

Peer Review

Review of: "MiND: A Minimal Deterministic Middleware for Auditable LLM Interaction"

Evgenii Arsentev¹

1. Independent researcher

1. What the work claims

The paper introduces **MiND**, a thin HTTP middleware that sits between a client and an LLM provider and records what happened in each call. The central move is a redefinition. The author states in §1 and again in §2 that determinism here applies to the request-handling procedure, not to the model output: *"MiND does not claim to infer model internals, eliminate uncertainty, or guarantee semantic reproducibility of responses"* (§1). What the middleware promises instead is an explicit pipeline (§3.2 lists eight steps, from "receive user input through the chat endpoint" to "return the generated output to the client"), an externalized configuration object, and a per-call record the paper calls a **cycle artifact**: four fields (input, config, output, meta) plus a `cycle_id` (§3.4). The reference implementation is FastAPI-based and exposes four endpoints (§3.3): `/health`, `/ame/config`, `/chat`, `/ame/last`. §1.1 lists four contributions, the third of which is that the paper *"demonstrates that auditable interaction can be operationalized through recoverable cycle artifacts"*. §5 reports a single local execution in which each of the four endpoints was called and responded as designed. §8 gives a GitHub repository and a PyPI package.

Two things deserve credit before I start objecting. First, the author is unusually disciplined about scope. The limitation statements in §3.5, §4 (final two paragraphs), §5.5, and §6 are specific, not ritual: no indexed history, no multi-agent coordination, no provider-agnostic abstraction, no comparative experiments, no user study, and an admission in §6 that the default system template *"may reflect project-specific assumptions or domain language"*. Papers that oversell are the norm; this one does not. Second, the artifact is genuinely public, with both a source repository and an installable package, which is more than many software papers in this venue manage. My difficulty is with what remains after the honest

subtractions. When a paper concedes that it has no evaluation, no comparison, no history, and no semantic guarantees, the question becomes whether the residue is a contribution or a description of a small program. On the current text, I think it is closer to the latter, and the fixes needed are not cosmetic.

2. *Whether the conclusion holds on the data*

2.1 There is no data. I want to state this plainly rather than dress it up. §5 is titled "Operational Demonstration," and it reports one local run of four endpoints. There is no sample size because $n = 1$ per endpoint. There is no baseline because direct provider API use is never instrumented for comparison. There are no statistics because nothing is measured. The only literal observation reproduced anywhere in the text is `"ok": true` in §5.2. The paper is aware of this and says so in §5.5 and §6, but awareness is not a substitute: §1.1 contribution 3 and §7 paragraph 2 both use the verb "demonstrates," and what was demonstrated is that four endpoints return responses, which is the definition of the software compiling and running.

2.2 The word in the title is the one thing never tested. "Deterministic" appears 16 times in the paper by my count. §2 defines the operative version: *"Given the same software state and the same external configuration path, the request-handling pipeline follows an explicit sequence."* That is a falsifiable statement, and it is cheap to test. Run the same input through the same configuration k times, hash the composed request frame and the config snapshot in each resulting cycle artifact, and show that the hashes are identical while the output fields differ. That experiment would take an afternoon and would convert the paper's central adjective from an assertion into a measurement. It was not run. As the text stands, procedural determinism is argued from the source code's structure, not from observed behaviour, and §5 does not contain a single repeated execution.

2.3 "Auditable" is undercut by the retention model. The paper uses "auditab*" 22 times, yet the retrieval surface is `/ame/last`, described in §3.3 and §4 as returning *"the most recently recorded cycle artifact"*, with §6 confirming the mechanism is *"narrow and intentionally focused on the latest cycle"*. An audit trail that holds one entry is not an audit trail. Each new call destroys the evidence of the previous one, which is the opposite of what an auditor needs. Worse, the paper never says whether earlier cycles are persisted to disk and merely not exposed, or discarded entirely. §3.1 names a "Cycle recording layer" that "persists a structured representation", and §3.4 speaks of a bounded record, but I could not determine from the text whether persistence is append-only, whether it survives a process restart, or where it is written. This is the single most consequential gap: the difference between "persists all cycles, exposes the latest" and

"keeps one cycle in memory" is the difference between an audit system and a debug convenience, and the paper does not tell me which one it built.

2.4 Nothing addresses the properties that make a record admissible as evidence. If the claim is auditability, the relevant literature and practice ask for integrity (hash chaining or signing, so a record cannot be edited after the fact), retention policy, access control, and redaction of sensitive content. None of the four appears anywhere in the paper. The cycle artifact stores the user input and model output verbatim (§3.4), so in any realistic setting it is a store of personal or confidential text, and §6 does not list privacy among its limitations. Nor is authentication mentioned for /chat. §5.2 says the /ame/config response exposes *"the selected model identifier, runtime parameters such as temperature and token limits, and a system-level template"*: an unauthenticated configuration endpoint that returns the system prompt is a design decision with consequences, and it passes without comment. I am not asking the author to solve these problems in a minimal system. I am asking that a paper whose title word is "auditable" state which of them it deliberately leaves open.

2.5 The claim of lightness is unquantified. "Lightweight" occurs 11 times and "minimal" 42 times. Not one of those occurrences carries a number. §3.4 says meta holds *"timestamps, latency, and provider-related information"*, and §5.2 says the retrieved cycle included *"execution metadata such as timestamp and latency"*, so latency was recorded during the demonstration and then not reported. That is a missed opportunity of the cheapest kind. Two columns (direct provider call, call through MiND) with median added latency in milliseconds, plus the on-disk size of a cycle artifact, would substantiate "lightweight" at almost no cost. The paper also never states the provider, the model identifier, the temperature value, or the token limit actually used, so a reader cannot even judge what fraction of end-to-end time the middleware could plausibly account for.

2.6 The novelty claim cannot be evaluated because there is no related-work section. This is a structural omission, not a stylistic one. The paper has eight numbered sections, and none of them surveys prior art. The bibliography holds five entries: [1] Vaswani et al. on Transformers, [2] Bommasani et al. on foundation models, [3] Bender et al., [4] Pineau et al. on the NeurIPS reproducibility programme, and [5], which is the author's own software repository. Four external citations, none of which is about request logging, tracing, or LLM observability. There is an established body of work and tooling in exactly this space (distributed tracing and its generative-AI semantic conventions, experiment-tracking systems, LLM gateway and proxy layers that log prompts, configurations, and responses), and a reader's first question is what MiND does that a tracing exporter or a logging proxy does not. §7 asserts a niche:

"Between direct prompting and full orchestration frameworks, there is a technical space for small systems that improve procedural visibility without demanding large architectural commitments." An asserted niche is not a demonstrated one. Without a comparison table, the contribution claim in §1.1 is unfalsifiable.

2.7 Smaller inconsistencies that a reader will trip over. The endpoint prefix */ame/* appears four times (§3.3, §5.2 twice, §5.3 context) and is never expanded or explained; it is the only piece of naming in the paper without a gloss. The naming story more broadly is confusing: §1 says the acronym *"originally derives from 'Minimal Nemosine Design'"* and is *"used in its software-publication sense"* as something else, the footnote places the work inside a *"Nemosine Nous research program"*, the repository is *nemosine-10-MiND*, and the package is *nemosine-mind*. A reader trying to find the artifact has to reconcile three names. Separately, §5.3 lists five properties that largely restate the four endpoint observations of §5.2, and the phrase *"minimal executable reference"* occurs 10 times across §1, §2, §3.5, §4, §5.4, §6, and §7. As rendered on Qeios, Figures 1, 2, and 3 appear only as caption lines (*"High-level architecture of MiND"*, *"Request-handling cycle in MiND"*, *"Structure of a retrievable MiND cycle artifact"*), so I could not assess their content, and none of my comments should be read as a judgment on them.

3. Reproducibility

The good part first, and it is real: the artifact is public at two addresses (§8 gives the GitHub repository and the PyPI package), the framework is named (FastAPI, §4), and the endpoint surface is specified precisely enough that a reader could re-derive the interface. For a short software paper, that is a decent starting position. What follows is what I could not reconstruct.

No version pins anywhere. §8 states that *"implementation details such as package versioning, default templates, and retrieval scope may evolve across releases"* and then directs the reader to treat the repository as the primary source. So the paper describes a moving target with no commit hash, no release tag, no PyPI version number, and no archived snapshot with its own DOI. Reference [5] cites the repository as *"Software repository"* with no identifier. The result is that the version of MiND this paper is about cannot be recovered in a year's time. For a paper whose subject is traceability, this is the defect I find hardest to accept.

No model version, no decoding parameters, no provider. The word *"temperature"* appears once in the whole paper (§5.2) and appears as a category name, not a value. The provider is never named. The model identifier is never given. Token limits are referred to and never stated. No seed is mentioned. This is the usual pattern in LLM papers, and here it has an extra sting because the paper's own thesis is that

configuration should be externalized and inspectable: the reference implementation exposes a configuration endpoint, and the paper does not print what that endpoint returned.

No prompts. The system template is central to the design (§2, fifth principle; §4, third paragraph), §6 concedes it may carry project-specific language, and its text appears nowhere. One appendix with the default template verbatim would settle the question of how neutral it is.

No example artifact. §3.4 and §5.2 describe the cycle record in prose. A single JSON block showing a real (redacted) cycle with its `cycle_id`, config snapshot, and meta fields would do more for the paper than either passage and would let a reader check the schema against the claims.

No environment or compute description. §5.1 says only that the service was *"launched as a local FastAPI service"*. Python version, operating system, hardware, and the date of the run are all absent. The last one matters: provider-side models change, and a demonstration with no date cannot be situated.

No procedure that a third party could repeat. There is no listed sequence of commands, no `curl` examples, no configuration file shown. §5 is a narrative account of a session rather than a protocol. I could not verify any of it from the text, and I want to be clear that I am not alleging the run did not happen: I am saying the paper gives me nothing to check it against.

4. What specifically to fix

1. **§5, new subsection: test the determinism claim.** Execute the same input under the same configuration k times ($k = 30$ is plenty), hash the composed request frame and the config snapshot from each cycle artifact, and report that all k hashes match while the output fields differ. Report the hash function and the k value. This single experiment is what converts the title's central word from an assertion into a result, and the paper cannot carry that word without it.
2. **§3.4 and §6: settle what happens to cycles other than the last one.** State explicitly whether prior cycles are written to durable storage, where, in what format, and whether they survive a restart; or state that they are overwritten. If they are overwritten, the abstract and the title need to stop using "auditable" without qualification, because a one-entry log is not an audit trail.
3. **New §2 (renumber the rest): related work.** Compare MiND against at least four named alternatives from the tracing, gateway, and experiment-tracking families, in a table whose rows are the properties this paper cares about: externalized configuration, per-call record, history depth, integrity guarantee, provider coupling, lines of code. The four current external references ([1]–[4])

do not cover this ground; [1] and [3] in particular are cited in §1 for general framing and carry no weight on the design question.

4. **§5.2: report the numbers you already collected.** Latency is in meta by the paper's own account. Give median and p95 added latency over a run of at least 100 cycles, against a direct provider call as the control, and give the byte size of a cycle artifact. That substantiates "lightweight," which currently appears 11 times without a single measurement behind it.
5. **Add an appendix with the default system template, verbatim.** §6 admits it may encode project-specific assumptions; print it and let readers judge.
6. **Add a second appendix with one complete cycle artifact as JSON,** redacted if necessary, showing `cycle_id`, input, the full config snapshot (provider, model identifier, temperature, token limit, mode), and all meta keys.
7. **§8 and reference [5]: pin the version.** Archive the exact commit described in this paper (Zenodo or Software Heritage), cite that archived snapshot with its identifier, and give the PyPI version number in the text. As written, §8's warning that details "may evolve across releases" leaves the paper describing a version that no longer exists.
8. **§3.3 and §5.2: explain or drop the /ame/ prefix,** and reconcile the three names (MiND, Nemosine Nous, *nemosine-mind*) in one sentence in §1 rather than across a footnote, §1, and §8.
9. **§6: add a security and privacy paragraph.** Say whether `/chat` and `/ame/config` require authentication, what the configuration endpoint withholds (API keys, in particular), and what the intended retention and redaction policy is for stored inputs and outputs. Two honest paragraphs are enough; silence is not.
10. **§1, §2, §3.5, §4, §5.4, §6, §7: cut the repetition.** "Minimal executable reference" occurs 10 times, "minimal" 42, and the distinction between procedural and semantic determinism is restated in at least five places. State it once in §1, once more when it does analytical work, and delete the rest. The paper would lose perhaps a fifth of its length and gain considerably in force. §5.3 should also be folded into §5.2, which it duplicates.
11. **§1.1 contribution 3 and §7, paragraph 2.** Replace "demonstrates" with what the evidence supports. One local run of four endpoints shows that the implementation runs; it does not show that auditable interaction has been operationalized. After fix 1 and fix 4 are in place, the stronger verb becomes available.
12. **Define the auditor.** Nowhere does the paper say who inspects these records and against what requirement: a developer debugging a prompt, a researcher reconstructing a session months later,

or a compliance function answering to an external record-keeping obligation. These three readers need different things, and naming the target one in §1 would let the reader judge whether the four-field artifact is sufficient.

5. Recommendation

Major revisions.

My reasoning, briefly. The problem MiND addresses is real, the author's restraint about scope is genuine and rarer than it should be, and the artifact is public. That is enough to keep the paper in play. But three things block acceptance in the present form, and each of them is fixable with work the author is clearly capable of doing. The title's central claim, procedural determinism, is never tested, though a repeated-execution test would take an afternoon (fix 1). The second claim, auditability, is contradicted in practice by a retrieval surface that exposes one record and by the absence of any integrity, retention, or access story (fixes 2 and 9). And with no related-work section and only four external references, none of them about logging or tracing, the novelty of the design cannot be assessed at all (fix 3).

I do not recommend rejection. Small, honestly scoped software papers have a place, and this one avoids the inflation that usually sinks them. What it lacks is the modest empirical layer that would let its own adjectives stand: repeat the run, measure the overhead, print the template and one artifact, pin a version, and compare against the tools that already occupy this space. With those in place, the paper becomes a short, credible engineering note that a reader can act on. Without them, it remains a description of a program rather than a finding about one.

Declarations

Potential competing interests: No potential competing interests to declare.