Non-Deterministic Output in Ollama

Why temperature=0 Doesn’t Guarantee Reproducibility

And What You Can Do About It

Technical Report — February 2026

1. Executive Summary

A common expectation when running large language models (LLMs) through Ollama is that setting **temperature=0** will yield perfectly deterministic, repeatable output. In practice, users frequently observe that different Ollama hosting instances—or even successive runs on the same instance—produce subtly different results despite identical prompts and parameters.

This report explains the root causes of this non-determinism, traces them through the software and hardware stack, and provides actionable recommendations for maximising reproducibility in Ollama-based deployments.

2. Background: How Ollama Generates Text

Ollama is a user-friendly wrapper around **llama.cpp**, the open-source C++ inference engine for transformer-based LLMs. When you issue a prompt, Ollama forwards it to llama.cpp, which performs the following pipeline:

- **Tokenisation:** The input text is converted into a sequence of integer token IDs.
- **Prompt evaluation (prefill):** All input tokens are processed in a batched forward pass through the model's transformer layers, populating the key-value (KV) cache.
- **Autoregressive generation:** One token at a time, the model computes a logit vector over the full vocabulary, a sampling strategy selects the next token, and the result is appended.
- **Sampling:** With temperature=0 (greedy decoding), the token with the highest logit is always selected. This eliminates randomness from the sampling step itself.

The critical insight is that **temperature only controls the sampling stage**. If the logits themselves differ between runs, greedy decoding will faithfully amplify that difference by selecting a different argmax token. Everything upstream of sampling—the matrix multiplications, normalisations, and attention computations—is where non-determinism originates.

3. Root Causes of Non-Determinism

3.1 Floating-Point Non-Associativity

The single most fundamental cause is that **floating-point arithmetic is not associative**. In IEEE 754 representation, (a + b) + c does not necessarily equal a + (b + c) due to rounding at each intermediate step. Since LLM inference consists of billions of multiply-accumulate operations, any change in the order those operations are performed can cause logits to shift by small amounts—typically around 1e-4 to 1e-6.

When two logits for competing tokens are very close in value (which happens regularly in natural language), even a tiny perturbation can flip which token is the argmax. Because generation is autoregressive, a single flipped token early on cascades into a completely different continuation.

3.2 GPU Kernel Non-Determinism

The General Matrix Multiply (GEMM) kernels provided by NVIDIA's cuBLAS library are heavily optimised for throughput. These optimisations include the use of Tensor Cores, dynamic algorithm selection, and internal parallelism that does not guarantee a fixed reduction order. This means the same matrix multiplication on the same GPU can yield slightly different results depending on thread scheduling and workload conditions.

Different GPU architectures (e.g., Ampere vs. Ada Lovelace vs. Hopper) may execute the same CUDA code with different Tensor Core generations and different PTX instruction sequences, further amplifying cross-machine divergence. Research from Ingonyama demonstrated that standard cuBLAS GEMM kernels produced results differing by approximately 1e-4 across an RTX 3090, RTX 4080, and NVIDIA L4.

3.3 Batch Size and Continuous Batching

When llama.cpp's server runs with multiple slots (concurrent request processing), continuous batching combines tokens from different requests into a single GPU batch. Different batch sizes change the dimensions of matrix operations and therefore change the floating-point reduction order. A user on the llama.cpp project demonstrated that sending the same prompt to 8 parallel slots on an H100 produced between 5 and 8 unique completion texts, even with temperature=0 and all other samplers disabled.

Even without concurrent requests, the batch size used during the prompt-evaluation (prefill) phase differs from the single-token batch used during generation. This means the first few generated tokens after prefill may already diverge from what a fully sequential implementation would produce.

3.4 Context Window Size (num_ctx)

Ollama dynamically adjusts the context window size if the num_ctx parameter is not explicitly set. Different context sizes change the dimensions of the attention computation and the KV cache layout, which can alter the numerical path through the model. Multiple community reports have confirmed that explicitly fixing num_ctx is essential for achieving any level of reproducibility. Without it, output is described as "slightly random" even with a fixed seed and temperature=0.

3.5 Quantisation Artefacts

Most models run through Ollama are quantised to 4-bit or 8-bit GGUF format to fit in consumer GPU VRAM. While the weights themselves are stored as integers, they are **dequantised to floating-point at runtime** for the actual multiply-accumulate operations. Additionally, normalisation layers (RMSNorm, LayerNorm) typically remain in full floating-point precision even in "fully quantised" models. This means the core floating-point non-associativity problem is not eliminated by quantisation.

Furthermore, different quantisation levels (e.g., Q4_K_M vs. Q8_0) will produce different logit distributions entirely, so two instances running different quants of the same base model will naturally produce different outputs.

3.6 CPU vs. GPU and Mixed Offloading

If a model does not fully fit in GPU VRAM, Ollama will partially offload layers to CPU. The number of layers offloaded depends on available VRAM, which varies between machines. CPU and GPU execute matrix operations with different instruction sets (e.g., AVX-512 vs. CUDA), different precision modes, and different reduction orders, producing different numerical results for the same logical computation.

3.7 Software Version Differences

Different versions of Ollama ship with different versions of the underlying llama.cpp engine, which may include changes to kernel implementations, memory layouts, default parameters, and tokeniser behaviour. Even minor updates can change the computational path enough to shift logits and produce different outputs.

3.8 First-Run vs. Subsequent-Run Divergence

A particularly confusing pattern reported by users is that the first invocation after loading a model produces output that differs from all subsequent invocations (which are themselves consistent). This is likely caused by differences in memory allocation and cache state during the initial model load: the first prefill may use a different code path or memory layout compared to subsequent calls where the model is already resident in GPU memory and caches are warm.

4. Summary of Non-Determinism Sources

Source	Mechanism	Cross-Machine Impact
FP Non-Associativity	Reduction order changes logits by ~1e-4	High — differs across any hardware
GPU Kernels (cuBLAS)	Tensor Cores, dynamic algorithm selection	High — architecture-dependent
Batch Size / Batching	Different batch dims change FP path	Medium — depends on server config
Context Window (num_ctx)	Dynamic sizing changes attention dims	Medium — preventable
Quantisation	Dequant at runtime; norm layers in FP	High — quant type must match exactly
CPU/GPU Offload Split	Different instruction sets & precision	High — VRAM-dependent

Software Versions	Kernel and layout changes between releases	Medium — pin versions to prevent
First-Run Effect	Cold cache / memory allocation differences	Low — same-machine only

5. Mitigation Strategies

5.1 Maximising Same-Machine Reproducibility

To achieve reproducible outputs across repeated runs on a single Ollama instance, apply all of the following settings in your API request:

- Set **temperature** to **0** (greedy decoding) to eliminate sampling randomness.
- Set a fixed **seed** value (e.g., seed: 123). While Ollama's documentation states this should produce the same text for the same prompt, in practice it is necessary but not sufficient.
- Explicitly set **num_ctx** to a fixed value (e.g., 2048 or 4096). This prevents Ollama from dynamically resizing the context window.
- Avoid concurrent requests. Run one request at a time (single-slot mode) to prevent continuous batching from altering the computational path.
- Discard the first response after model load. Issue a throwaway prompt first, then use subsequent responses, which tend to be consistent with each other.

With all of these measures in place, same-machine reproducibility is generally achievable for the same Ollama + llama.cpp version on the same hardware.

5.2 Maximising Cross-Machine Reproducibility

Achieving identical output across different machines is substantially harder and, in the general case, not fully possible with standard Ollama builds. The following measures help narrow the gap:

- Use identical hardware (same GPU model, same CPU architecture) across all instances.
- Pin the exact same Ollama version, and therefore the same llama.cpp commit, across all machines.
- Use the exact same GGUF model file (same quantisation type, same file hash) everywhere.
- Ensure full GPU offloading on all machines (set num_gpu to a sufficiently large number) so no layers fall back to CPU. This requires all machines to have enough VRAM.
- Match operating system, CUDA toolkit version, and GPU driver version across machines.

Even with all of the above, small floating-point divergences from cuBLAS kernel scheduling may still produce occasional token-level differences. The Ingonyama research team found that true cross-architecture determinism required rewriting the GEMM CUDA kernels to avoid Tensor Cores and enforce a deterministic reduction order—a modification far beyond standard Ollama usage.

5.3 Alternative Approaches for Strict Determinism

If your use case absolutely requires bit-exact reproducibility (e.g., for auditing, benchmarking, or ZK proofs), consider the following approaches:

- **CPU-only inference:** Running the model entirely on CPU with deterministic BLAS libraries and fixed thread counts eliminates GPU kernel non-determinism. This comes at a significant performance cost.
- **Custom deterministic kernels:** The Ingonyama project demonstrated that rewriting llama.cpp's GEMM kernels to use only CUDA cores (no Tensor Cores) and enforce a fixed reduction order can achieve cross-GPU-architecture determinism, with modest performance penalties during prefill but negligible impact during generation.
- **Containerised inference:** Package the full stack (OS, drivers, Ollama, model file) in a container image and run on identical hardware to maximise environmental consistency.
- **Semantic equivalence testing:** Rather than requiring bit-exact output, design your application to tolerate paraphrased-but-equivalent responses. Use embedding similarity or structured output validation to verify functional equivalence.

6. Practical API Configuration

The following JSON snippet shows the recommended Ollama API options for maximum reproducibility:

```
{
"model": "llama3:8b-q8_0",
"messages": [{"role": "user", "content": "Your prompt"}],
"options": {
"seed": 42,
"temperature": 0,
"num_ctx": 4096,
"num_gpu": 999,
"num_thread": 8
}
}
```

Key points: **temperature: 0** enables greedy decoding. **seed** fixes the random number generator. **num_ctx** prevents dynamic context resizing. **num_gpu: 999** forces all layers to GPU (assuming sufficient VRAM). **num_thread** fixes the CPU thread count for any CPU-side computation.

7. Conclusion

Non-determinism in Ollama at temperature=0 is not a bug—it is an inherent consequence of how modern GPU hardware executes floating-point arithmetic. The temperature parameter only controls the sampling stage; it cannot compensate for numerical variations in the millions of matrix operations that produce the logits being sampled from.

For same-machine reproducibility, the combination of a fixed seed, fixed num_ctx, temperature=0, and single-slot operation is usually sufficient. For cross-machine reproducibility, hardware and software environments must be matched as closely as possible, with the understanding that cuBLAS kernel-level non-determinism may still introduce occasional token differences.

True bit-exact reproducibility across different GPU architectures requires modifications to the inference kernels themselves—a level of intervention that is feasible but imposes performance trade-offs and is not supported out of the box by Ollama or llama.cpp.

8. References

• Ollama GitHub Issue #586: "/api/generate with fixed seed and temperature=0 doesn’t produce deterministic results"

• Ollama GitHub Issue #5321: "Llama3: Generated outputs inconsistent despite seed and temperature"

• llama.cpp GitHub Issue #7052: "Non-deterministic output of the llama.cpp server when using multiple slots"

• Ingonyama: "Solving Reproducibility Challenges in Deep Learning and LLMs" (September 2024)

• LLMs-from-scratch Issue #249: "Ch07 – Ollama reproducibility" (June 2024)

• NVIDIA: "Determinism in Deep Learning" (GTC 2019, Session S9911)

• PyTorch Documentation: "Reproducibility" (pytorch.org/docs/stable/notes/randomness.html)

• Keywords AI: "How to Get Consistent and Reproducible LLM Outputs in 2025"