Deterministic Identifiers for Large Language Models: Reducing Citation Loss from Identifier Transcription Errors

J. R. Wells
Busy, Inc.
hello@busy.inc

Working draft — August 21, 2026
(DTID format introduced in production November 2025)

Abstract

Some citation failures in retrieval-augmented systems are not retrieval failures. They are identifier failures: long random strings fracture into messy tokens, drift across model turns, and come back one character wrong. When a database expects one exact identifier and the model returns a nearby variant, the citation cannot be resolved and the citation fails to render. We propose treating identifiers as part of the model interface and describe a deterministic, tokenizer-friendly format — five zero-padded three-digit groups (e.g. `263:037:515:764:525`) — that replaces the LLM-facing surface of a UUID while leaving internal identifiers unchanged. Under both current OpenAI tokenizers, every DTID encodes to exactly 9 tokens with a single fragmentation shape, where UUIDs average 22.7 tokens with no two sharing a shape. On a sorting benchmark that isolates transcription fidelity we find two distinct effects at two scales. At 100 identifiers, small models transcribe DTIDs more accurately than UUIDs (21.0%→8.7% error on one), while frontier models sit at zero under both formats — the task is too small to separate them. At 1,000 identifiers, the scale a production citation pipeline actually reaches, a second and larger effect appears on exactly those frontier models: gpt-5.5 recovered *zero* of 1,000 citations under UUIDs, abandoning or declining the task in all three trials, and a perfect 1,000 under DTIDs in all three; gemini-2.5-pro lost citations on every UUID trial and none on any DTID trial. The failure at the frontier is not corruption but abandonment. We also report a failure mode the format introduces, without claiming a mechanism: on several older small models, the regularity of DTIDs invites wholesale fabrication of well-formed identifiers, which exact-match validation must catch. The format was developed in production, in response to citations that resolved correctly at retrieval and failed at render.

1 Introduction

In a retrieval system, it is natural to blame missing citations on bad ranking, bad embeddings, or missing source data. But in practice, some failures happen after retrieval succeeds. The model finds the right citation, carries its identifier through multiple tool calls and model turns, and then emits a string that is almost right.

Almost right is still broken. If the database expects one exact identifier and the model returns a nearby variant, the frontend cannot resolve it. The citation vanishes. The user sees a plain sentence instead of a linked, inspectable source. From the outside it looks like the model never found the source at all.

That distinction matters because it changes the solution. If the failure is in recall, you improve search. If the failure is in representation, you improve the identifier. And just as important, you make the failure

legible enough that engineers can tell where the system went wrong instead of treating every broken citation like a retrieval mystery.

This paper makes three contributions:

1. a characterization of identifier transcription as a distinct failure mode in LLM citation pipelines, separate from retrieval failure;
2. a deterministic, tokenizer-friendly identifier format (DTID) designed to survive repeated transcription across model turns; and
3. a seeded, multi-trial benchmark that isolates transcription fidelity, evaluated on 14 models across two model generations, with production evidence from a deployed system.

2 Related Work

Provenance of this work. The DTID format was designed and put into production in November 2025 in response to citation failures observed in a live retrieval system. It was not derived from the academic literature, and no prior work informed its design. We state this plainly because a related-work section conventionally signals intellectual debt, and there is none to signal here.

Adjacent fields, and why none of them is cited here. A survey conducted at draft stage identified five bodies of work that touch this problem from other directions: tokenization effects on character-level manipulation; citation grounding and attribution in retrieval-augmented generation; citation hallucination at scale; long-context degradation benchmarks; and grammar-constrained decoding. None of it informed the design, and none of it makes a claim this paper depends on, so we do not cite it as lineage. Readers familiar with those literatures will recognize the neighborhood; readers who want the survey can find the compiled list, with sources verified against primary records, in the `busy-research` repository alongside the data.

On novelty. A search of arXiv and the ACL Anthology conducted for this draft found no published academic work treating identifier format as a design concern for language-model interfaces. Identifiers appear in that literature incidentally — for instance as memorization canaries — but not as an object of design. Informal prior art does exist in practitioner writing and open-source tooling, including schemes that re-encode UUIDs as checksummed word sequences so that agents reproduce them correctly; we note this rather than claim an unqualified first. The contribution here is formalization and measurement, not priority.

3 The Tokenization Problem

LLMs do not read strings the way humans do. They read tokens: irregular chunks chosen by a learned tokenizer. Common words often map cleanly. Unfamiliar strings do not. A UUID tends to shatter into inconsistent pieces: one token for part of a hex group, another for a hyphen plus a letter, another for a trailing fragment that means nothing on its own. The UUID `f1166652-812d-480d-8d11-5495e8b92e31`, for example, fragments into irregular chunks with irregular punctuation — easy to transpose, and hard to notice when one symbol drifts.

That fragmentation creates two layers of difficulty. First, the model has more pieces to preserve. Second, those pieces do not correspond to semantic units. There is no human-like sense that one chunk belongs with the next. The string is just statistical debris.

A fixed pattern of three-digit groups changes that. The tokenizer tends to preserve each triplet as an atom. The resulting sequence is not shorter only in characters; it is cleaner in model-space.

Table 1 makes this quantitative: over 1,000 seeded identifiers per format, every DTID encodes to exactly 9 tokens with one identical token-boundary shape, under both OpenAI tokenizers we can measure locally. The 1,000 UUIDs average 22.7 tokens each — and produce 1,000 distinct fragmentation shapes: no two UUIDs in the sample fragment the same way. DTIDs are also 54% fewer tokens per identifier, so identifier-dense prompts get cheaper as a side effect.

Table 1: Tokenizer fragmentation over 1,000 seeded identifiers per format (tiktoken). Distinct shapes = unique token-length sequences in the sample; 1 means every identifier fragments identically.

Encoding	Format	Mean tokens	Min–max	Distinct shapes
cl100k_base (GPT-3.5/4)	UUID	22.7	17–29	1000
cl100k_base (GPT-3.5/4)	DTID	9.0	9–9	1
o200k_base (GPT-4o/4.1)	UUID	22.7	17–29	1000
o200k_base (GPT-4o/4.1)	DTID	9.0	9–9	1

4 The DTID Format

The proposed format is intentionally plain. Instead of a 36-character UUID, use five groups of three digits: `263:037:515:764:525`. This is long enough to avoid practical collisions at citation scale, but structured enough that a model can carry it through a conversation without constantly breaking it apart.

The design reduces to four principles:

Fixed width. Every group should look identical from the model's perspective. Zero padding matters because it removes visual ambiguity.

Stable separators. Hyphens or colons are fine. The key is a repeating rhythm the model can predict and preserve.

Enough entropy. Five groups of three digits gives about 10^{15} combinations: plenty for citation-scale identifiers without turning them back into noise.

Optional determinism. The same shape can be random or derived from a stable source value like a record id or SHA, depending on the use case.

The deeper point is not the punctuation. Hyphens, colons, brackets, or wrapper syntax can change by surface. What matters is the underlying shape: a repeated sequence of compact, fixed-width atoms. A reference implementation is small enough to give in full:

```
class DtidUtility
  def self.generate_random
    5.times.map { format("%03d", rand(1000)) }.join(":")
  end

  def self.generate_from_sha(input)
    digest = Digest::SHA256.hexdigest(input.to_s)
    digits = digest.scan(/[0-9a-f]{3}/).first(5)
```

```
                    .map { |chunk| chunk.to_i(16) % 1000 }
    digits.map { |n| format("%03d", n) }.join(":")
  end
end
```

Once the shape is stable, the rest of the system gets easier to inspect. Human debugging gets easier. Prompt examples get easier. Render-time validation gets easier. If a model drifts, the drift is visible, which reduces a lot of the chaos in understanding whether the failure lived in retrieval, transport, prompting, or rendering.

5 Benchmark Design

A useful stress test is simple: give the model a large set of identifiers and ask it to sort them. The task forces the model to read, preserve, and re-emit every identifier — pure pass-through copying cannot satisfy it — and makes transcription errors measurable.

Protocol. The benchmark runs at two scales: $N = 100$ across fourteen models (Section 6) and $N = 1{,}000$ across four frontier models (Section 7). Each trial presents N identifiers, one per line, each wrapped in ⟦⟧[1] (the citation-tag delimiter the production system uses). The prompt, verbatim:

> `Below are [N] identifiers, one per line, each wrapped in ⟦⟧.`
> `Sort them in ascending lexicographic (plain string) order and return the complete sorted list, one identifier per line, each wrapped in ⟦⟧. Return only the sorted list, nothing else.`

Identifier sets are generated from a seed derived from (condition, trial), so every model sees identical inputs and the sets regenerate bit-for-bit. Five independent trials per model per condition at $N = 100$ and three at $N = 1{,}000$, temperature 1, models accessed through OpenRouter in August 2026. At $N = 1{,}000$ reasoning tokens are explicitly budgeted, because an unbounded reasoning model can otherwise spend its entire completion allowance thinking and return nothing. A UUID trial is roughly 2,530 prompt tokens and 2,469 completion tokens; a DTID trial roughly 1,162 and 1,145.

Scoring. All ⟦⟧-wrapped strings are extracted from the response. An emitted identifier is *valid* if it exactly matches an input identifier. Invalid identifiers are classified as *mutated* (Levenshtein distance ≤ 3 from some input identifier — transcription damage) or *hallucinated* (fabricated).

Separating fidelity from completeness. The original test's metric, $1 - \text{valid}/N$, conflates two independent failures: the model corrupting identifiers it returns, and the model not returning them at all. These are different phenomena with different causes, and at scale they dissociate sharply — we observe frontier models that transcribe several hundred identifiers with perfect fidelity and then simply stop, cleanly, well short of the task. Charging that abandonment to the identifier format would misattribute a task-compliance failure to transcription. We therefore report two metrics:

Fidelity error the share of *returned* identifiers that do not exactly match an input identifier. This is the transcription-quality measure, and it is the paper's primary metric.

[1] The delimiters are the CJK corner brackets U+3010/U+3011, rendered here as ⟦ ⟧.

Completeness the share of the input set returned correctly. A run is marked *abandoned* when the model stops cleanly — not at a token ceiling — having returned under 90% of the task.

We report abandonment as its own outcome rather than discarding abandoned runs. Dropping them would be the more convenient choice, but abandonment may itself be format-dependent: the two conditions differ in output length for the same task (a 1,000-identifier DTID list is roughly 9,000 tokens against roughly 22,700 for UUIDs), so silently excluding these runs could conceal a real effect of the format on task compliance. Separately, runs truncated by the harness's own token ceiling rather than by model behavior are excluded and re-collected with a higher ceiling; these are instrumentation artifacts and carry no signal about either model or format. Two further failure modes are flagged per run: *runaway* (more than $1.2N$ identifiers emitted) and *loop* (any single identifier emitted ten or more times).

The harness, all raw responses, and the scoring code are in the `busy-research` repository.

6 Results

Table 2 reports mean error rates with 95% confidence intervals over five trials per cell, for the models of the DTID invention era (November 2025) and, separately, the current generation (August 2026). Two findings emerge.

DTIDs roughly halve transcription error, concentrated where it matters. Pooling clean runs (runaways excluded) across all models, the UUID condition errs on 5.5% of identifiers and the DTID condition on 2.5%. The reduction concentrates on small, inexpensive models — the ones a citation-volume pipeline would actually run: gpt-4.1-nano falls from 21.0% to 8.7%, gpt-3.5-turbo from 12.0% to 6.8%, gemini-2.5-flash-lite from 2.4% to 0.4%, and gpt-5.4-mini from 8.0% to 0.2%.

At 100 identifiers the effect is an accuracy effect, and it is confined to small models. Ordering the evaluated models by capability, the benefit of the DTID format tracks inversely: large reductions on the weakest models (gpt-4.1-nano, gpt-3.5-turbo), moderate reductions in the middle tier (gemini-2.5-flash-lite, gpt-5.4-mini), and no measurable difference at the frontier, where both formats sit at or near zero. We initially read the frontier result as the effect disappearing with capability. Section 7 shows that reading was an artifact of the task size.

Format regularity introduces a fabrication risk on weak models. Nine of 69 DTID runs — on claude-3-haiku, gpt-4o-mini, and gpt-4.1-nano — stopped transcribing and emitted long sorted sequences of well-formed identifiers that never appeared in the input (300+ per run; 1,416 fabricated identifiers across the DTID condition versus 42 across UUID). No UUID run exhibited this. A regular format is easier to carry and also easier to fake convincingly; exact-match validation at render time remains mandatory under either format. We report this as an observation; we do not have evidence sufficient to identify its mechanism, and it does not co-occur reliably with the accuracy effect above (only two of the seven affected models exhibit both).

The August 2026 generation narrows but does not close the gap: its small models still err on UUIDs (8–9% on the gpt-5.4 nano/mini tier) and still benefit from the format (97.5% reduction on gpt-5.4-mini), while gpt-5.4-nano shows no improvement within noise. The original November 2025 single-run observations (1.1%/0.0% on Gemini 2.5 Pro at $N = 1{,}000$; 8.5%/1.2% on Gemini 2.5 Flash) are consistent with the rerun's direction on the same model family.

Table 2: Sorting benchmark, mean error rate ± 95% CI over 5 trials per cell, $N = 100$, temperature 1. "Clean" marks cells where runaway runs are excluded (runaway counts shown). Era = model generation. claude-sonnet-5 completes this task cleanly at $N = 100$; it is the larger task in Section 7 that it could not be measured on.

Era	Model	UUID error	DTID error	Runaways (DTID)
2025	gpt-3.5-turbo	12.0% ± 2.9	6.8% ± 3.2	0
2025	gpt-4o-mini	12.8% ± 5.6	11.0% clean	3 of 5
2025	gpt-4.1-nano	21.0% ± 14.4	8.7% clean	2 of 5
2025	gpt-4.1	0.0%	0.0% clean	1 of 5
2025	gemini-2.5-flash-lite	2.4% ± 3.1	0.4% ± 0.7	0
2025	gemini-2.5-flash	0.6% ± 0.7	0.0%	0
2025	gemini-2.5-pro	0.0%	0.0%	0
2025	claude-3-haiku	5.6% ± 3.2	3.0% clean	3 of 5
2025	claude-haiku-4.5	0.4% ± 0.7	0.0%	0
2026	gpt-5.4-nano	9.0% ± 5.5	11.2% ± 16.1	0
2026	gpt-5.4-mini	8.0% ± 18.8	0.2% ± 0.6	0
2026	gemini-3.5-flash-lite	0.6% ± 1.1	0.2% ± 0.6	0
2026	gemini-3.7-flash	0.0%	0.0%	0
2026	claude-sonnet-5	0.0%	0.0%	0

7 Scale: What Happens at 1,000 Identifiers

The frontier models' indifference to format at $N = 100$ invites an obvious conclusion — that capable models have absorbed the problem — and that conclusion is wrong. It reflects a task small enough that neither format is difficult. Raising the task to $N = 1{,}000$, the scale at which the format was originally deployed and at which real citation pipelines operate, produces a second effect on precisely those models, and a larger one.

We re-ran the same protocol at $N = 1{,}000$ on four frontier models, three trials per condition, with reasoning explicitly budgeted so that a model's thinking could not consume its own answer. Table 3 reports identifiers recovered intact out of 1,000.

Table 3: Identifiers recovered intact out of 1,000, three trials per condition. Individual trials in parentheses. The failure at this scale is abandonment, not transcription: fidelity error among returned identifiers remained at or near zero in every completed run.

Model	UUID	DTID	Note
gpt-5.5	**0** (0, 0, 0)	**1000** (1000, 1000, 1000)	Quit or declined every UUID run
gemini-2.5-pro	996 (999, 994, 996)	**1000** (1000, 1000, 1000)	Lost citations on every UUID run
gemini-3.1-pro	904 (997, 995, 721)	998 (999, 997, 999)	
claude-sonnet-5	Did not complete the task in either format		See below

Three results stand out. First, DTIDs produced no transcription errors at all on any model that completed the task, in any trial. Second, gpt-5.5 recovered nothing under UUIDs across three trials — it did not corrupt the identifiers, it abandoned the work, in one case declining explicitly: *"I can't reliably sort and return all 1000 identifiers manually without risking omissions or ordering errors."* Handed the identical task in the shorter format, the same model returned a complete, correctly sorted 1,000 three times out

of three. Third, gemini-2.5-pro — the model on which the format was originally developed in production — lost citations on all three UUID trials and none of the three DTID trials, reproducing the direction of the original November 2025 observation.

An unmeasurable cell. claude-sonnet-5 could not be evaluated at this scale. Under every configuration we tried, including an explicit reasoning-token budget and an effort-level control, it spent its entire completion allowance on reasoning and emitted no answer, or emitted a single identifier. We report this as a limitation of our harness rather than a property of the model, and we do not score it in either direction.

Interpretation. The two scales measure different things. At $N = 100$ the format changes how accurately a weak model copies. At $N = 1{,}000$ it changes whether a strong model attempts the work. Both are consequences of the same design choice, and the second is the one that matters operationally: a pipeline whose model silently declines a thousand-citation task fails completely, not partially.

8 Why Identifiers Circulate

Single-turn prompting already stresses identifiers. Multi-step agents stress them far more. A citation id might move from retrieval output to tool result, from tool result to planner, from planner to model response, from model response to renderer. Every hop is a chance to mutate the string. That is one reason production systems can look worse than toy benchmarks: the identifier is not merely generated once. It circulates.

This is also why the failure is easy to misdiagnose. A system that resolves citation tags against a database at render time reports only that a tag did not resolve. It cannot distinguish a retrieval miss from a transcription error, so both arrive in the same bucket and the natural response — improve retrieval — leaves the second cause untouched. The format was originally developed in response to exactly this pattern in a live system: citations that had been retrieved correctly, then lost somewhere between retrieval and render. The benchmark in this paper exists to isolate that second cause under controlled conditions, and Sections 6 and 7 report what it costs.

9 Discussion: Identifiers as Model Interface

Traditional software treats identifiers as back-office implementation details. In LLM systems, that is no longer true. The identifier is now part of the model interface. It lives inside prompts. It gets copied by a generative model. It is parsed by a renderer. That means identifier design belongs next to prompt design, tool design, and output-schema design.

The right mental model is not only "make IDs unique." It is also "make IDs survivable." Uniqueness is table stakes. Survivability makes the system easier to reason about when something goes sideways.

That does not mean every system should abandon UUIDs globally. It means systems should be willing to introduce an LLM-facing identifier layer where exact string fidelity matters. The implementation pattern is to keep the alias close to the data model: create a dedicated citation-identifier mapping table, generate the deterministic id once, validate uniqueness, and let prompts speak only in that alias. The model never needs to see the raw UUID. The format is debuggable by humans too: when an identifier is visually structured, engineers can spot transpositions faster, compare examples faster, and write sanity checks that match the mental model of the system.

The fabrication result sharpens the design guidance rather than undermining it. The identifier layer improves the *copying* problem; it does not remove the need for the resolver to reject identifiers that do

not exist. A system adopting DTIDs should keep exact-match validation at render time, and may want a cheap structural safeguard — a checksum group, or resolver-side rejection of identifiers that validate structurally but miss the database — precisely because a well-formed fake now looks indistinguishable from a real identifier.

The operational value depends on where the system sits. For a pipeline running small models at modest per-request identifier counts, the value is accuracy: a format change that moves identifier error from 21% to 8.7%, at no inference cost and with roughly 54% fewer tokens spent on identifiers, changes which model a system can afford to deploy. For a pipeline running frontier models over large identifier sets, the value is completion: whether the model returns an answer at all. The second case is the more consequential failure, because a declined task produces no partial result to salvage — the pipeline returns nothing and the reason is invisible unless someone reads the model's prose.

10 A Hypothesis for Future Work: Representational Overhead

The measurements above establish *that* identifier format affects small-model accuracy. They do not establish *why*. One hypothesis is worth stating explicitly, as a target for future work rather than a claim of this paper.

Carrying an irregular string may impose a cost that competes with the task itself. Every UUID is 22.7 tokens of unpredictable, mutually unrelated fragments; nothing about the four-hundredth identifier assists in producing the four-hundred-and-first. A DTID is nine tokens in a shape the model has already seen a thousand times. If holding the former consumes capacity that the latter does not, then the accuracy gradient across model sizes follows directly: a model with abundant capacity absorbs the overhead invisibly, and a small model does not.

Two observations in our data are consistent with this account without confirming it. First, the benefit scales inversely with model capability, which is what a capacity-competition story predicts. Second, in the UUID condition, models under-generate — returning fewer identifiers than requested — in 10 of 66 runs, against 1 of 69 under DTIDs ($p = 1.2 \times 10^{-4}$, Fisher exact), as though the harder format were being abandoned rather than completed.

We flag two reasons for caution. The under-generation signal is a property of the sorting task, in which quitting manifests as returning fewer items; production citation emission has no equivalent behavior, and the result should not be read as a prediction about live systems. And the fabrication effect does not co-occur reliably with the accuracy effect at the model level, so a single unifying mechanism is not supported by these data. Discriminating between a capacity-competition account and simpler alternatives — for example that digit-triplet sequences are simply better represented in training data — requires experiments this paper does not run.

11 Conclusion

LLM systems fail on small things that classical software barely notices: a separator, a token boundary, a repeated shape that the model can or cannot hold in working memory. Those details feel cosmetic until they become the difference between a working citation and an invisible one.

Deterministic identifiers are a narrow idea, but they point at a broader lesson. If a string has to survive inside a language model, its form matters. Human readability matters. Tokenizer regularity matters. Repetition matters. In agentic systems, representation is part of the architecture.

Data and Code Availability

The complete harness, every raw model response, the scoring code, and the generated result tables are in the `busy-research` repository under `dtid/`. Identifier sets are generated from seeds derived from (condition, trial), so every input set in this paper regenerates bit-for-bit without being stored. Reproducing the study requires an OpenRouter API key; total API spend for the results reported here was $3.70.

Models were accessed through OpenRouter in August 2026 at the version each slug resolved to on that date. Tokenizer measurements use `tiktoken` with the `cl100k_base` and `o200k_base` encodings.

This paper cites no prior literature; see Section 2 for why.