is essential for the safe integration of LLMs into critical domains such as healthcare and finance.

H Model & Dataset Licenses

Table 8: LLMs and Datasets used in this paper alongside their licenses.

Model/Dataset	License
Llama3.1 8B [22]	Llama Community License
Qwen3 4B [23]	Apache-2.0
Qwen3 30B [23]	Apache-2.0
DeepSeek R1 Distill Qwen 7B [24]	MIT License
Qwen3 Thinking 30B [23]	Apache-2.0
Qwen3-235B-A22B-Instruct-2507-AWQ [31]	Not specified
GPT-4o-mini [32][3]	OpenAI Terms of Use
GSM8K [25]	MIT License
GPQA Diamond [26]	CC BY 4.0
SimpleQA Verified [27]	Apache-2.0
LiveCodeBench v6 [28]	CC
JailbreakBench [30]	MIT License

[3] Used as LLM judge for SimpleQA and JailbreakBench

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: **[TODO]**

   Justification: **[TODO]**

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.

- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: **[TODO]**

    Justification: **[TODO]**

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.