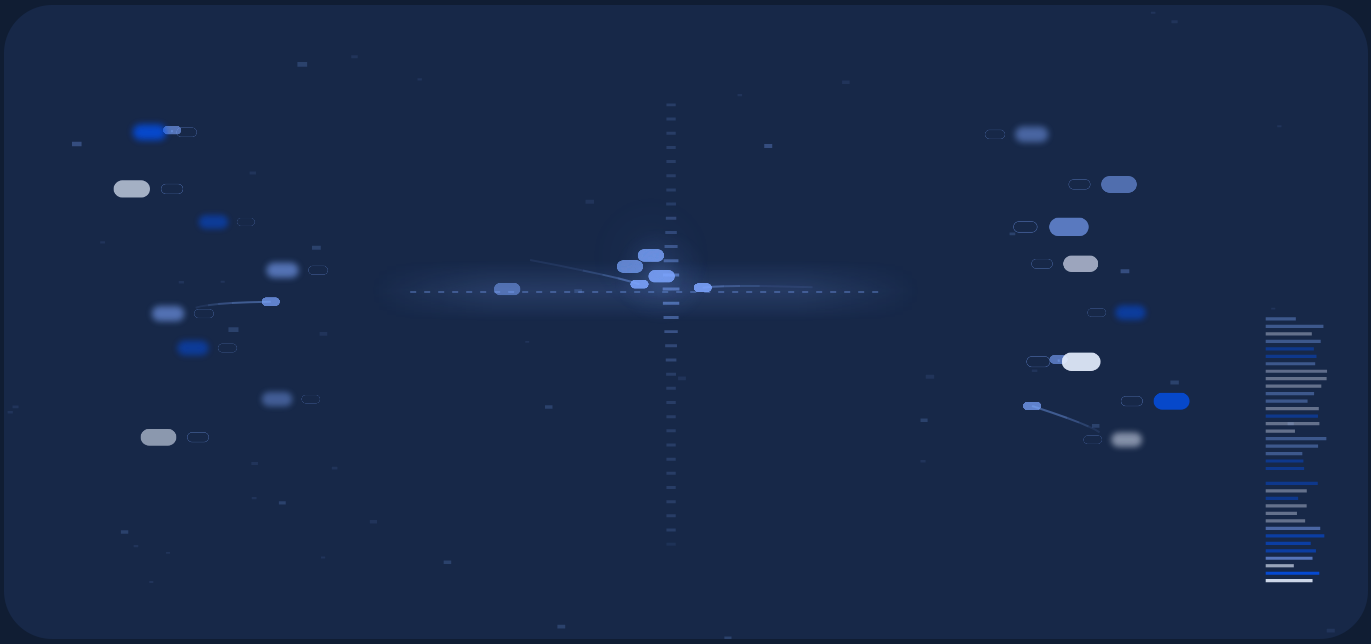
Rillor

EXPLAINER

LLM nondeterminism: why the same input gets different answers

Why the same request to an LLM can return different answers even at temperature 0: sampling, floating-point order, server load, hardware and serving changes under one model ID.



Rillor

Contents

01 Summary

02 Background

03 Method

04 Findings

05 Implications

06 Limits of this analysis

07 Sources

Summary

An LLM can return different answers to an identical request even with random sampling switched off, because of how inference servers compute, batch and change the serving code around the model [1, 3, 11, 24].

- Temperature 0, the setting that makes a model take its most probable token at every step, narrows variation without removing it. In a September 2025 experiment, 1,000 temperature-0 runs of one prompt on an open-weights model produced 80 different completions [1], and model providers' own documentation says identical requests can still return different outputs [12, 15].
- Repeated runs at fixed settings move measured accuracy. A 2024 study of five models from different families, each run ten times at temperature 0 with a fixed seed, found accuracy varying by up to 15% between runs [2].
- Floating-point addition can round differently when its order changes, and inference servers change that order with the number of requests they process together [1, 20, 33]. In a September 2025 experiment, GPU routines (kernels) that keep the order fixed whatever the batch size, which its authors call batch-invariant, made all 1,000 temperature-0 completions of one prompt identical [1].
- Hardware and number formats move results too. In a NeurIPS 2025 study, changing GPU type, GPU count and batch size under the 16-bit bfloat16 format, with the most probable token taken at every step, gave a distilled reasoning model a standard deviation of up to 9% in accuracy, against virtually zero in 32-bit floating point [3].
- A pinned model ID fixes the weights and leaves the serving stack free to change. One provider's documentation says its request router, safety classifiers and sampling logic can change under a fixed ID [11], and its September 2025 postmortem describes infrastructure bugs that changed outputs for some requests, one only at certain batch sizes [24].
- Deterministic inference is available in open engines and costs speed. The SGLang engine's deterministic mode returned one unique output in every test, with most offline slowdowns between 25% and 45% [21]. The hosted API documentation reviewed calls seeds best effort or says determinism is not guaranteed [13, 14, 15].
- A single run is one sample. The sources recommend repeated runs with error bars, full records of prompts, settings and model versions, and separate tests for rephrased inputs, since formatting changes alone moved one open-weights model by up to 76 accuracy points [3, 7, 8, 9, 30].

Background

What counts as the same input

A caller controls the prompt, the model ID and the sampling settings: temperature, top-p and, where offered, a seed. Nondeterminism, as used here, is variation in the output when all of those stay the same. The caller does not control the other requests a server processes at the same moment, the GPU type and count the request lands on, the kernel and library versions that run it, or the router and sampling code around the model [1, 3, 11, 18]. The September 2025 post "Defeating Nondeterminism in LLM Inference" calls the other concurrent users, from one user's side, "a nondeterministic property of the system" [1].

Two neighbouring effects are kept apart. A rephrased or reformatted prompt is a different input: in an ICLR 2024 study, formatting changes alone moved a 13-billion-parameter open-weights model by up to 76 accuracy points in few-shot settings, and the authors recommend reporting a range across plausible formats [7]. A new model version is also a different input. Two dated versions of one widely used hosted model, three months apart in 2023, scored 84% and 51% on the same questions about prime and composite numbers [6]. Both matter to evaluation, and neither is run-to-run nondeterminism.

How a model picks a token

At each step a language model assigns a probability to every token in its vocabulary, and a decoding rule picks one. In the Transformers library, greedy search is the default and selects the most likely next token. Multinomial sampling, switched on with `do_sample=True`, gives every token with a non-zero probability a chance to be selected [28]. Temperature reshapes the distribution. One provider's API reference accepts values between 0 and 2, defaults to 1, and describes 0.8 as more random and 0.2 as more focused and deterministic [14]. Top-p, or nucleus sampling, keeps only the tokens in a set share of the probability mass: 0.1 means the top 10%, and the reference recommends changing temperature or top-p, generally not both [14]. Nucleus sampling comes from a 2019 paper that found decoding strategies alone can dramatically change the quality of text from the same model [27].

Temperature 0 is greedy decoding, which makes sampling deterministic in theory [1, 15]. The findings below explain why practice differs.

Why it matters

Evaluations of language models often score a single output per example [4], and an agent task that passed on one run might fail on the next [31]. Article 15 of the EU AI Act requires high-risk AI systems to perform consistently in accuracy, robustness and cybersecurity throughout their lifecycle [29]. NIST AI 600-1 lists the variability of performance over time among the factors for risk tiers and asks that system inventories record the versions of underlying models [30].

This report is for engineers, evaluators and product owners who test or deploy LLM agents and need to know where variation comes from before they read a score.

Method

What was examined. The causes of different outputs for an identical request: sampling, temperature 0, floating-point order, batching, hardware and number formats, tensor-parallel size, mixture-of-experts routing, and changes to the serving stack under one model ID. Also the controls offered against them, their measured costs, and what the sources say about evaluation.

Sources. Thirty-five: studies and preprints that measure variation or its causes [2, 3, 4, 5, 6, 7, 32, 33, 34, 35], the September 2025 post on batch invariance [1], papers on sampling, routing and evaluation [8, 9, 10, 25, 26, 27], model providers' documentation, guides and postmortem [11, 12, 13, 14, 15, 16, 17, 24, 31], library and engine documentation [18, 19, 20, 21, 22, 23, 28], and the EU AI Act and NIST AI 600-1 [29, 30].

How. Figures were read in each source's full text and are given with their model description, settings and date. No experiments were run for this report. Figures from different studies use different metrics and are never combined. Web pages were viewed on 7 November 2026.

Terms. Greedy decoding takes the most probable token at each step. A batch is the set of requests a server processes together. BF16 (bfloat16) and FP16 are 16-bit floating-point formats, and FP32 is the 32-bit format. Tensor-parallel size is the number of GPUs that share the computation of one model.

Findings

1. Sampling is random by design: temperature, top-p and seeds

At default settings, variation is intended. One provider's API reference sets the default temperature at 1 [14], and another provider uses a random seed unless the caller fixes one [15].

The spread at those defaults is wide. In a study of a widely used hosted chat model on 829 code-generation problems from CodeContests, APPS and HumanEval, five requests per problem at temperature 1, the share of problems where the five programs produced no equal test output was 75.76%, 51.00% and 47.56% [5]. In a January 2026 preprint, exact repeats of the same prompts at temperature 0.7 gave a mean unique output fraction of 0.987 for both a hosted model and an 8-billion-parameter open-weights model run locally: almost every run yielded a new answer [32].

Sampling also changes scores. A July 2024 study that drew 16 to 128 samples per instance found greedy decoding generally ahead of sampling on most tasks, with math reasoning and code generation the most affected by sampling variance and constrained-output benchmarks such as MMLU consistent across configurations [4]. In certain cases the ranking of models changed with the configuration, and alignment can reduce sampling variance [4].

2. Temperature 0 narrows variation and does not remove it

Model providers say plainly that greedy settings are no guarantee. One provider's glossary states that results are not fully deterministic even with temperature set to 0, and that identical inputs might produce different outputs across API calls, on its own service and through third-party cloud providers [12]. Another calls temperature-0 responses mostly deterministic, with a small amount of variation still possible [15]. The September 2025 post adds that self-hosted inference with an open-source library such as vLLM or SGLang is not deterministic at temperature 0 either [1].

Study	Models and setting	Measure	Result	Source
September 2025 experiment	An open-weights model, 1,000 runs of one prompt, temperature 0	Distinct completions of 1,000 tokens	80	[1]
2024 five-model study	One widely used hosted model, college math, ten runs, temperature 0, top-p 1, fixed seed	Questions with the same raw string in all ten runs	0.0%	[2]
Code-generation study, 2023 and 2024	A widely used hosted chat model, five requests per problem, temperature 0	Problems with no equal test output on CodeContests, APPS and HumanEval	43.64%, 27.40%, 18.29%	[5]

Study	Models and setting	Measure	Result	Source
January 2026 preprint	A hosted model and a local 8B open-weights model, exact repeats, temperature 0	Unique output fraction	0.240 and 0.093	[32]

Table 1. Variation measured at temperature 0. Each figure is the study's own metric and is not comparable with the others.

The code-generation study's authors conclude that temperature 0 brings less nondeterminism than the default without guaranteeing determinism [5].

The 2024 five-model study ran eight tasks, zero-shot and few-shot, with the same compute infrastructure, inputs and configurations [2]. Accuracy varied by up to 15% across runs [2]. For the widely used hosted model on few-shot college math, the best possible accuracy over ten runs (a question counts as correct if any run got it right) was 88.0, the median accuracy 69.0 and the worst possible accuracy 44.0, and parsed answers agreed across all ten runs on 50.0% of questions [2]. For one open-weights model on the same task, the best possible and worst possible accuracy over the ten runs were 75% and 3% [2]. Models were far more stable at the parsed answer level than at the string level, and a downstream parser can be affected significantly when the format changes [2]. Accuracy distributions over 20 runs were not normal (Kolmogorov-Smirnov $p < 1e-9$), and the authors recommend reporting maximum and minimum scores across runs [2].

3. Floating-point addition depends on order

A floating-point operation that complies with IEEE 754 gives the same result on every compliant implementation for a given format and rounding mode, but the rules of ordinary arithmetic do not carry over. CUDA's floating-point white paper shows that rounding A + B and then adding C can differ from adding A to the rounded B + C, so seemingly identical computations can produce different results [20]. Results also depend on fused multiply-add and on whether additions run in series or in parallel, and changing the number of threads per block rearranges the parentheses in a long string of sums [20].

Order alone does not make a fixed program change its answer on every run. The September 2025 post presents the common explanation, in which whichever concurrent core finishes first changes the order of additions, and argues that it is incomplete: all the kernels in a language model's forward pass are run-to-run deterministic, and the same matrix multiplication on the same data always gives bitwise equal results [1].

The libraries underneath state narrow guarantees. cuBLAS generates the same bit-wise results at every run for a given toolkit version on GPUs with the same architecture and the same number of SMs (streaming multiprocessors), with no guarantee across toolkit versions. The guarantee no longer holds with multiple CUDA streams active or fixed-point emulation, and routines that use atomics are not guaranteed to be bit-wise reproducible [18]. PyTorch does not guarantee completely reproducible results across releases, commits

or platforms, or between CPU and GPU even with identical seeds. Its deterministic operations are often slower, and `torch.use_deterministic_algorithms()` raises an error where no deterministic version of an operation exists [19].

4. Batch invariance: output depends on what else the server is processing

The September 2025 post finds the missing piece in batching [1]. A kernel can be run-to-run deterministic and still lack batch invariance: its result for one request changes with the size of the batch the request is processed in [1]. A January 2026 preprint describes the same mechanism as floating-point non-associativity combined with dynamic batching and GPU kernels whose reduction orders vary with batch size [33]. Batch size depends on how many requests arrive at a given moment, so the post's authors argue that varying load is the primary reason nearly all LLM inference endpoints are nondeterministic, on CPU and TPU endpoints as well as GPUs [1]. For hosted models, the 2024 five-model study says it can only speculate about the cause [2].

The experiment ran an open-weights model 1,000 times at temperature 0 on the prompt "Tell me about Richard Feynman", 1,000 tokens each [1]. It produced 80 unique completions, the one seen most often appearing 78 times [1]. Every completion matched for the first 102 tokens [1]. At token 103, 992 continued with "Queens, New York" and 8 with "New York City" [1].

The fix is to make every kernel batch-invariant. The post treats RMSNorm, matrix multiplication and attention, and requires a reduction order for each element that stays fixed regardless of the kernel's batch size [1]. With its batch-invariant kernels enabled, all 1,000 completions were identical [1]. In reinforcement learning, batch-invariant inference kept the KL divergence between sampler and trainer flat at 0 [1].

Other serving features may play a part. The 2024 five-model study lists continuous batching, chunked prefill and prefix caching as optimizations that "might lead to non-deterministic behavior" [2]. The January 2026 preprint notes that switching off dynamic batching severely degrades throughput [33].

5. GPU type, GPU count, precision and parallelism move results

A NeurIPS 2025 study measured how system configuration changes outputs for reasoning and non-reasoning open-weights models of 7 to 8 billion parameters [3]. It ran 12 configurations: two GPU types (L40S and A100), two GPU counts (2 and 4) and three batch sizes (8, 16 and 32), mainly in vLLM [3]. Its random-sampling runs used temperature 0.7 and top-p 0.95 [3].

The number format decided how much those configurations mattered. FP32 achieved near-perfect reproducibility, FP16 showed moderate variability and BF16 substantial instability [3]. Under BF16 with greedy decoding, a 7-billion-parameter reasoning model distilled from a larger one showed up to 9% standard deviation in accuracy on AIME'24, against virtually zero under FP32, and a difference of up to 9,000 tokens in response length [3]. The authors trace this to floating-point non-associativity under limited precision: in reasoning models, minor rounding differences in early tokens can cascade into divergent chains of thought [3]. Their fix, LayerCast, stores weights in 16 bits and computes in FP32 [3].

Tensor-parallel size adds another variable. A paper first posted in November 2025 found that identical inputs can yield different outputs when tensor-parallel size or batch size varies, even under greedy decoding, because reduction orders differ across GPUs [34]. It reports that batch-invariant kernels left this an open problem, and that its tree-based kernels give bit-wise identical results regardless of tensor-parallel size, confirmed in vLLM and FSDP [34]. vLLM's documentation warns that under tensor parallelism, all-reduce operations whose size is not a multiple of 16 bytes can switch between NCCL and a custom all-reduce depending on batch size [22].

6. Mixture-of-experts routing under capacity limits

Sparse mixture-of-experts models have a mechanism of their own. An ICLR 2024 paper states that under capacity constraints, sparse approaches route tokens in groups of a fixed size and balance load within each group [25]. When a group holds tokens from different sequences, they compete for spots in expert buffers, so the model is deterministic only at the batch level and no longer at the sequence level [25]. The paper lists token dropping among the problems of sparse models [25]. The Switch Transformers paper sets each expert's capacity from the total tokens, the number of experts and a capacity factor, and notes that when tokens are dispatched unevenly, the tokens that overflow an expert are not processed by that layer [26].

Whether a token reaches an expert can therefore depend on the other inputs grouped with it. No source reviewed shows this mechanism at work in any named or hosted model. The code-generation study lists it only as one hypothesis, alongside floating point and unreliable GPU calculations, and calls the cause at temperature 0 still controversial [5].

7. A fixed model ID does not fix the serving stack

Pinning a model ID fixes the weights. One model provider's versioning documentation states that each model ID identifies a pinned version, that for its newer generations a dateless ID maps to a single fixed snapshot, and that it does not update the weights or configuration of an existing ID. Its older dateless aliases resolve to the most recent dated snapshot [11]. The request router, safety classifiers and sampling logic around the model can change over time, and infrastructure updates occasionally produce minor differences in behavior when the ID and weights have not changed [11].

Aliases move further. Another provider's stable model names usually do not change, while its latest aliases are hot-swapped with every new release, with two weeks' notice [17]. A third provider's 2023 guide described a system fingerprint for the current combination of weights, infrastructure and configuration, which changes when the provider updates the numerical configuration of its serving infrastructure, something that "may happen a few times a year" [13].

Infrastructure changes can also be faults. One model provider's September 2025 postmortem describes three infrastructure bugs that intermittently degraded response quality between August and early September 2025, and states that it never reduces model quality due to demand, time of day or server load [24].

- **Context window routing.** A routing error affected 0.8% of one model's requests at first and 16% at the worst affected hour, on 31 August [24]. The routing was sticky [24].
- **Output corruption.** A runtime performance optimization on TPU servers corrupted outputs from 25 August to 2 September [24].
- **Sampling and the compiler.** The sampler applies top-p, typically 0.99 or 0.999 [24]. In December 2024 its TPU implementation occasionally dropped the most probable token at temperature 0, because of mixed bf16 and fp32 precision under an XLA compiler flag [24]. An approximate top-k operation sometimes returned completely wrong results, only for certain batch sizes and model configurations [24].

In the postmortem's words, "The same prompt might work perfectly on one request and fail on the next" [24]. The provider's evaluations did not capture the degradation users reported, and its response is to run evaluations continuously on true production systems [24].

Version changes are a separate effect, and they can be larger. In 2023, one widely used hosted model identified prime and composite numbers with 84% accuracy in its March version and 51% in its June version, and the authors conclude that the same hosted service can change substantially in a short time [6].

8. What seeds, fingerprints and sampling controls promise

Control	What the source says	Source
Seed and fingerprint, one provider's 2023 guide	With the same seed, parameters and fingerprint, outputs are mostly identical, with a small chance that responses differ	[13]
Seed and fingerprint, the same provider's current API reference	Both marked deprecated. The seed is a best effort, and determinism is not guaranteed	[14]
Seed, another provider	Best effort, no guarantee. A change of model or settings such as temperature can change the response under the same seed. Random by default	[15]
Temperature, top-p and top-k, one provider's newest families	Custom values ignored	[16]
vLLM seed	Defaults to 0, setting the random state of each worker, so results stay consistent for each vLLM run even above temperature 0. Reproducibility itself is not guaranteed by default	[23]
SGLang seeded sampler	The same inputs and seed always yield the same sample, even above temperature 0	[21]

Table 2. What each control promises, in each source's terms.

No provider source reviewed promises identical output for identical requests [12, 13, 14, 15]. SGLang's seeded sampler is part of its deterministic mode (finding 9), and in online serving vLLM's only reproducibility setting is batch invariance [21, 23].

9. Deterministic inference in open engines, and its cost

The September 2025 post measured its own cost on one GPU with an 8-billion-parameter open-weights model generating 1,000 sequences of 90 to 110 tokens [1]. Default vLLM took 26 seconds, its unoptimized deterministic version 55 seconds and the version with an improved attention kernel 42 seconds, which the authors called "not disastrous" [1].

The SGLang team published deterministic inference on 22 September 2025, built on the post's batch-invariant operators and compatible with chunked prefill, CUDA graphs, radix cache and non-greedy sampling [21]. Over 50 sampling trials on an 8-billion-parameter open-weights model, normal mode gave up to 18 unique outputs and deterministic mode 1, in every test [21]. In offline tests on an H200 with tensor-parallel size 1 and 256 requests, most slowdowns ranged from 25% to 45%, averaging 34.35%, against 61.5% for the original post's implementation [21]. The team recommends the mode primarily for debugging and reproducibility, and its post, dated 22 September 2025 and updated on 24 September, sets a goal of bringing the gap under 20%, or ideally to parity [21].

vLLM enables batch invariance with `VLLM_BATCH_INVARIANT=1` on NVIDIA GPUs of compute capability 8.0 or higher, listed AMD Instinct GPUs and Intel XPUs with Triton [22]. Its documentation defines the feature as output independent of the batch size or the order of requests in a batch, and warns that it may impact performance. The page, dated 4 November 2026, labels the feature beta [22]. The reproducibility page, dated 7 October 2026, states that vLLM does not guarantee reproducible results by default, for the sake of performance [23].

A January 2026 preprint, LLM-42, argues that batch-invariant kernels impose fixed overheads however much of the workload needs determinism, and measured SGLang's deterministic mode at up to 56% overhead [33]. LLM-42 decodes on the fast path and enforces determinism with a verify-rollback loop only for requests that ask for it, off by default. With 1 in 11 requests needing determinism, it reached 911 tokens per second, 2.2 times SGLang's deterministic throughput and within 3% of the best-case throughput of non-deterministic mode [33].

Approach	Test	Cost reported	Source
Batch-invariant kernels in vLLM, September 2025	One GPU, 8B model, 1,000 sequences	26 s default, 55 s unoptimized, 42 s with improved attention	[1]

Approach	Test	Cost reported	Source
SGLang deterministic mode, September 2025	H200, tensor-parallel size 1, 256 requests	Most slowdowns 25% to 45%, average 34.35%	[21]
SGLang deterministic mode, measured by LLM-42	The preprint's workload	Up to 56% overhead	[33]
LLM-42 per-request determinism, January 2026	1 in 11 requests deterministic	Within 3% of non-deterministic best case	[33]

Table 3. Each cost comes from its own test and is not comparable with the others.

10. What nondeterminism does to evaluation and to agents

Single runs. The July 2024 study found that evaluations often overlook nondeterminism by scoring one output per example [4]. In the 2024 five-model study, none of the models consistently delivered repeatable accuracy across all tasks, much less identical output strings [2].

Error bars. A November 2024 paper recommends standard errors, clustered when questions come in related groups, resampling answers to reduce variance, paired comparisons between models and power analysis [8]. In its worked example with binary scores, going from one answer per question to two reduces variance by a third and to four by half, with an upper limit of two-thirds [8]. It advises against lowering the temperature to reduce variance unless the purpose is to study the model at the new temperature: in its example, temperature 0 tripled the minimum variance, from 1/12 to 1/4 [8]. The NeurIPS 2025 study recommends multiple runs at a non-zero temperature, with mean accuracy, average answer length and error bars, and FP32 when a single greedy run is used [3].

Records. A paper drawn from three years of developing the lm-eval harness asks evaluators to provide evaluation code, full prompts and methodology, including generation hyperparameters and answer extraction heuristics, and notes that full prompts are necessary but not sufficient for perfect reproducibility [9]. NIST AI 600-1 asks inventories to record the versions of underlying models [30].

Agents. A January 2026 guide from one model provider notes that agent behavior varies between runs, and lists non-determinism as a weakness of model-based graders [31]. pass^k, the probability that all k trials succeed, makes this visible: at 75% success per trial it is about 42% over three trials [31]. tau-bench, which introduced pass^k in June 2024, found a leading function-calling agent quite inconsistent, with pass^8 below 25% in retail [10]. An April 2026 single-author preprint ran six models from three providers on 19 tasks with deterministic tools, 10 runs per task at temperature 1.0 [35]. Agents reliably selected the same tools in the

same order, with a mean tool-sequence similarity of 0.87, and varied substantially in their arguments, with a mean argument consistency of 0.69 [35]. Sixty percent of the divergence originated in the first two pipeline steps [35].

Implications

For teams evaluating agents. Treat one run as one sample. Run each case several times, report the mean with error bars [3, 8], and give the maximum and minimum where runs are few [2]. Score at the temperature the agent runs at, since lowering it only to reduce variance is what the error-bars paper advises against [8]. Record the prompts, code, generation settings and model version behind every score [9, 30]. Score parsed answers as well as raw strings [2], and test rephrasings and new model versions as separate inputs [6, 7].

For teams building on hosted models. Every provider source reviewed describes determinism as best effort or not guaranteed [12, 13, 14, 15], so parsers and downstream checks need to tolerate variation in format [2]. A pinned model ID fixes the weights and leaves the serving stack free to change, and an alias can move to a new model [11, 17]. One provider's response to its own incidents was to run evaluations continuously on production systems [24].

For teams serving open-weights models. Deterministic modes in vLLM and SGLang reproduce outputs at a cost in speed, which SGLang measured at 25% to 45% in most offline tests [21, 22]. vLLM promises reproducibility only on the same hardware and the same vLLM version [23]. FP32 gave near-perfect reproducibility in the NeurIPS 2025 study [3]. Research systems make determinism a per-request choice or extend it across tensor-parallel sizes [33, 34].

For governance. The EU AI Act asks high-risk systems to perform consistently throughout their lifecycle [29], and NIST AI 600-1 names variability over time as a risk factor [30]. This report is research and is not legal advice.

Pre-deployment testing. Rillor tests the agents an organization plans to deploy on its own tasks, with black-box logging and replayable runs, and scores consistency: the same decision across repeated runs, rephrasings and model versions. Each agent runs on a named model family and a pinned version, and its baselines are rerun when that version changes.

Limits of this analysis

- The mechanisms were shown on open-weights models in open inference engines [1, 3, 21]. One study says it can only speculate about the cause inside hosted APIs [2], and the claim that load explains nearly all endpoint nondeterminism is the September 2025 post's argument [1].
- The figures in Tables 1 and 3 come from different models, metrics, settings and dates, and are not comparable.
- Several sources are preprints. Two are single-author studies [32, 35], one of which notes that its lexical metrics can overestimate drift [32]. The agent study used one temperature [35].
- Provider and engine documentation changes often. The pages cited were viewed on 7 November 2026, and one provider's seed guide is archived [13].
- No source reviewed shows mixture-of-experts routing at work in a named or hosted model. One lists it only as a hypothesis [5, 25, 26].
- Each cost figure comes from one setup, and SGLang's own average overhead (34.35%) differs from the figure LLM-42 measured for it (up to 56%) [21, 33].
- The only statement found on how often a provider changes its serving configuration is one provider's 2023 guide [13].

Sources

1. Defeating Nondeterminism in LLM Inference. Horace He in collaboration with others at Thinking Machines, Thinking Machines Lab, Connectionism, 10 September 2025, viewed 7 November 2026. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/

2. Non-Determinism of "Deterministic" LLM Settings. Berk Atil and others, arXiv:2408.04667, version 5, 2 April 2025 (first version 6 August 2024), viewed 7 November 2026. https://arxiv.org/abs/2408.04667

3. Understanding and Mitigating Numerical Sources of Nondeterminism in LLM Inference. Jiayi Yuan and others, NeurIPS 2025, arXiv:2506.09501, version 2, 24 October 2025, viewed 7 November 2026. https://arxiv.org/abs/2506.09501

4. The Good, The Bad, and The Greedy: Evaluation of LLMs Should Not Ignore Non-Determinism. Yifan Song, Guoyin Wang, Sujian Li and Bill Yuchen Lin, arXiv:2407.10457, 15 July 2024, viewed 7 November 2026. https://arxiv.org/abs/2407.10457

5. An Empirical Study of the Non-determinism of ChatGPT in Code Generation. Shuyin Ouyang, Jie M. Zhang, Mark Harman and Meng Wang, ACM Transactions on Software Engineering and Methodology, arXiv:2308.02828, version 2, 17 October 2024, viewed 7 November 2026. https://arxiv.org/abs/2308.02828

6. How is ChatGPT's behavior changing over time? Lingjiao Chen, Matei Zaharia and James Zou, arXiv:2307.09009, version 3, 31 October 2023, viewed 7 November 2026. https://arxiv.org/abs/2307.09009

7. Quantifying Language Models' Sensitivity to Spurious Features in Prompt Design. Melanie Sclar, Yejin Choi, Yulia Tsvetkov and Alane Suhr, ICLR 2024, arXiv:2310.11324, version 2, 1 July 2024, viewed 7 November 2026. https://arxiv.org/abs/2310.11324

8. Adding Error Bars to Evals: A Statistical Approach to Language Model Evaluations. Evan Miller, arXiv:2411.00640, 1 November 2024, viewed 7 November 2026. https://arxiv.org/abs/2411.00640

9. Lessons from the Trenches on Reproducible Evaluation of Language Models. Stella Biderman, Hailey Schoelkopf and others, arXiv:2405.14782, version 3, 31 May 2026, viewed 7 November 2026. https://arxiv.org/abs/2405.14782

10. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. Shunyu Yao, Noah Shinn, Pedram Razavi and Karthik Narasimhan, arXiv:2406.12045, 17 June 2024, viewed 7 November 2026. https://arxiv.org/abs/2406.12045

11. Model IDs and versioning, Claude Platform documentation. Anthropic, viewed 7 November 2026. https://platform.claude.com/docs/en/about-claude/models/model-ids-and-versions

12. Glossary, Claude Platform documentation. Anthropic, viewed 7 November 2026. https://platform.claude.com/docs/en/about-claude/glossary

13. How to make your completions outputs consistent with the new seed parameter. Shyamal Anadkat, OpenAI Cookbook, 6 November 2023, viewed 7 November 2026. https://developers.openai.com/cookbook/examples/reproducible_outputs_with_the_seed_parameter

14. Create chat completion, API reference. OpenAI, viewed 7 November 2026.

https://developers.openai.com/api/reference/resources/chat/subresources/completions/methods/create

15. Content generation parameters, Gemini Enterprise Agent Platform documentation. Google Cloud, last updated 7 November 2026, viewed 7 November 2026. https://docs.cloud.google.com/gemini-enterprise-agent-platform/models/capabilities/content-generation-parameters

16. Text generation, Gemini API documentation. Google, viewed 7 November 2026. https://aistudio.google.com/docs/text-generation

17. Gemini models, Gemini API documentation. Google, viewed 7 November 2026. https://aistudio.google.com/docs/models

18. cuBLAS documentation, section 2.1.4, Results Reproducibility. NVIDIA, cuBLAS 13.5, viewed 7 November 2026. https://docs.nvidia.com/cuda/cublas/index.html

19. Reproducibility, PyTorch notes. PyTorch, documentation 2.15, viewed 7 November 2026. https://docs.pytorch.org/docs/2.15/notes/randomness.html

20. Floating Point and IEEE 754 Compliance for NVIDIA GPUs. NVIDIA, CUDA 13.5 documentation, viewed 7 November 2026. https://docs.nvidia.com/cuda/floating-point/index.html

21. Towards Deterministic Inference in SGLang and Reproducible RL Training. The SGLang Team, LMSYS Org, 22 September 2025, updated 24 September 2025, viewed 7 November 2026. https://www.lmsys.org/blog/2025-09-22-sglang-deterministic/

22. Batch Invariance, vLLM documentation. vLLM project, 4 November 2026, viewed 7 November 2026. https://docs.vllm.ai/en/latest/features/batch_invariance/

23. Reproducibility, vLLM documentation. vLLM project, 7 October 2026, viewed 7 November 2026. https://docs.vllm.ai/en/latest/usage/reproducibility/

24. A postmortem of three recent issues. Anthropic, 17 September 2025, viewed 7 November 2026. https://www.anthropic.com/research/a-postmortem-of-three-recent-issues

25. From Sparse to Soft Mixtures of Experts. Joan Puigcerver, Carlos Riquelme, Basil Mustafa and Neil Houlsby, ICLR 2024, arXiv:2308.00951, version 2, 27 May 2024, viewed 7 November 2026. https://arxiv.org/abs/2308.00951

26. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. William Fedus, Barret Zoph and Noam Shazeer, Journal of Machine Learning Research, arXiv:2101.03961, version 3, 16 June 2022, viewed 7 November 2026. https://arxiv.org/abs/2101.03961

27. The Curious Case of Neural Text Degeneration. Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes and Yejin Choi, ICLR 2020, arXiv:1904.09751, version 2, 14 February 2020, viewed 7 November 2026. https://arxiv.org/abs/1904.09751

28. Generation strategies, Transformers documentation. Hugging Face, viewed 7 November 2026. https://huggingface.co/docs/transformers/main/en/generation_strategies

29. AI Act, Article 15: Accuracy, robustness and cybersecurity. European Commission, AI Act Service Desk, Regulation (EU) 2024/1689, viewed 7 November 2026. https://ai-act-service-desk.ec.europa.eu/en/ai-act/article-15

30. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile, NIST AI 600-1. National Institute of Standards and Technology, July 2024, viewed 7 November 2026. https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf

31. Demystifying evals for AI agents. Anthropic, 9 January 2026, viewed 7 November 2026. https://www.anthropic.com/research/demystifying-evals-for-ai-agents

32. Quantifying non deterministic drift in large language models. Claire Nicholson, arXiv:2601.19934, 12 January 2026, viewed 7 November 2026. https://arxiv.org/abs/2601.19934

33. LLM-42: Enabling Determinism in LLM Inference with Verified Speculation. Raja Gond, Aditya K Kamath, Ramachandran Ramjee and Ashish Panwar, arXiv:2601.17768, version 2, 30 January 2026, viewed 7 November 2026. https://arxiv.org/abs/2601.17768

34. Deterministic Inference across Tensor Parallel Sizes That Eliminates Training-Inference Mismatch. Ziyang Zhang, Xinheng Ding, Jiayi Yuan and others, arXiv:2511.17826, version 2, 29 May 2026 (first version 21 November 2025), viewed 7 November 2026. https://arxiv.org/abs/2511.17826

35. How Consistent Are LLM Agents? Measuring Behavioral Reproducibility in Multi-Step Tool-Calling Pipelines. Abel Yagubyan, arXiv:2605.28840, 23 April 2026, viewed 7 November 2026. https://arxiv.org/abs/2605.28840

About this report

Published by Rillor on 8 November 2026. Superintelligence (SI) tools assist research and drafting, and every figure cites its source. Rillor builds and secures SI systems, supplies the data they learn from and delivers the infrastructure they run on. We apply SI to financial and compute markets.

Cite as: Rillor. LLM nondeterminism: why the same input gets different answers. 8 November 2026. https://rillor.com/insights/llm-nondeterminism

Notices

NVIDIA, AMD, Ceph, Kubernetes and related product names are trademarks of their owners and are used only to identify products. No affiliation or endorsement is implied.

Superintelligence (SI) outputs can be wrong. Important decisions should include human review.

Work with Rillor

Send us the tasks your agent handles, the model versions it runs on and the decisions it is trusted with. Contact us at rillor.com/contact?type=evaluation or contact@rillor.com. More on SI evaluation: rillor.com/si/evaluation