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 spanning 4B-30B parameters 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 such as prefix caching and CUDA graphs, custom kernels, and engine-specific defaults in logit processing. Our findings identify the inference backend as a previously unreported yet consequential hyperparameter in LLM evaluations 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 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, including tools such as vLLM [1], SGLang [2], and llama.cpp [3], which are 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.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

This potential source of variance has critical implications. Different inference engines can yield different 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., Hugging Face transformers [4]) may exhibit different, potentially unsafe behaviors when deployed on a high-throughput engine such as 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 16.6 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.
- **Root Cause Analysis.** By isolating specific optimizations, we trace backend-induced variance to two primary sources: correctable systematic defaults and optimization-induced numerical drift.
- **Reproducibility in ML Research.** We analyze over $35,000$ recent publications from top ML conferences to quantify how frequently the inference stack is reported.

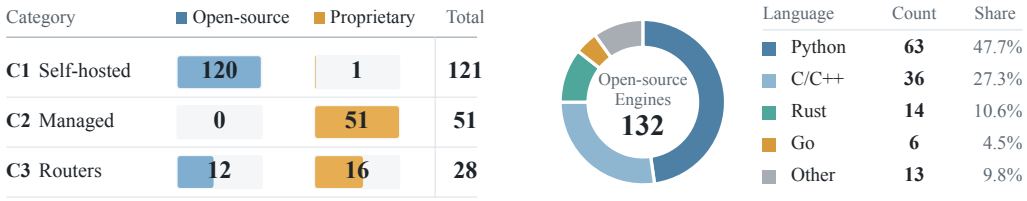
By quantifying this variability, we aim to establish new reporting standards that improve scientific reproducibility in the era of optimized inference. To support reproducibility and future research, we release all code and experimental artifacts at `https://github.com/Dormant-Neurons/inference_backends`.

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], model versioning [10], request-routing services [11], and serving-stack effects on tool-use evaluation [12]. 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 evaluations of inference engines focus primarily on hardware optimizations, speed, and energy efficiency [13–15]. Other work studies system-design choices such as caching and decoding [16], extends earlier findings on CNNs and RNNs [17], or examines quantization-induced differences [18], including cross-kernel scaling and rounding effects [19]. Recent security studies further exploit platform- and kernel-dependent numerical differences to activate backdoors in LLMs and VLMs [20, 21]. Despite optimizations for determinism [22] and guidelines recommending fixed seeds and low temperatures [23], LLMs retain inherent randomness. We build on these performance-centric studies to quantify how backend design choices fundamentally alter generation trajectories.



(a) Engine Categories

(b) Programming Languages

Figure 1: **Landscape of Inference Engines. (a)** Counts of the 200 surveyed inference engines by category (C1-C3) and source-code availability (blue: open-source; orange: proprietary). **(b)** The distribution of primary programming languages used across the open-source engines.

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. We survey this ecosystem to quantify its diversity.

3.1 Survey Methodology and Scope

We define an *inference engine* as standalone software that can load a transformer-based model and generate completions. We surveyed the ecosystem as of January 2026.

Search Strategy. We identified candidates through GitHub searches (e. g., "LLM inference" and "inference-engine"), general web search, and community discussions on platforms including Reddit and Hugging Face.

Inclusion and Exclusion Criteria. We include software that (1) supports open-weight or local models, (2) has a verifiable open-source repository or API, and (3) shows evidence of adoption or availability (at least 100 GitHub stars for open-source projects or an active API for proprietary services). These criteria focus the survey on systems relevant to open-weight evaluation that can be independently identified and accessed. We exclude foundational libraries (e. g., PyTorch, JAX) and training/application-layer tools (e. g., LangChain), as neither is a dedicated inference engine. We further exclude engines that serve only VLMs or diffusion models, as they fall outside our text-generation scope.

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:

Category 1
Self-hosted inference libraries
Users run the model and manage inference software and compute.

Examples:
vLLM, llama.cpp, SGLang.

Category 2
Managed inference platforms
Providers run models and expose inference through APIs.

Examples:
Fireworks AI, Together AI.

Category 3
Aggregators and routers
A interface routes requests to providers or self-hosted endpoints.

Examples:
OpenRouter, LiteLLM.

3.3 Landscape Analysis

Our search identified **200** inference engines (Figure 1a): 121 self-hosted libraries (Category 1; 60.5 %), 51 managed platforms (Category 2; 25.5 %), and 28 aggregators/routers (Category 3; 14.0 %). This variety of available engines shows software choice is a significant variable in experiments. Of these, 132 are open-source, concentrated in Category 1 (120/121); all Category 2 systems are proprietary, while Category 3 includes both (12 open-source, 16 proprietary). Source availability affects how much of each system can be inspected and managed by the user.
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 isolate inference engine effects using a controlled experimental framework.

4.1 Experimental Scope

Inference Backends. We selected five Category 1 (self-hosted) inference engines based on adoption and experimental compatibility: vLLM [1], SGLang [2], llama.cpp [3], LMDeploy [24, 25], and Ollama [26]. We use Hugging Face transformers [4] as the reference implementation. Self-hosting lets us hold the surrounding inference environment fixed and isolate the backend; managed platforms and routers introduce additional hidden or provider-controlled components. Full backend-selection details are provided in Appendix A.

Models. We selected five open-weight text-generation models spanning 4B-30B parameters and multiple architectures. *Standard models:* Llama 3.1 8B [27], Qwen3 4B, Qwen3 30B [28]. *Reasoning models:* DeepSeek R1 Distill Qwen 7B [29] and Qwen3 Thinking 30B [28].

Benchmarks. To study performance differences across inference backends rather than absolute model performance, we use four established benchmarks covering distinct tasks: GSM8K (Math) [30], GPQA Diamond (Science) [31], SimpleQA Verified (Factuality) [32], and LiveCodeBench v6 (Code) [33].

Licensing information for the models and datasets used in this study is provided in Appendix J.

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. We apply the tokenizer's Jinja2 chat template externally to avoid backend-specific templating.

Batch Size. Evaluations use a batch size of one to prevent batching-induced instability [8, 34].

Generation Parameters. We set the maximum number of output tokens to $2,048$ (Standard) and $32,768$ (Reasoning), with context windows of $4,096$ and $34,816$ respectively.

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

5 Evaluation and Results

All evaluations use a unified Docker container (Ubuntu 22.04, Python 3.12), a single NVIDIA H100 GPU, and fixed backend versions (Appendix A, Table 2). We study behavioral reproducibility by combining aggregate scores with per-example, token-level, and probability-level divergence. Complementary throughput and latency measurements are reported in Appendix B.

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

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 rankings. 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.

We additionally examine whether backend-induced score shifts change model ordering. Across our model set, 19 of 20 backend-benchmark combinations preserve the transformers ordering, which is unsurprising when models are separated by large performance gaps. However, for more closely matched models, reversals can occur: on GPQA, LMDeploy reverses Qwen3 4B and DeepSeek R1 Distill Qwen 7B relative to transformers (35.35 vs 35.86, compared with 35.86 vs 33.33). This suggests that backend choice is particularly consequential when comparing models with similar benchmark performance.

Ultimately, no backend matches transformers across all benchmarks. Nor do we find evidence for a universally superior backend. Across the 20 model-benchmark combinations, four backends achieve the best mean rank for at least one model; Appendix C provides the full analysis. We therefore treat backend selection as an experimental variable rather than an optimization for accuracy.

Disagreement Rate (Figure 3 in Appendix D.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 example, Ollama disagrees with transformers on $27.37\,\%$ of DeepSeek R1 Distill Qwen 7B's GSM8K predictions, more than 360 of $1,319$ questions. Disagreement is also pronounced for reasoning models on GPQA and LiveCodeBench.

Length Error (Figure 4 in Appendix D.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 ≤ 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 substantially for all tested models. Reasoning models show large length differences on all datasets. For instance, the Ollama backend produces DeepSeek R1 Distill Qwen 7B 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 D.3), we observe that reasoning models consistently diverge from the reference generation, much earlier than standard models. This is amplified on difficult benchmarks like GPQA, where, for example, Llama 3.1 8B 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. 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

Table 1: **Backend Performance Variance across Benchmarks.** Performance of five inference backends and the transformers reference across models and benchmarks. Metrics are accuracy (%) for GPQA and GSM8K, F1 for SimpleQA, and pass@1 for LiveCodeBench. The final column reports the maximum−minimum score difference. 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

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 E).

Safety implications (Appendix E.1). Using JailbreakBench [35], we found that DeepSeek R1 Distill Qwen 7B's vulnerability to adversarial prompts fluctuated by 8.9 percentage points based solely on the inference engine.

Batching (Appendix E.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 Variation (Appendix E.3). Evaluating on NVIDIA L40 GPUs yielded consistent, slightly increased backend-induced variance (up to 17.2 percentage points between backends), confirming divergence is rooted in software implementations rather than specific GPU architectures.

Stochastic Sampling (Appendix E.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

To explain the score shifts in Section 5, we ablate specific optimizations and trace execution across Llama 3.1 8B, DeepSeek R1 Distill Qwen 7B, and Qwen3 4B on GSM8K and GPQA. We categorize the observed causes into two groups: (1) systematic, correctable engine defaults that alter the prompt or decoding behavior and can explain some of the largest performance divergences 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 F).

6.1 Systematic Engine Defaults

The largest effects in our ablation study arise from engine-specific defaults when these are applicable. In particular, several of the extreme divergences in Table 1, such as DeepSeek R1 Distill Qwen 7B

scoring only $61.87\,\%$ on Ollama compared to $78.47\,\%$ on the transformers reference, can be traced to hidden changes to the prompt or decoding procedure rather than numerical optimizations alone.

Hidden Prompt Mutation: Even with an externally applied Jinja2 template and Ollama's `raw=True` parameter, Ollama prepends a beginning-of-sequence (BOS) token; LMDeploy also inserts a BOS token for DeepSeek R1 Distill Qwen 7B. Removing these tokens raises GSM8K accuracy by 8.34 points for Llama 3.1 8B with Ollama, 7.35 for DeepSeek R1 Distill Qwen 7B with Ollama, and 4.70 for DeepSeek R1 Distill Qwen 7B with LMDeploy. On GPQA, removing Ollama's BOS token raises Llama 3.1 8B accuracy by 4.55 points. Qwen3 4B is unaffected because its template does not use a BOS token.

Default Penalty Parameters: Ollama applies a default repetition penalty of 1.1, severely degrading chain-of-thought generation. Setting it to 1.0 raises GSM8K accuracy by 11.67 points for DeepSeek R1 Distill Qwen 7B and 1.67 for Llama 3.1 8B, substantially narrowing the largest gaps. On GPQA, it changes accuracy by +2.02 points for Llama 3.1 8B and +1.52 for DeepSeek R1 Distill Qwen 7B, with no aggregate effects for Qwen3 4B.

The defaults can therefore explain large backend-induced differences, but their effects depend on the model, backend, and benchmark. Unlike the numerical effects below, they are largely correctable once prompt tokens and decoding parameters are aligned.

6.2 Optimization-Induced Numerical Variance

After aligning prompts and decoding defaults, backend-specific optimizations still introduce numerical differences that can alter generation; their magnitude and direction vary across settings.

Prefix Caching and CUDA Graphs: vLLM, SGLang, llama.cpp, and Ollama enable prefix caching by default. Processing a prompt in fragmented chunks changes floating-point reduction order and can therefore alter subsequent computations. Disabling prefix caching ranges from negligible effects to several percentage points. On GPQA, it changes DeepSeek R1 Distill Qwen 7B's accuracy by +3.54 points with Ollama and -0.51 with llama.cpp.[2] CUDA graphs show a similarly configuration-dependent effect: disabling them leaves aggregate accuracy unchanged in several settings, while shifting GPQA accuracy by up to 1.52 points.

Kernel-Level Tie-Breaking and Accumulation Precision: When tokens have identical FP16 logits, PyTorch deterministically selects the lowest token ID, whereas LMDeploy performs greedy decoding through a parallel Top-K kernel ($K = 1$) creating a hardware race condition that picks arbitrarily. Patching it to match PyTorch changes DeepSeek R1 Distill Qwen 7B's GPQA accuracy by -3.54 points, although other settings change little or not at all. llama.cpp and Ollama also accumulate intermediate matrix multiplications in FP32, introducing additional rounding differences relative 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 accumulate across layers, eventually cascading into different Top-1 token predictions.

Overall, unlike the engine defaults in Section 6.1, we find no individual numerical optimization that consistently increases or decreases performance across settings.

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

Paper Corpus. We collected accepted papers published from 2023-2025 at top-tier ML and NLP conferences (NeurIPS, ICML, ICLR, ACL, and EMNLP) via the OpenReview and ACL Anthology APIs. This yielded an initial corpus of more than $35,000$ papers.

[2] GPQA contains 198 examples, so one net change in correctness corresponds to approximately 0.51 percentage points.

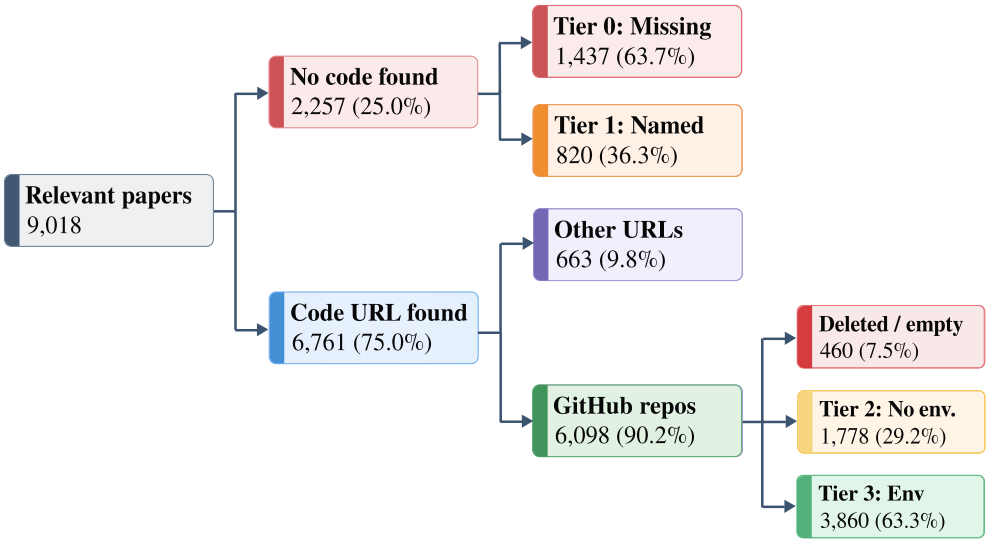


Figure 2: **Reproducibility artifacts among** $9,018$ **papers using local LLM inference.** Percentages are relative to the parent node. Tiers 2 and 3 are assigned only to inspectable GitHub repositories.

Reproducibility Tiers. To categorize reproducibility artifacts available for each paper, we classified papers into four Reproducibility Tiers: **Tier 0**, neither an inference backend nor a code artifact is mentioned; **Tier 1**, the backend is documented textually, but no code artifact is provided; **Tier 2**, an inspectable code repository is available, but specific environment dependencies are absent; and **Tier 3**, an inspectable code repository is provided alongside an explicit dependency or environment specification, such as `requirements.txt`, `pyproject.toml`, or a `Dockerfile`.
These tiers distinguish between verifiable and machine-readable environment definitions (Tier 3) and ambiguous textual descriptions that fail to capture implementation details (Tier 1). 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.

Filtering and Extraction Pipeline. We extracted PDF text using `pymupdf` and applied a keyword pre-filter to identify potentially relevant papers. An LLM-based relevance classifier then identified $9,018$ papers in which the authors perform local inference with open-weight text-generation models. For these 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. Appendix G.1 details the pipeline; G.2- G.3 provide keywords and judge prompts.

Repository Inspection. For direct GitHub repository URLs, we used the GitHub API to inspect repository contents systematically for dependency and environment specifications (Appendix G.4 provides the complete set of file patterns). Code and project URLs that did not directly resolve to GitHub repositories were still counted as available artifacts, but their environment specifications were not assessed systematically. Consequently, these artifacts cannot be reliably distinguished between Tier 2 and Tier 3 based on our automated analysis.

7.2 Results

Setup. We utilized Qwen3-235B-A22B-Instruct-2507-AWQ [36] as the LLM judge, served with vLLM (version 0.13.0) using greedy decoding.

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

Code Artifact Availability. Of the $9,018$ relevant papers, $6,761$ ($75.0\,\%$) contained an author-provided code or project URL. Of these, $6,098$ ($90.2\,\%$) were direct GitHub repositories; the remaining 663 URLs comprised 247 ($3.7\,\%$) GitHub Pages, 74 ($1.1\,\%$) Hugging Face URLs, and 342 ($5.0\,\%$) URLs hosted elsewhere. These were counted as available artifacts, but their environment specifications were not assessed systematically. Among the $6,098$ GitHub repositories, 460 were deleted, empty, or contained only documentation (e. g., a README or license) 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 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 or environment specifications. 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 H).

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 backend-specific implementation choices and optimized execution paths. 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 16.6 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 complicates comparative benchmarking, particularly when performance margins are small or inference stacks differ, as apparent gains may reflect backend artifacts rather than architectural superiority.

No Universally Preferred Backend. Backend-induced variance does not produce a consistent performance ordering: although transformers has the best aggregate mean rank, the preferred backend varies across models and tasks (Appendix C). This makes backend choice an additional experimental variable and reinforces the need to fix and report the inference stack before evaluation.

Security and Robustness. Backend variance also introduces a critical "deployment gap." An identical model's vulnerability to jailbreaks fluctuates by 8.9 percentage points 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. ***Category-Specific Reporting:*** Reporting requirements should reflect the level of control over the inference stack: C1 *Self-Hosted:* Report the inference engine and version, exact model artifact and precision, hardware, decoding parameters, prompt/chat template handling, and relevant optimization settings. Where possible, release environment specifications such as a container or pinned dependency file. C2 *Managed:* Report the provider, exact model identifier, evaluation date, all API parameters, and any model revision or system fingerprints exposed by the service. Because hidden components may change without user control, important evaluations should be repeated and interpreted as time dependent. C3 *Routers:* Report the router and API/version, routing policy, selected provider and model for each request where available, and whether retries or fallbacks were enabled. Exact request parameters and relevant response metadata should be retained.

When neither the route nor underlying model revision can be fixed, results should be described as time-dependent. ***Ensure Fair Comparisons:*** If a SOTA method's backend is undocumented, rerun comparisons using the same backend. ***Account for Non-Determinism:*** Repeated runs can quantify within-backend variability but cannot eliminate systematic differences between engines. Researchers should report these sources of variance separately. ***Report Per-Sample Scores:*** Alongside aggregate metrics, provide per-sample scores to enable paired statistical tests that quantify uncertainty in benchmark differences [37, 38].

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 Backend Sensitivity:*** Where feasible, evaluate using both a reference implementation and a high-throughput engine, and report the score difference.

Inference Engine Developers and 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

Our controlled setup isolates Category 1 self-hosted backends by fixing model artifacts, hardware, prompts, decoding, and software versions. It therefore does not capture additional variability from production concurrency, batching regimes, hardware changes, or Category 2/3 systems, where model revisions, routing, retries, fallbacks, and provider updates may be hidden. Our primary focus is behavioral and evaluation reproducibility; Appendix B reports complementary GSM8K throughput, TTFT, and inter-token latency, but we do not benchmark serving efficiency across workloads or deployment configurations.

We evaluate open-weight transformer text-generation models from 4B to 30B parameters. Our results apply to this model scale. We additionally do not evaluate multimodal, diffusion, or audio models, whose inference stacks may introduce different sources of variance.

We use greedy decoding to reduce sampling randomness, although it is not optimal for every model and can induce repetition in some reasoning models (e. g., DeepSeek R1 Distill Qwen 7B). However, our goal is relative backend differences rather than optimized absolute scores. Appendix E.4 shows that substantial backend differences persist at $T = 0.7$. 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 establish inference backends as an important variable in LLM evaluation. 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. We therefore recommend reporting backend configurations, repeating evaluations, and supporting deterministic 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 Research, Technology and Space (BMFTR) under the grants AIgenCY (16KIS2012) and SisWiss (16KIS2330). In addition, this work was funded by zukunft.niedersachsen, the joint science funding program of the Lower Saxony Ministry of Science and Culture and the Volkswagen Foundation, and the Daimler and Benz Foundation under the grant Ladenburger Kolleg, Project KonCheck. *Competing interests: The authors declare no competing interests.*

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 (SOSP)*, 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*, 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 *Proceedings of the 63rd Annual Meeting of the 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 *Proceedings of the 2024 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 *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[9] Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Jeffrey Hsu, 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, François 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 *Network and Distributed System Security Symposium (NDSS)*, 2026.

[11] Joe Higton, Joshua A. Tucker, and Christopher Barrie. Request routing services undermine the reproducibility of open weight LLMs. *SocArXiv preprint*, 2026.

[12] Lijuan Tang and Yuemeng Zheng. Measuring the serving stack instead of the model: Hidden confounds in local tool-use evaluation. In *Proceedings of the 2nd Workshop for Research on Agent Language Models (REALM)*, 2026. To appear.

[13] 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 *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024.

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

[15] 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. *ACM Trans. Intell. Syst. Technol.*, 2026.

[16] 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 Computing Surveys*, 2025.

[17] Bin Xu, Ayan Banerjee, and Sandeep Gupta. Hardware acceleration for neural networks: A comprehensive survey. *Transactions on Machine Learning Research*, 2026.

[18] 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.

[19] Teng-Ruei Chen. The integer alibi: Localizing cross-kernel divergence in INT8-quantized LLM inference. *arXiv preprint arXiv:2608.13756*, 2026.

[20] Nils Loose, Jonas Sander, Felix Mächtle, and Thomas Eisenbarth. Floatdoor: Platform-triggered backdoors in llms. *arXiv preprint arXiv:2606.19535*, 2026.

[21] Yezheng Cheng, Zexin Li, Jiaqi Wu, Zhihong Zhang, and Simin Chen. VLMSysTrojan: Stealthy system-aware backdoor attacks against vision-language models. In *Computer Vision – ECCV 2026*, 2026.

[22] 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.

[23] 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.

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

[25] 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.

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

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

[28] 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.

[29] Daya Guo et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature 645*, 2025.

[30] Karl Cobbe, Vineet Kosaraju, Mohammad 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.

[31] 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 (COLM)*, 2024.

[32] 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.

[33] 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 *International Conference on Learning Representations (ICLR)*, 2025.

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

[35] 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 *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024.

[36] AIDXteam. Qwen3-235B-A22B-Instruct-2507-AWQ, 2025. URL `https://huggingface.co/AIDXteam/Qwen3-235B-A22B-Instruct-2507-AWQ`.

[37] Jonas Kübler, Kailash Budhathoki, Matthäus Kleindessner, Xiong Zhou, Junming Yin, Ashish Khetan, and George Karypis. When LLMs get significantly worse: A statistical approach to detect model degradations. In *International Conference on Learning Representations (ICLR)*, 2026.

[38] Evan Miller. Adding error bars to evals: A statistical approach to language model evaluations. *arXiv preprint arXiv:2411.00640*, 2024.

[39] NVIDIA. TensorRT LLM. `https://github.com/NVIDIA/TensorRT-LLM`, 2023.

[40] OpenAI. GPT-4o-Mini. `https://developers.openai.com/api/docs/models/gpt-4o-mini`, 2024.

A Backend Selection and Versions

Backend Selection. Our landscape survey identifies many inference engines, but a controlled comparison requires a consistent setup. We therefore selected the five self-hosted engines used in our evaluation based on adoption and experimental compatibility.

We restrict the controlled comparison to Category 1 engines because they let us hold model artifacts, prompts, decoding parameters, hardware, and software versions fixed. Managed platforms may hide or update the backend, model revision, batching strategy, or hardware; routers additionally introduce routing, parameter translation, retries, and fallbacks. Evaluations in those settings therefore characterize the full hosted inference path rather than an isolated backend.

For adoption, we prioritized widely used self-hosted engines identified in Section 3. For compatibility, each engine had to support our models, hardware, tasks, and unified software environment. This yielded vLLM [1], SGLang [2], llama.cpp [3], LMDeploy [24, 25], and Ollama [26], with Hugging Face transformers [4] as the reference implementation.

These requirements excluded some otherwise widely adopted inference engines. For example, TensorRT-LLM [39] introduced dependency conflicts with other evaluated backends, including vLLM, and could not be incorporated without changing the unified environment. Its exclusion therefore reflects experimental compatibility rather than a lack of relevance or adoption.

Software versions. Table 2 lists the inference-backend and reference-library versions used in the controlled experiments. 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 Inference Efficiency

Optimized inference engines are primarily adopted for their serving efficiency. To contextualize the practical trade-off underlying our study, we measure throughput, time-to-first-token (TTFT), P95 TTFT, and inter-token latency (ITL) for every model-backend combination on GSM8K under the same setting as our controlled evaluation. These results provide complementary context rather than a comprehensive serving benchmark, as efficiency depends on workload characteristics, batching, concurrency, sequence length, and engine-specific tuning.

As shown in Table 3, the optimized backends provide substantial efficiency gains over transformers, particularly for the 30B models, while no backend dominates across all models and metrics. For example, SGLang achieves the highest throughput for both 30B models, whereas LMDeploy achieves the lowest mean TTFT. These results illustrate the practical incentive for optimized inference engines that motivates our study, while our main experiments examine whether these gains come with changes in model behavior.

Table 3: **Inference Efficiency on GSM8K.** Throughput is reported in generated tokens/s; TTFT and ITL are reported in seconds. All measurements use the same setting as the controlled evaluation. Best values for each model are shown in bold.

Model	Backend	Throughput ↑	TTFT ↓	P95 TTFT ↓	ITL ↓
Qwen3 4B	transformers	58.146	0.025	0.030	0.015
	llama.cpp	123.694	0.021	0.023	0.007
	LMDeploy	**153.517**	**0.015**	0.019	**0.005**
	Ollama	136.685	0.088	0.090	0.006
	SGLang	147.788	0.019	0.023	**0.005**
	vLLM	140.209	**0.015**	**0.015**	0.006
Llama 3.1 8B	transformers	76.259	0.027	0.030	0.012
	llama.cpp	102.361	0.026	0.027	0.009
	LMDeploy	**112.834**	**0.021**	**0.022**	**0.008**
	Ollama	100.140	0.112	0.115	0.009
	SGLang	109.071	0.023	0.025	**0.008**
	vLLM	106.996	**0.021**	**0.022**	0.009
Qwen3 30B	transformers	12.255	0.548	0.586	0.070
	llama.cpp	93.747	0.159	0.173	0.008
	LMDeploy	131.651	**0.032**	**0.036**	**0.006**
	Ollama	100.993	0.121	0.163	0.008
	SGLang	**137.455**	0.038	0.043	**0.006**
	vLLM	126.893	0.035	0.042	**0.006**
DeepSeek R1 Distill Qwen 7B	transformers	72.400	0.024	0.030	0.013
	llama.cpp	92.430	0.025	0.026	0.010
	LMDeploy	**114.157**	**0.019**	0.023	**0.008**
	Ollama	103.984	0.095	0.098	**0.008**
	SGLang	108.928	0.021	0.026	**0.008**
	vLLM	106.494	**0.019**	**0.019**	0.009
Qwen3 Thinking 30B	transformers	12.827	0.552	0.589	0.070
	llama.cpp	103.055	0.165	0.180	0.009
	LMDeploy	139.336	**0.036**	**0.039**	**0.006**
	Ollama	111.773	0.124	0.164	0.008
	SGLang	**152.070**	0.042	0.046	**0.006**
	vLLM	136.762	0.039	0.046	**0.006**

C Universal Best Backend Analysis

The performance differences in Table 1 raise the question of whether one inference backend systematically achieves better benchmark results. Because the benchmarks use different metrics and scales, we rank the six implementations within each of the 20 model-benchmark combinations and average their ranks. Rank 1 denotes the highest score; ties receive their average rank.

Table 4: **Aggregate Backend Rankings.** Mean rank across the 20 model-benchmark combinations in Table 1 (lower is better). Engines are ranked within each combination according to their mean benchmark score; ties receive average ranks. Top finishes report sole and tied highest-scoring combinations separately.

Backend	**Mean Rank ↓**	**Top Finishes (Sole / Tied)**
transformers	**2.83**	**7 / 1**
vLLM	3.33	3 / 2
LMDeploy	3.55	2 / 2
SGLang	3.63	2 / 1
llama.cpp	3.68	1 / 0
Ollama	4.00	2 / 1

Table 4 shows that transformers obtains the best aggregate mean rank (2.83), but is the sole top-performing backend in only 7 of 20 combinations. Moreover, paired row-bootstrap analysis does not establish a better aggregate rank for transformers than any individual alternative at the 95 % confidence level.

Table 5: **Backend Rankings Depend on the Evaluation Setting.** Best mean-rank backend when aggregating separately within each model (across four benchmarks) and benchmark (across five models).

Subset	**Best Backend**	**Mean Rank ↓**
By model		
Qwen3 4B	transformers	2.50
Llama 3.1 8B	LMDeploy	2.13
Qwen3 30B	Ollama	1.88
DeepSeek R1 Distill Qwen 7B	vLLM	2.00
Qwen3 Thinking 30B	transformers	1.50
By benchmark		
GPQA	transformers	2.70
GSM8K	transformers	1.90
SimpleQA	transformers	3.00
LiveCodeBench	vLLM	1.80

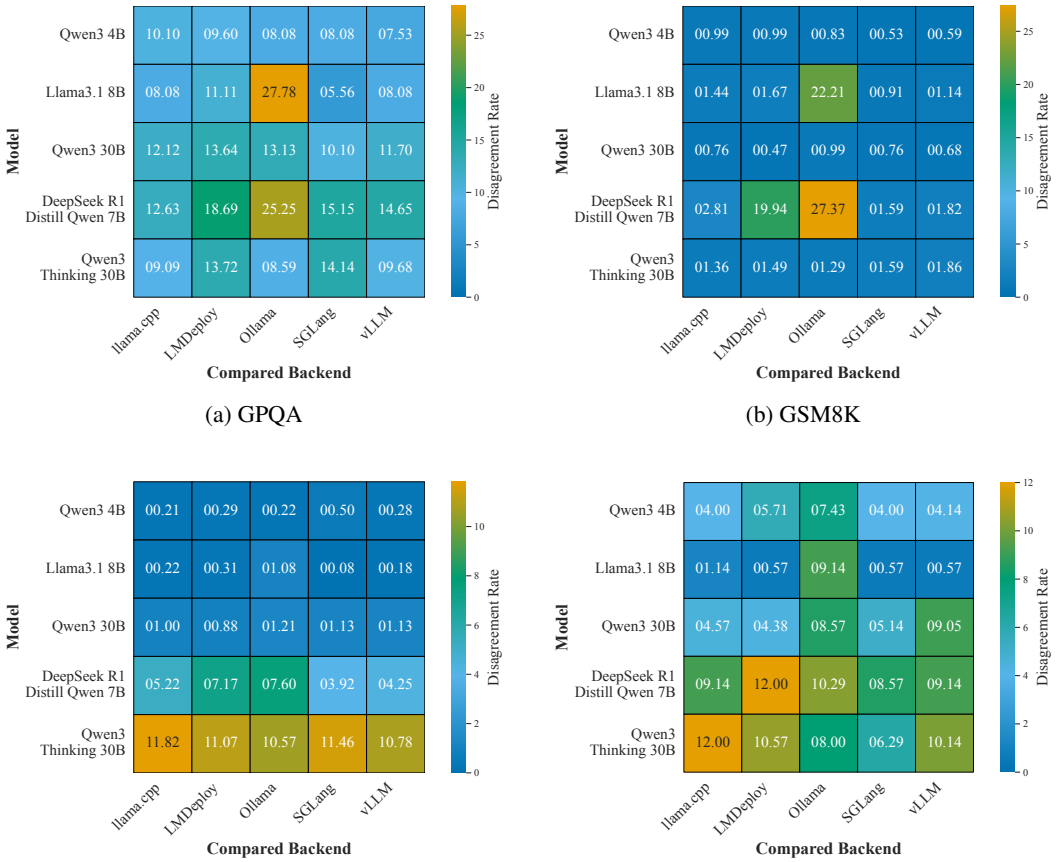
The preferred backend also depends on the evaluated setting (Table 5). Four different backends achieve the best average rank for at least one model, and vLLM ranks first on LiveCodeBench. The aggregate result is also sensitive to the specific evaluation configuration: the leader changes from transformers to vLLM when either GSM8K or Qwen3 Thinking 30B is omitted.
Thus, although transformers has the best average rank across the complete evaluation, we find no evidence for a backend that is uniformly preferable across models and tasks. Selecting a backend after observing benchmark results could improve a particular score. We therefore recommend fixing and reporting the inference stack before evaluation. Our analysis does not provide evidence that prior work intentionally exploits backend choice to inflate results.

D 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.

D.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.



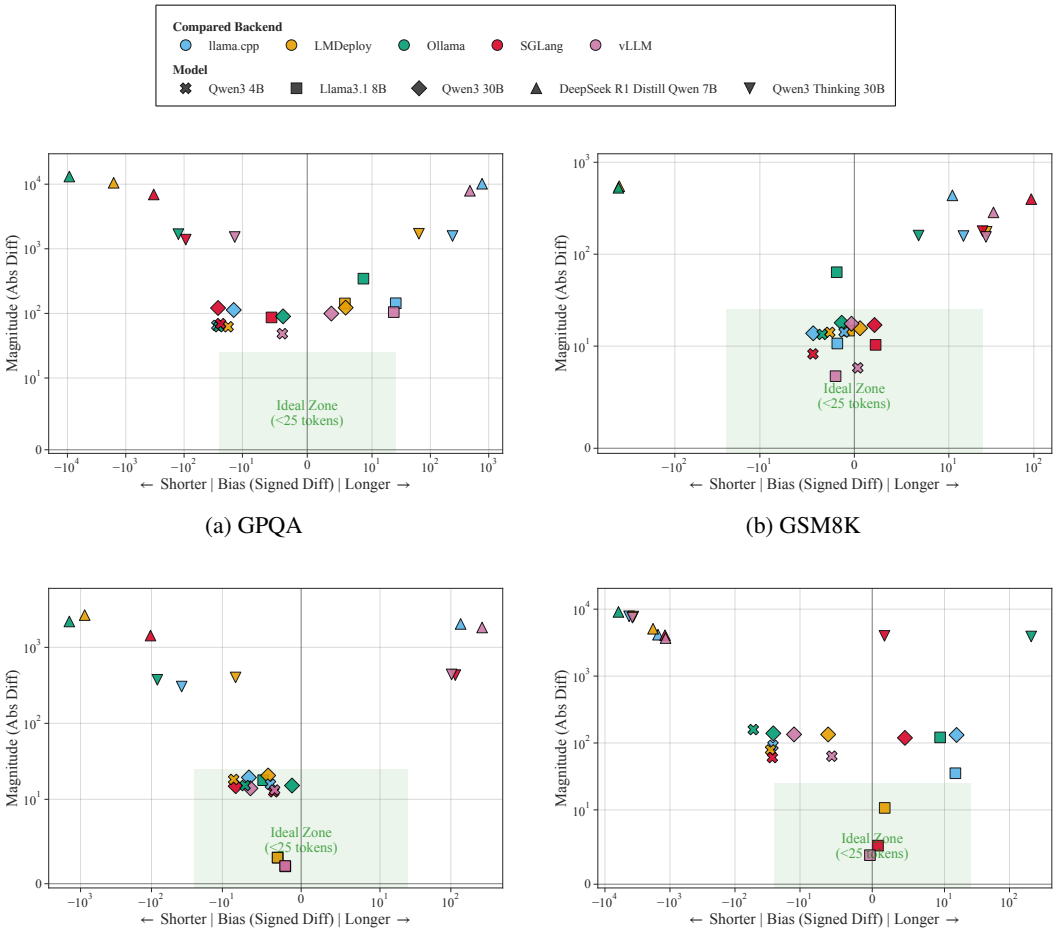
(a) GPQA (b) GSM8K

(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.

D.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.

D.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.

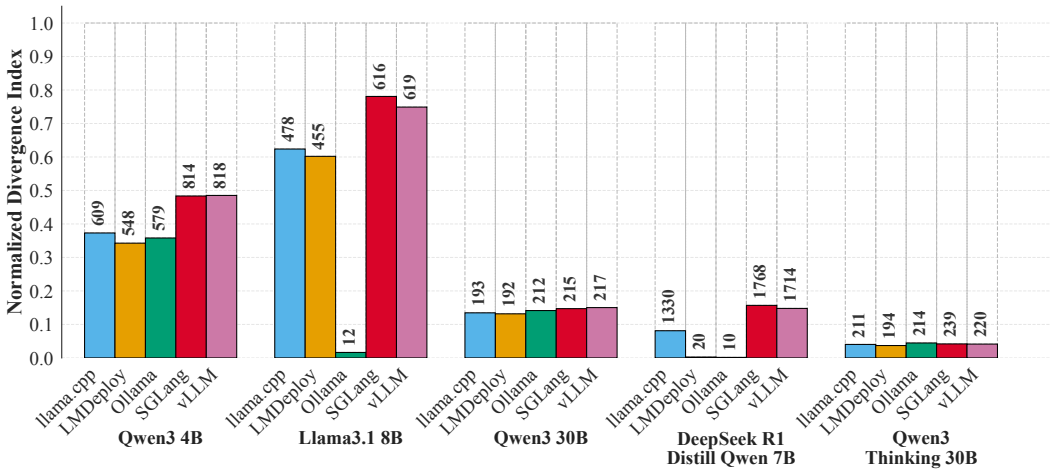


Figure 5: **Token-Level Divergence Analysis (GPQA).** Normalized divergence scores relative to the transformers reference. Larger values indicate high similarity (divergence happens late), while smaller values indicate early divergence. The labels above the bars indicate the average token position at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

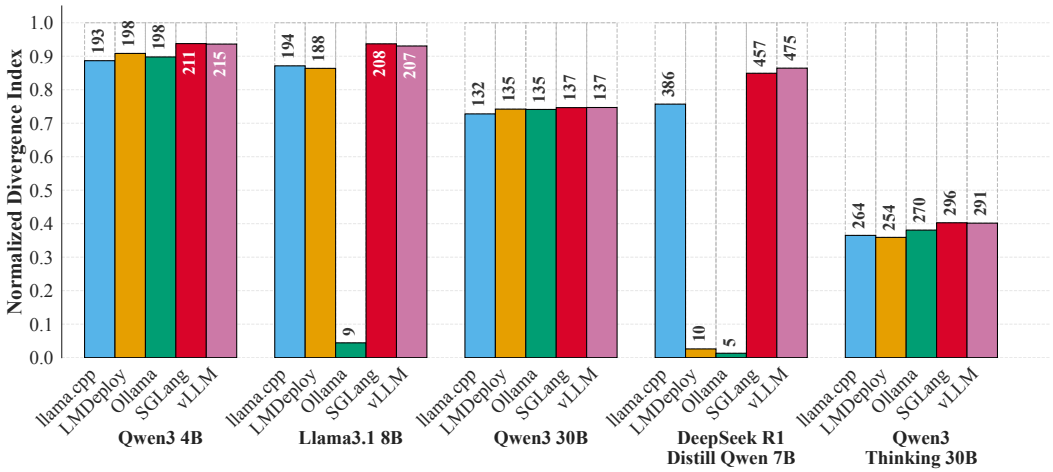


Figure 6: **Token-Level Divergence Analysis (GSM8K).** Normalized divergence scores relative to the transformers reference. Larger values indicate high similarity (divergence happens late), while smaller values indicate early divergence. The labels above the bars indicate the average token position at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

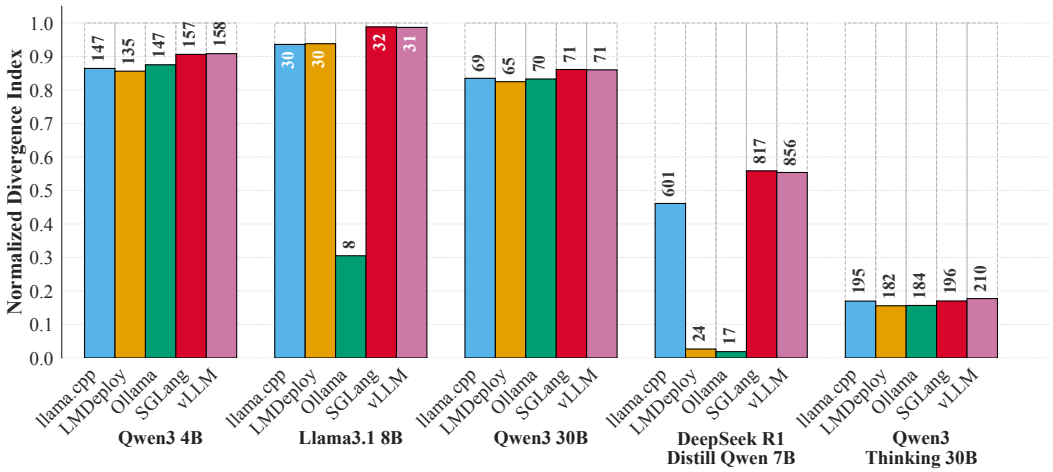


Figure 7: **Token-Level Divergence Analysis (SimpleQA).** Normalized divergence scores relative to the transformers reference. Larger values indicate high similarity (divergence happens late), while smaller values indicate early divergence. The labels above the bars indicate the average token position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

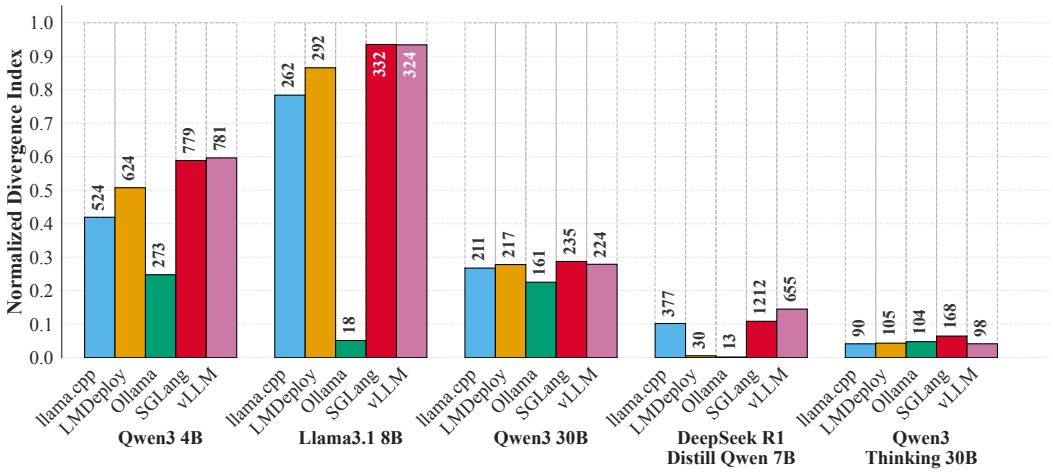
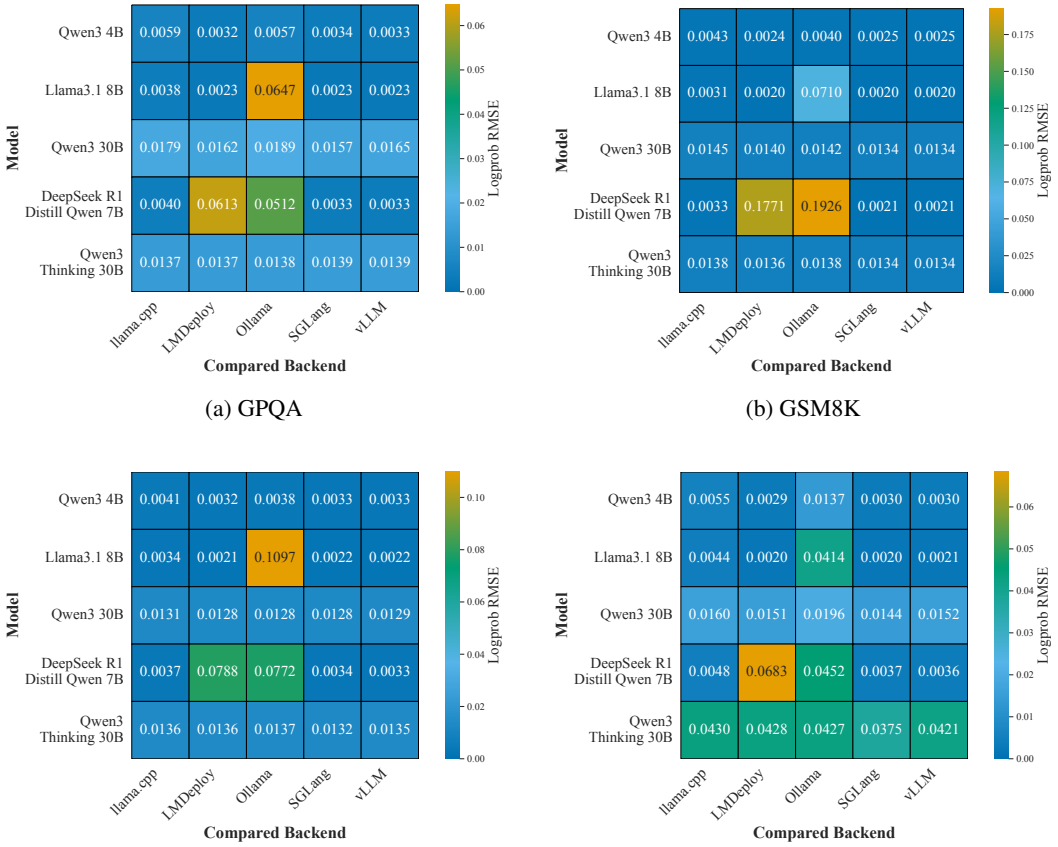


Figure 8: **Token-Level Divergence Analysis (LiveCodeBench).** Normalized divergence scores relative to the transformers reference. Larger values indicate high similarity (divergence happens late), while smaller values indicate early divergence. The labels above the bars indicate the average token position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

D.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.



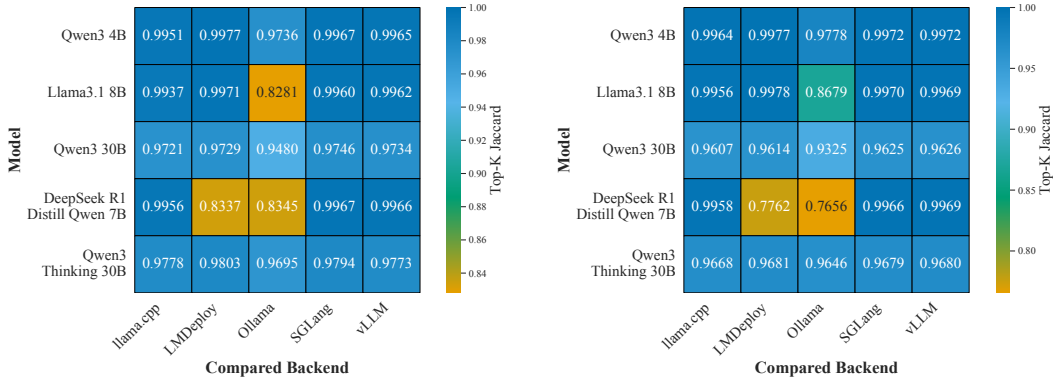
(a) GPQA (b) GSM8K

(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.

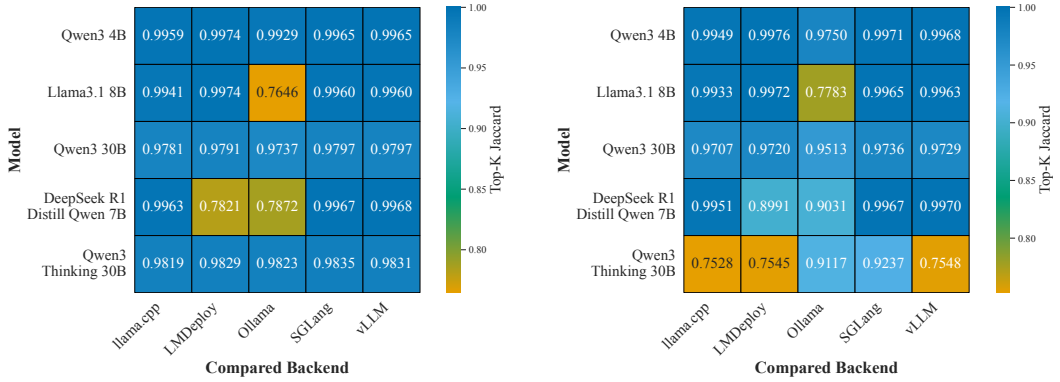
D.5 Top-5 Token Jaccard Similarity

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 metric (particularly in reasoning models) indicate that numerical instability is occasionally severe enough to completely reshape the distribution, pushing entirely 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.

E 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.

E.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 6, while Llama 3.1 8B is consistently robust, DeepSeek R1

Distill Qwen 7B's vulnerability fluctuates by 8.9% depending on the inference engine. This demonstrates that identical model weights can exhibit noticeably distinct safety profiles when deployed with different inference engines.

Table 6: **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 Distill Qwen 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

E.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. We omit llama.cpp and Ollama, because they do not natively support batched generation. As shown in Table 7, 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 7: **Ablation: Batched Generation (Batch Size = 4).** Accuracy (%) on GSM8K.

Model	transformers	LMDeploy	SGLang	vLLM	Max - Min
Llama 3.1 8B	<u>84.00</u> ± 0.00	84.15 ± 0.00	**84.22** ± 0.02	<u>84.00</u> ± 0.09	0.22
DeepSeek R1 Distill Qwen 7B	**78.62** ± 0.00	<u>74.07</u> ± 0.00	**78.62** ± 0.00	78.38 ± 0.20	4.55

E.3 Hardware Variation

To verify that our observations are not an artifact of our specific NVIDIA H100 (Hopper) setup, we repeated the evaluations on NVIDIA L40 GPUs (Ada Lovelace architecture). Table 8 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 8: **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 Distill Qwen 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

E.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 9, 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 from which samples are drawn remain fundamentally different, proving this variance is not an artifact of greedy decoding.

Table 9: **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** ± 00.70	81.00 ± 00.70	80.90 ± 01.00	<u>73.30</u> ± 01.00	81.20 ± 00.80	81.10 ± 00.80	08.70
DeepSeek R1 Distill Qwen 7B	**78.10** ± 01.00	77.40 ± 01.70	70.40 ± 02.90	<u>61.70</u> ± 01.10	77.90 ± 00.50	76.50 ± 03.90	16.30

F Root Cause Analysis

As established in Section 6, several of the large differences in benchmark performance across inference engines can be traced to specific system-level design choices. To isolate and quantify these sources of variance, we conduct targeted ablations on GSM8K and GPQA using Llama 3.1 8B, DeepSeek R1 Distill Qwen 7B, and Qwen3 4B. The complete results are reported in Tables 10 and 11.

Because inference backends differ in their architectures, default configurations, and optimization techniques, not every ablation applies to every engine or model. For example, Ollama applied hidden preprocessing and logit-processing defaults, whereas vLLM and SGLang primarily differ through high-throughput execution optimizations. Similarly, hidden BOS insertion is not applicable to Qwen3 4B because its chat template does not use a BOS token.

We broadly categorize these root causes into two groups:

1. **Systematic Engine Defaults:** These are correctable, engine-specific behaviors that alter the prompt or decoding procedure. Hidden BOS insertion and default repetition penalties produce the largest effects in our ablations when applicable. On GSM8K, removing hidden BOS tokens improves accuracy by up to 8.34 percentage points, while disabling Ollama's repetition penalty improves DeepSeek R1 Distill Qwen 7B by 11.67 points. The GPQA experiments show the same mechanisms can remain consequential across tasks, although their magnitude depends on the model and backend.

2. **Optimization-Induced Numerical Variance:** Even after prompt construction and decoding defaults are aligned, backend-specific execution paths introduce numerical differences. Prefix caching, CUDA graphs, and custom greedy decoding kernels can alter floating-point computation and subsequent token selection. In contrast to systematic defaults, their effects vary in both magnitude and direction across configurations. In our experiments, these ablations range from no net accuracy change to shifts of 3.54 percentage points on GPQA.

Tables 10 and 11 report the baseline accuracy and the results after ablating each feature. Importantly, an unchanged aggregate accuracy does not imply identical model behavior: even configurations with $\Delta = 0$ exhibit sample-level disagreement with their corresponding baseline.

Table 10: **Root Cause Ablation Results on GSM8K.** Performance impact of isolating specific system-level defaults and hardware optimizations. Metrics are reported as Accuracy (%). The Δ columns report the percentage-point change relative to the original configuration.

Inference Engine	Ablated Feature / Optimization	Llama 3.1 8B			DeepSeek R1 Distill Qwen 7B			Qwen3 4B		
		Original	Ablated	Δ	Original	Ablated	Δ	Original	Ablated	Δ
llama.cpp	Disable Prefix Caching	84.15	84.08	−0.07	78.24	78.70	+0.46	90.45	90.67	+0.22
LMDeploy	Patch Argmax Kernel Tie-Breaking	84.53	84.08	−0.45	73.54	73.62	+0.06	90.45	90.45	0.00*
	Remove Hidden BOS Token	–	–	–	73.54	78.24	+4.70	–	–	–
Ollama	Disable Prefix Caching	74.30	73.24	−1.06	61.87	61.41	−0.47	90.75	90.52	−0.23
	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	90.75	90.75	0.00*
SGLang	Disable Prefix Caching	84.23	83.85	−0.48	78.24	78.54	+0.30	90.45	90.30	−0.15
	Disable CUDA Graphs	84.23	84.23	0.00*	78.24	78.24	0.00*	90.45	90.45	0.00*
vLLM	Disable Prefix Caching	83.85	84.31	+0.46	78.32	78.32	0.00*	90.47	90.67	+0.20
	Disable CUDA Graphs	83.85	84.00	+0.15	78.32	78.47	+0.15	90.47	90.60	+0.13

† Ollama enforces a default repetition penalty of 1.1; disabling it sets the penalty to 1.0.
* A zero net accuracy shift does not imply identical predictions; these configurations still exhibit sample-level disagreement.

Table 11: **Root Cause Ablation Results on GPQA.** Performance impact of isolating specific system-level defaults and hardware optimizations. Metrics are reported as accuracy (%). The Δ columns report the percentage-point change relative to the original configuration. GPQA contains 198 examples, such that one net change in correctness corresponds to approximately 0.51 percentage points.

Inference Engine	Ablated Feature / Optimization	Llama 3.1 8B			DeepSeek R1 Distill Qwen 7B			Qwen3 4B		
		Original	Ablated	Δ	Original	Ablated	Δ	Original	Ablated	Δ
llama.cpp	Disable Prefix Caching	23.74	23.23	-0.51	31.82	31.31	-0.51	38.89	38.38	-0.51
LMDeploy	Patch Argmax Kernel Tie-Breaking	25.76	25.76	0.00^*	35.86	32.32	-3.54	35.35	34.85	-0.50
	Remove Hidden BOS Token	–	–	–	35.86	35.86	0.00^*	–	–	–
Ollama	Disable Prefix Caching	22.22	23.23	$+1.01$	27.27	30.81	$+3.54$	37.88	36.87	-1.01
	Remove Hidden BOS Token	22.22	26.77	$+4.55$	27.27	29.29	$+2.02$	–	–	–
	Disable Repetition Penalty[†]	22.22	24.24	$+2.02$	27.27	28.79	$+1.52$	37.88	37.88	0.00^*
SGLang	Disable Prefix Caching	23.23	23.74	$+0.51$	31.31	30.81	-0.50	34.85	35.86	$+1.01$
	Disable CUDA Graphs	23.23	24.75	$+1.52$	31.31	31.31	0.00^*	34.85	34.85	0.00^*
vLLM	Disable Prefix Caching	25.76	25.76	0.00^*	33.84	34.34	$+0.50$	34.81	34.85	$+0.04$
	Disable CUDA Graphs	25.76	25.25	-0.51	33.84	32.32	-1.52	34.81	35.86	$+1.05$

[†] Ollama enforces a default repetition penalty of 1.1; disabling it sets the penalty to 1.0.
[*] A zero net accuracy shift does not imply identical predictions; these configurations still exhibit sample-level disagreement.

G 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.

G.1 Pipeline Methodology

Corpus Collection. We retrieved accepted papers from NeurIPS, ICML, and ICLR through the OpenReview API and from ACL and EMNLP through the ACL Anthology API, covering publication years 2023-2025. This produced the initial corpus of more than 35,000 PDFs used by the filtering pipeline below.
To efficiently and accurately process this initial corpus, 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 raw extracted text (using `pymupdf` Python library) of all PDFs for specific terms related to open-weight models and local execution (see Section G.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.

G.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",
]
```

G.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>
```

G.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"
]
```

H 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.

I 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.

J Model and Dataset Licenses

Table 12 summarizes the licenses and terms of use associated with the models and datasets used in our study.

Table 12: LLMs and Datasets used in this paper alongside their licenses.

Model/Dataset	**License**
meta-llama/Meta-Llama-3.1-8B-Instruct [27]	Llama Community License
Qwen/Qwen3-4B-Instruct-2507 [28]	Apache-2.0
Qwen3-30B-A3B-Instruct-2507 [28]	Apache-2.0
deepseek-ai/DeepSeek-R1-Distill-Qwen-7B [29]	MIT License
Qwen/Qwen3-30B-A3B-Thinking-2507 [28]	Apache-2.0
AIDXteam/Qwen3-235B-A22B-Instruct-2507-AWQ [36]	Not specified (base model: Apache-2.0)
gpt-4o-mini-2024-07-18 [40][3]	OpenAI Terms of Use
GSM8K [30]	MIT License
GPQA Diamond [31]	CC BY 4.0
SimpleQA Verified [32]	Apache-2.0
LiveCodeBench v6 [33]	CC (exact variant not specified)
JailbreakBench [35]	MIT License

[3] Used as LLM judge for SimpleQA and JailbreakBench

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The claims about backend-induced benchmark variance are supported by Sections 5- 6, while the claim that inference stacks are rarely documented is supported by Section 7.

   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: [Yes]

   Justification: The paper includes a dedicated "Limitations" section (Section 8.2) that discusses the constraints of the controlled environment and the reliance on greedy decoding.

   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: [N/A]

Justification: The paper presents an empirical systems/reproducibility study and does not introduce new theoretical results or proofs.

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: [Yes]

Justification: Section 4, Section 5, and Appendix A detail the experimental setup, including fixed hyperparameters, specific backend software versions, FP16 precision, and exact prompt templating strategies.

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: [Yes]

   Justification: We release all code, artifacts, and commands to reproduce the environment and results at `https://github.com/Dormant-Neurons/inference_backends`.

   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: [Yes]

   Justification: Section 4.2 and Section 5 detail all relevant evaluation settings, including context windows, max output tokens, batch sizes, and the use of twelve unique seeds. No model training was performed.

   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: [Yes]

   Justification: Results are reported as mean $\pm$ one standard deviation over 12 runs with distinct seeds, as shown in the main results (Table 1) and ablation tables (e. g., Table 6). The standard deviation is computed over run-level benchmark scores.

   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: [Yes]

   Justification: Section 5 and Appendix E.3 specify the hardware required to reproduce the experiments (a single NVIDIA H100 GPU with 94GB VRAM, and NVIDIA L40 GPUs).

   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: [Yes]

   Justification: The research involves evaluating publicly available, open-weight models on standard benchmarks and surveying published literature, which conforms with the NeurIPS Code of Ethics.

   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: [Yes]

    Justification: Discussed in Appendix I.

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: [N/A]

    Justification: The paper does not release any new models, image generators, or scraped datasets that carry a high risk of misuse.

    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: [Yes]

    Justification: All existing assets are cited, and their terms of use and licenses have been respected. License information is mentioned in Appendix J.

    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: [N/A]

    Justification: The paper conducts an empirical evaluation using existing models, datasets, and inference engines, and does not introduce new datasets or models.

    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: [N/A]

    Justification: No human subjects were involved.

    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: [N/A]

    Justification: The research does not involve human subjects.

    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: [Yes]

    Justification: The paper declares the use of an LLM-as-a-judge (Qwen3-235B) to categorize research papers for the literature survey in Section 7, and provides the exact methodology and prompts used in Appendix G.

    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.