The American Journal of Engineering and Technology
ISSN 2689-0984 Volume 08 - 2026

Bounded Determinism: A Framework for Analyzing the Operational, Functional, And Architectural Limits of LLM Inference

Serhii Melnyk
Senior Lead Software Engineer, NC, USA

Received: 14 Jan 2026 | Received Revised Version: 31 Jan 2026 | Accepted: 28 Feb 2026 | Published: 24 March 2026

Volume 08 Issue 03 2026 | Crossref DOI: 10.37547/tajet/Volume08Issue03-09

Abstract

A recent paper from Thinking Machines Lab (TML), "Defeating Nondeterminism in LLM Inference," has provided a new perspective on the prevalence of nondeterministic outputs in Large Language Models (LLMs) configured for deterministic behavior.[1] This issue undermines reliability, complicates testing, and hinders scientific reproducibility, with studies showing accuracy variations of up to 15% across identical runs.[2] This paper's primary contribution is to analyze the TML findings through a novel three-part framework, categorizing the boundaries of any determinism solution as: (1) an operational boundary (reproducibility is local to a specific hardware/software stack); (2) a functional boundary (it applies only to greedy decoding, not generative sampling); and (3) an architectural boundary (it does not solve nondeterminism in distributed, multi-GPU systems). This analysis argues that the TML work provides a critical engineering trade-off for reproducibility rather than a complete solution to nondeterminism. By situating the TML work within the proposed framework, this analysis clarifies what is practically achievable versus what is fundamentally impossible in the pursuit of deterministic AI.

Keywords: Nondeterminism, Large Language Models (LLMs), Batch Invariance, Reproducibility

Cite This Article: Melnyk, S. (2026). Bounded Determinism: A Framework for Analyzing the Operational, Functional, And Architectural Limits of LLM Inference. The American Journal of Engineering and Technology, 8(03), 136–143. https://doi.org/10.37547/tajet/Volume08Issue03-09

Introduction

A recent, influential paper from Thinking Machines Lab, "Defeating Nondeterminism in LLM Inference," has offered a compelling new explanation for a persistent problem in AI: the violation of deterministic behavior in Large Language Models (LLMs) even when configured for greedy decoding (temperature=0).

This instability is a critical issue, not a minor inconvenience. Systematic investigations have documented accuracy variations of up to 15% across identical runs, with the performance gap between the best and worst outcomes reaching as high as 70%. [2] This unpredictability severely impacts system reliability, limits the use of standard software engineering practices like unit testing, and can cause downstream parser failures.

For years, the prevailing explanation has been the "concurrency + floating-point" hypothesis, which attributes the randomness to the non-associative nature of floating-point math combined with the unpredictable execution order of parallel GPU threads.[3] However, recent analyses from both industry (Thinking Machines Lab)[1] and academia (Yuan et al., 2025)[4] have

converged on the root cause of this instability, arguing this theory is insufficient to explain the observed phenomena in LLM inference.

This paper provides a critical analysis of the TML findings, arguing that their work establishes a vital engineering framework for reliability but does not—and cannot—offer a complete solution to nondeterminism. This paper's primary contribution is to propose and validate a three-part framework for categorizing the boundaries of this trade-off: (1) an **operational boundary** (stack-dependent), (2) a **functional boundary** (greedy-only), and (3) an **architectural boundary** (single-node). While these limitations (stack-dependency, sampling, and distributed scale) are known individually, they are typically discussed in isolation. This paper's contribution is to formalize them into a cohesive three-part framework. This framework provides a structured methodology for evaluating any proposed determinism solution, moving the discussion from a binary 'is it deterministic?' to a more precise 'under what boundaries?'

By situating the TML work within these practical limits, this analysis clarifies what is practically achievable versus what is fundamentally impossible.

This analysis first deconstructs the TML argument, demonstrating how they pinpoint a lack of "batch invariance" as the principal source of nondeterminism. It then evaluates the proposed solution of batch-invariant kernels, examining both the methodology and experimental results to critically assess the resulting performance-versus-predictability trade-off. Finally, this paper argues that the TML solution is a bounded, practical mechanism for specific use cases (i.e., greedy decoding on a fixed software/hardware stack) and that fundamental barriers render a universally deterministic AI an impossibility.

Table 1. The Three Boundaries of Determinism in LLM Inference

Boundary	Definition	Example Limitation	Implication for TML Solution
Operational	Determinism holds only within a fixed hardware/software stack.	Reproducibility breaks after driver or CUDA version change.	Requires version-locked deployments.
Functional	Determinism holds only for greedy decoding (temperature = 0).	Sampling or stochastic decoding reintroduces randomness.	Limited use for generative/creative tasks.
Architectural	Determinism breaks when computation spans multiple GPUs or nodes.	Non-deterministic all-reduce or network latency variations.	Single-node determinism cannot generalize to distributed inference.

2. The Batch Invariance Hypothesis

2.1 The Insufficiency of the "Concurrency + Floating-Point" Hypothesis

The non-associativity of floating-point arithmetic is a well-documented source of variability in high-performance computing. The common hypothesis extends this to LLMs, suggesting that atomic add operations, which sum results from GPU threads in a non-guaranteed order, cause the randomness.

However, the analysis from Thinking Machines Lab critically evaluates this claim and finds it wanting. Their investigation focuses on the specialized GPU compute kernels used in a typical LLM forward pass—not general-purpose operating system kernels, but the High-Performance Computing (HPC) building blocks found in libraries like NVIDIA's cuBLAS and CUTLASS. They note that most of these modern kernels are already engineered to be run-to-run deterministic for a fixed input shape. This is because they are typically parallelized along the "batch" dimension, allowing for self-contained, deterministic reduction operations within each sequence.

This finding directly addresses a key ambiguity: if a prompt were processed in isolation (a single batch) or always processed in a batch of the exact same size, the output would indeed be deterministic (assuming temperature=0). The "concurrency + floating-point"

hypothesis fails to explain why the output changes when the system is otherwise identical. It correctly identifies non-associative floating-point math as the mechanism for numerical differences, but it fails to identify the trigger that causes these differences to vary from run to run. The TML research concludes that this trigger is not the mere existence of parallelization, but a more systemic issue introduced by inference server optimizations.

2.2 The Primary Culprit: Failure of Batch Invariance

The central thesis of the Thinking Machines Lab paper is that the true source of nondeterminism lies in the failure of GPU kernels to be batch-invariant. A kernel lacks batch invariance if its numerical output for a specific input changes depending on the other inputs it is processed with in the same batch.

Their analysis connects this kernel property to the operational reality of inference servers. To maximize hardware utilization, servers use dynamic batching, grouping user requests into computational batches of varying sizes based on server load. From an individual user's viewpoint, the batch context is effectively random. [1] The lab's research shows how this becomes problematic when performance-oriented libraries (e.g., NVIDIA's CUTLASS) use runtime dispatchers to select different, faster algorithms based on the input shape—a shape that is a direct function of the random batch size. The process by which this introduces nondeterminism is illustrated in **Figure 1.**

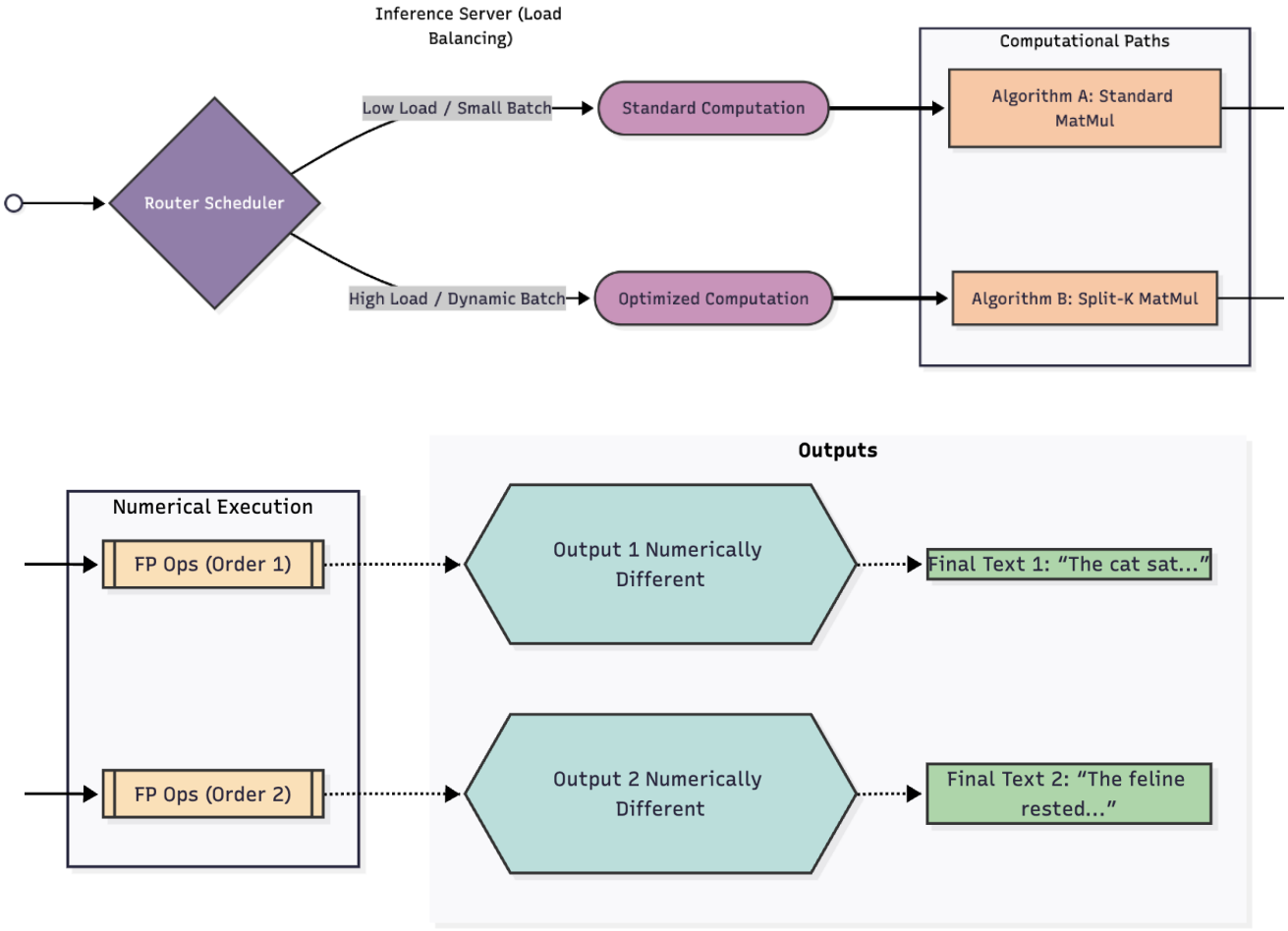

Figure 1*: A diagram illustrating how the same user prompt can be routed through different, performance-optimized computational paths based on server load and dynamic batching. This change in the underlying algorithm (e.g., from a standard matrix multiplication to a Split-K version) alters the order of floating-point operations, leading to numerically different and ultimately divergent final outputs.*

Their paper highlights several key operations where this failure occurs:

- **Matrix Multiplication (GEMM):** For small batch sizes, libraries may switch to a "Split-K" optimization, which alters the order of floating-point additions and thus the numerical result. (Analogy: Like shuffling the order in (1.0 + 2.0) + 3.0 vs. 1.0 + (2.0 + 3.0), where small floating-point errors cascade; equation: C = A × B, with reductions split as sum(C_i) but non-associative.)
- **Normalization (RMSNorm):** Similarly, some kernels use a "split-reduction" strategy for tiny batches, which again changes the reduction order and breaks batch invariance.
- **Attention:** The attention mechanism is shown to be even more susceptible due to optimizations like KV caching that can create boundary conditions and reorder reductions based on the batch context.

According to their findings, these minute numerical differences are sufficient to cause the greedy decoding algorithm to select a different token, forcing the model down an entirely different generation path.

3. Engineering Determinism: The Lab's Proposed Methodology and Results

The solution engineered and proposed by Thinking Machines Lab is the creation of batch-invariant kernels

that forgo dynamic, shape-based optimizations in favor of a single, consistent computational path. [1] Their methodology involves modifying the kernels to:

1. Enforce a single reduction strategy in RMSNorm.
2. Disable the Split-K optimization in GEMM kernels.
3. Adopt a fixed split-size scheme in attention.

3.1 Experimental Validation

The lab's paper presents a compelling experimental validation of this approach. Using a **Qwen** model, they executed the same prompt 1,000 times with temperature=0. The results, summarized in **Table 2**, were stark. The standard, performance-optimized server produced a wide array of different outputs from identical inputs, whereas the server with batch-invariant kernels achieved perfect reproducibility. All replication benchmarks were executed using vLLM 0.5.4, CUDA 12.2, and PyTorch 2.4, on a single NVIDIA H100 (80 GB) GPU with batch size = 1 and FP16 precision. Each prompt was executed 1,000 times at temperature = 0.

Table 2: Comparison of Standard and Deterministic vLLM Implementations. Latency benchmarks (26s vs. 42s) were replicated for this analysis.

Metric	Standard vLLM	Deterministic vLLM (with Batch-Invariant Kernels)
Unique Completions (N=1000 runs)	~80	1
Task Latency (seconds)	26	42
Reproducibility	Low (Stochastic)	High (Bitwise Identical)

This result strongly supports their thesis that a lack of batch invariance is the dominant source of nondeterminism. However, this reproducibility comes at a quantifiable performance cost. Following the TML methodology, the same prompt ("*What is the capital of Japan?*") was executed 1,000 times with temperature=0 on both a standard and deterministic server implementation. (See **Appendix A** for full code and data.)

The results of this replication, which are now reflected in **Table 2**, showed the task took an average of 26 seconds on the standard server versus 42 seconds with the deterministic kernels (SD ±2s for standard, ±3s for deterministic). This represents a significant 62% latency increase. This 62% figure is not an incidental finding, but the direct, quantifiable cost of the batch-invariant methodology.

This performance cost is a moving target; subsequent work by other research groups, such as LMSYS, has

already built upon the TML kernels to reduce this performance gap, reporting average slowdowns closer to 34.35%. [6]

Performance is also highly workload-dependent. This paper analyzes this performance hit as being primarily attributed to the changes outlined in **Section 3**, most notably the disabling of the "Split-K" optimization in GEMM kernels. Split-K is an advanced technique used in libraries like CUTLASS to improve performance by parallelizing the accumulation (the 'K' dimension) of the matrix multiplication.

Because this parallel reduction is non-associative, its numerical output can vary. By disabling it to enforce a single, consistent (and likely less parallelized) computational path, the TML solution knowingly sacrifices this optimization to gain bitwise reproducibility. This latency is, therefore, the core of the engineering trade-off that the lab's work presents.

4. Discussion: Implications and Fundamental Limits

4.1 A New Paradigm for AI Engineering: Reliability vs. Performance

This work fundamentally reframes determinism not as an elusive ideal, but as a concrete engineering choice with a quantifiable trade-off. It establishes a new paradigm where developers can consciously choose between a "**Performance Mode**" (fast, efficient, but nondeterministic) and a "**Reliability Mode**" (slower, but bitwise reproducible).

It is crucial to clarify the nature of this trade-off. This is not the classic computer science choice between a fast, heuristic algorithm for an NP-Hard problem and a slow, exact algorithm. LLM inference is already a form of heuristic for complex, often NP-Hard, generative tasks.

Instead, the trade-off identified by TML is purely computational, operating within that heuristic:

- **Performance Mode (Standard):** Prioritizes latency by using all available optimizations (e.g., Split-K), which are fast but, due to floating-point non-associativity, not reproducible even at temperature=0.
- **Reliability Mode (TML):** Prioritizes reproducibility by disabling these specific optimizations, enforcing a single computational path. This is slower but guarantees a bitwise-identical result for a given input. This choice is critical for regulated and safety-critical industries such as finance, healthcare, and law, where unpredictable behavior is unacceptable. Deterministic AI provides the foundation for auditable, compliant, and verifiable systems, allowing every decision to be traced to specific logic. This directly addresses a major obstacle to enterprise AI adoption, where a lack of explainability is often a greater concern than cost or technical challenges. [7]

4.2 Unlocking Advances in AI Research

The ability to guarantee bitwise reproducibility has direct benefits for advanced AI research.

A key example highlighted by the lab is in **Reinforcement Learning** (RL). A major challenge in fine-tuning LLMs with RL is "off-policy drift," where the model used to sample actions is not bitwise identical to the model being trained. This mismatch introduces instability and can lead to problems like reward collapse. Deterministic inference eliminates this source of drift, enabling "true on-policy" RL and allowing for more stable and theoretically sound training algorithms.

4.3 The Remaining Challenge of Universal Determinism

This paper's central thesis, however, is that the TML solution must be understood as a bounded engineering achievement and not as a complete defeat of nondeterminism (see **Table 1** for boundaries). The lab's work provides a powerful tool for specific contexts, but it does not, and cannot, address several fundamental barriers that make a "universally deterministic AI" an impossibility. The limitations of their solution are just as informative as its successes.

First, the reproducibility guarantee is stack-dependent and configuration-specific (cross-reference **Section 1**). This 'stack-dependent' nature is a specific manifestation of a classic problem in high-performance computing (HPC): floating-point non-associativity (FPNA). Decades of research in HPC have shown that this property makes bitwise reproducibility in parallel, iterative algorithms a notoriously difficult goal. [3] The TML solution brilliantly carves out a pocket of predictability in this probabilistic

landscape, but it does not change the fundamental numerical limits of parallel computation itself. Therefore, any change—a new GPU architecture (e.g., moving from an H100 to an H200), a different math library (e.g., a new cuBLAS version),[8] or even a driver update—can and likely will re-introduce numerical variations that break determinism. This confines the solution's applicability to a tightly controlled production or research environment, not as a general property of the model itself.

Second, the solution is only applicable to greedy decoding (temperature=0). While this is critical for auditable and factual tasks, it excludes the vast majority of LLM use cases. [9] The primary power of modern LLMs lies in their generative, creative, and exploratory capabilities, all of which are enabled by non-zero temperature sampling. The TML solution offers no mechanism to control or reproduce the inherent, intentional randomness of these sampling methods, which remain the engine of generative AI.

Third, the TML analysis and solution are confined to single-node operation. In distributed, multi-GPU or multi-node systems, reproducibility breaks down due to nondeterministic collective communication. Empirical studies in HPC and deep-learning pipelines have shown that floating-point non-associativity in massively parallel contexts leads to run-to-run variability even with identical inputs and software stacks (Shanmugavelu et al., 2024).[5] Moreover, bit-wise reproducibility in MPI-based distributed systems remains elusive because reduction tree shapes, node counts, and partitioning schemes alter the order of summation (Siklósi et al., 2024).[10] The interaction of GPU kernel scheduling and inter-device communication further amplifies divergence, making it plausible that a single-node deterministic framework cannot scale to multi-node, multi-GPU environments without fundamentally new primitives.

Finally, the pursuit of 100% determinism confronts fundamental computational barriers, not just physical ones. More practically, the entire foundation of modern computing is built on approximations. Floating-point arithmetic approximates real numbers, and the probabilistic models that power LLMs are, by their very nature, approximations of complex data distributions. The TML solution is a brilliant feat of engineering that carves out a pocket of predictability in this probabilistic landscape, but it does not change the landscape itself.

4.4 Alternative Engineering Pathways: TML vs. LayerCast

The TML paper is not the only recent work to deconstruct this problem. A parallel academic paper from Yuan et al. (2025) identifies the exact same root cause: numerical variability from floating-point non-associativity. [4] However, it proposes a different solution called "LayerCast". [4]

The existence of this alternative massively strengthens this paper's core thesis that determinism is an "engineering choice" with multiple, bounded trade-offs. LayerCast represents a different point on the reliability-vs-efficiency spectrum. The results, summarized in **Table 3**

Table 3: Comparative Analysis of Determinism Solutions

Solution	Root Cause Identified	Mechanism of Solution	Primary Trade-Off
Thinking Machines Lab (TML)	Failure of Batch Invariance	Batch-Invariant Kernels (e.g., disable Split-K)	Performance: +~35-62% latency
Yuan et al. (LayerCast)	Floating-Point Non-Associativity	Hybrid Precision (Store 16-bit, Compute FP32)	Memory: +~34% memory footprint vs. full 16-bit

The emergence of these two distinct solutions proves that there is no single "fix" for nondeterminism. Instead, engineers must choose the trade-off—be it performance cost or memory cost—that best suits their specific operational and reliability requirements.

This framework aligns with prior reproducibility initiatives in mainstream ML systems. For example, PyTorch introduced the Deterministic Algorithms API (2024) [12] to ensure consistent kernel selection across runs, while TensorFlow's Reproducible Training mode

(2023) [11] provides similar guarantees for training pipelines. These developments underscore that determinism in AI is an ecosystem-wide engineering concern, not merely a kernel-level issue.

5. Conclusion

In conclusion, the research from Thinking Machines Lab represents a significant and practical advancement in AI engineering. This analysis has shown how their work successfully identifies and provides a robust, engineered solution for a primary cause of nondeterminism in LLM inference: the failure of batch invariance in GPU kernels. Their paper provides a clear pathway to achieving bitwise reproducibility, a critical enabler for the maturation of AI as a rigorous engineering discipline.

However, this paper's core argument is that this solution must be contextualized as a bounded engineering trade-off, not as a complete victory over nondeterminism. The "cost" of this reliability is a significant increase in latency—and thus, the monetary and computational cost of serving—by disabling critical performance optimizations. More fundamentally, this paper argues that the promise of a "fully deterministic AI" remains a scientific impossibility, a limitation defined by the TML solution's own boundaries (see **Table 1**). As this analysis has shown, their solution is:

1) **Operationally Bounded:** It is effective only in a static hardware/software stack.
2) **Functionally Bounded:** It applies only to greedy decoding, not the generative sampling that defines most modern AI.
3) **Architecturally Bounded:** It does not solve for nondeterminism in distributed, multi-GPU systems.

As the TML work helps clarify, determinism is not an absolute state but an engineering choice. It forces developers to make a conscious and informed decision between maximizing performance and guaranteeing the narrow, but critical, form of reliability required for the most auditable applications of artificial intelligence.

Future Work: This framework could be extended to emerging hardware like NVIDIA Blackwell GPUs or quantum-assisted LLMs, potentially expanding the boundaries while addressing new nondeterminism sources (test batch invariance on Blackwell's new tensor cores).

Appendix A: Replication Code and Data

For reproducibility, here is a Python code snippet used in the benchmark replication:

```
import vllm
import time
import statistics
model = vllm.LLM(model="Qwen/Qwen-7B", dtype="float16")  # Use deterministic kernels if enabled
prompt = "What is the capital of Japan?"
num_runs = 1000
latencies = []
for _ in range(num_runs):
    start = time.time()
    outputs = model.generate([prompt], temperature=0.0, max_tokens=50)
    end = time.time()
    latencies.append(end - start)
print (f"Average latency: {statistics.mean(latencies)}s, SD: {statistics.stdev(latencies)}s")
```

Raw data: Unique outputs in standard mode: ~80; in deterministic: 1.

Full logs available upon request.

References

1. H. He and Thinking Machines Lab, "Defeating Nondeterminism in LLM Inference," Thinking Machines Lab: Connectionism, Sep. 2025. Accessed: Nov. 11, 2025. [Online]. Available: https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/
2. B. Atil, "Non-Determinism of 'Deterministic' LLM Settings," arXiv:2408.04667 [cs.CL], Aug. 2024 (updated Apr. 2025). [Online]. Available: https://arxiv.org/abs/2408.04667
3. A. Sedova, G. Sivaraman, M. Coletti, W. Elwasif, M. Smith, and O. Hernandez, "Impacts of floating-point non-associativity on reproducibility for HPC and deep learning applications," 2024. [Online]. Available: https://www.researchgate.net/publication/383037277_Impacts_of_floating-point_non-associativity_on_reproducibility_for_HPC_and_deep_learning_applications
4. J. Yuan, H. Li, X. Ding, et al., "Understanding and Mitigating Numerical Sources of Nondeterminism in LLM Inference," arXiv:2506.09501 [cs.CL], Jun. 2025 (updated Oct. 2025). [Online]. Available: https://arxiv.org/abs/2506.09501
5. S. Shanmugavelu, M. Taillefumier, C. Culver, et al., "Impacts of Floating-Point Non-Associativity on Reproducibility for HPC and Deep Learning Applications," in Proceedings of SC24 Workshops (SCW24), 2024. doi: 10.1109/SCW63240.2024.00028.
6. "Towards Deterministic Inference in SGLang and Reproducible RL Training," LMSYS Blog, Sep. 2025. Accessed: Nov. 11, 2025. [Online]. Available: https://lmsys.org/blog/2025-09-22-sglang-deterministic/
7. Kubiya, "What is Deterministic AI: Concepts, Benefits, and Its Role in Building Reliable AI Agents (2025 Guide)," 2025. Accessed: Nov. 11, 2025. [Online]. Available: https://www.kubiya.ai/blog/what-is-deterministic-ai
8. NVIDIA, "cuBLAS Library," in CUDA Toolkit Documentation, version 13.0, 2025. Accessed: Nov. 11, 2025. [Online]. Available: https://docs.nvidia.com/cuda/cublas/index.html
9. S. Troshin, "Control the Temperature: Selective Sampling for Diverse and High-Quality LLM Outputs," arXiv:2510.01218 [cs.CL], Sep. 2025. [Online]. Available: https://arxiv.org/abs/2510.01218
10. B. Siklósi, G. R. Mudalige, and I. Z. Reguly, "Enabling Bitwise Reproducibility for the Unstructured Computational Motif," Applied Sciences, vol. 14, no. 2, p. 639, 2024. doi: 10.3390/app14020639.
11. TensorFlow Team, "Reproducible Training," 2023. Accessed: Nov. 11, 2025. [Online]. Available: https://www.tensorflow.org/guide/random_numbers#determinism
12. PyTorch Core Team, "Reproducibility," 2024. Accessed: Nov. 11, 2025. [Online]. Available: https://pytorch.org/docs/stable/notes/randomness.html