

Feedback Blindness: Diagnosing Silent Failures in Closed-Loop Agentic Data-Generation Pipelines

Nguyen Minh Loc^{1,*}, Nguyen Khanh Long¹, Vu Gia Nam¹, Nguyen Tran Minh Nhat¹

¹Hanoi University of Science and Technology, Vietnam

*Corresponding author: loc.nm235765@sis.hust.edu.vn

Abstract

Closed-loop synthetic-data pipelines combine planning, LLM generation, verification, and deduplication; accepted outputs steer later generation. We study a production pipeline for Vietnamese contact-center compliance data and identify *feedback blindness*: an internal feedback mechanism stops serving its intended function while every stage completes normally, leaving only downstream metric degradation as evidence. We diagnose four instances: cumulative-count eviction empties a recency window; batch-level deduplication misses candidates scored before any batch member is committed; static boundary exemplars repeatedly anchor generation; and a fixed-size anti-repeat reference set loses informativeness as the dataset grows. We describe architecture-level fixes, mechanism-specific health signals, and a controlled ablation and scaling evaluation. The central lesson is that aggregate duplicate and acceptance rates are insufficient for diagnosis; pipelines should also monitor the feedback mechanisms that produce those outcomes.

1 Introduction

Synthetic-data pipelines combine generate-then-verify methods (Wang et al., 2023; Sengupta et al., 2025) with adaptive planning and embedding-based deduplication. Accepted samples then update coverage estimates, influence later prompts, and enter the index used to admit future candidates. This closed loop is intended to be self-correcting, but only while its feedback mechanisms remain informative.

We study a production pipeline for Vietnamese contact-center compliance data and find that these mechanisms can degrade without producing an exception or malformed component output. A recency window can collapse, candidates within the same batch can remain mutually invisible to deduplication, boundary exemplars can become stale anchors, and a fixed anti-repeat reference set can

cover a vanishing fraction of accumulated data. The generator, verifier, and filter still complete normally; only duplicate rate, acceptance rate, and cost drift later. We call this failure class *feedback blindness*. The phenomenon relates to hidden technical debt, data cascades, and degradation in recursive generative systems (Sculley et al., 2015; Sambasivan et al., 2021; Shumailov et al., 2024). Our focus instead lies in pipeline infrastructure, specifically eviction, commit ordering, exemplar selection, and reference sizing, rather than model weights or annotation processes. These mechanisms are directly inspectable and fixable, but aggregate output metrics do not identify which one has failed.

This paper makes three contributions: (1), We define feedback blindness and give a four-part taxonomy stated in terms of general architectural preconditions. (2), We provide fixes and health signals that expose regressions before aggregate metrics visibly degrade. (3), We specify ablation, scaling, and downstream evaluations of efficiency and data quality.

2 Related Work

LLM-based synthetic data generation pipelines.

Large language models have become an effective mechanism for constructing synthetic datasets for downstream NLP tasks, particularly in domains where manual annotation is expensive or sensitive. Early work such as Self-Instruct (Wang et al., 2023) demonstrated that LLMs can bootstrap instruction-following data from a small set of seed examples. The survey of Long et al. (2024) organizes later methods into a workflow of planning, generation, curation, evaluation, and feedback, reflecting a shift from one-shot prompting toward automated pipelines. LoopTool closes the data-training loop by using model failures and verifier feedback to guide subsequent data expansion (Zhang et al., 2026). Whereas LoopTool studies how a function-

ing closed loop can improve a model, we study how the loop’s own infrastructure can silently stop providing useful feedback.

Quality control in agentic data-generation systems. Modern synthetic-data pipelines rely on multiple quality-control stages, including LLM verification, duplicate filtering, and automatic cleaning workflows. MAG-V uses separate generation and verification agents for synthetic-data quality control (Sengupta et al., 2025). Related verifier research improves the verification signal itself: Weaver combines multiple weak verifiers (Saad-Falcon et al., 2025), while DnDScore studies decomposition and decontextualization in factuality verification (Wanner et al., 2025). AutoDC-Workflow treats data cleaning as an iterative LLM-generated workflow (Li et al., 2025). Deduplication work shows that removing repeated training text improves language-model behavior (Lee et al., 2022), but generally treats deduplication as corpus cleaning rather than as mutable shared state inside a live generation loop. Our concern is complementary: the modules may continue to execute and emit valid outputs while the feedback mechanisms connecting them lose effectiveness.

Stream processing and cache monitoring. The diagnosed failures are closely related to state-management problems in streaming and caching systems. MillWheel manages persistent state and record delivery under fault-tolerance guarantees (Akidau et al., 2013). MIMIR profiles cache hit rates as a function of memory capacity to guide provisioning (Saemundsson et al., 2014). Our contribution is not a new eviction policy or general monitoring technique, but an account of how state-management failures disable feedback in agentic generation. Accepted samples alter subsequent plans and prompts, so stale or incomplete state changes what is generated next. We therefore examine feedback usefulness, rather than execution recovery or cache provisioning.

Reliability of ML systems. Hidden technical debt (Sculley et al., 2015) and data cascades (Sambasivan et al., 2021) provide a broader framing for system-level degradation. Model collapse concerns degradation from recursive training on model-generated data (Shumailov et al., 2024). In contrast, our failures occur within the generation pipeline, before its output is used for downstream training.

3 System Description

Our pipeline (Figure 1) generates labeled Vietnamese text for a four-class classification task (otp, password, pin, normal) describing whether a contact-center agent’s utterance improperly requests sensitive authentication information from a customer. It contains four processing stages, a planner, generator, verifier, and deduplication stage, followed by acceptance and storage. Accepted samples update several upstream feedback mechanisms.

Planner. An attribute planner samples, for each generation plan, a value along each of several configurable dimensions (e.g. *scenario*, *politeness*, *urgency*, *wording*) in addition to a target label. Sampling is adaptive: per-label, per-dimension coverage counts are tracked, and under-represented values are weighted more heavily via inverse-frequency sampling. Label allocation across a batch is similarly corrected toward the target distribution each round. A sliding window tracks the most recently generated attribute combinations per label; combinations seen within the window are skipped with high probability to force exploration of fresh combinations.

Generator. An LLM (temperature 1.0) is prompted with the label definition, positive and negative boundary examples, the planner’s sampled attributes, and a set of previously generated examples the model is instructed not to repeat. Two distinct mechanisms feed this prompt: (i) the *boundary-example pool*, drawn from the label’s positive and negative examples to illustrate the decision boundary, and (ii) the *anti-repeat reference set*, sampled from previously accepted same-label output. Section 4.3 concerns the former and Section 4.4 the latter. Plans sharing an identical label–attribute pair may be batched into one call, with at most 64 samples requested per call.

Verifier. A separate LLM call (temperature 0.0) independently predicts a label and confidence for each candidate. A candidate passes verification only when the predicted label matches the target and confidence exceeds $\tau_v = 0.92$. This separates a high-temperature generator optimized for diversity from a verifier optimized for correctness.

Deduplication. We encode verified candidates with `multilingual-e5-large` and search previously accepted samples using a FAISS `IndexFlatL2` index. Candidates whose cosine similarity to an indexed sample is at least $\tau_s = 0.92$

are discarded as duplicates; the remaining candidates are added to the dataset and index.

4 Diagnosed Failure Modes

We describe four failure modes found in this pipeline. For each, we state the architectural precondition, observed manifestation, and general fix. The individual bugs resemble known software-engineering patterns; our contribution is showing how they silently disable feedback in a closed-loop agentic pipeline.

4.1 Stale-State Windowing

Precondition. A bounded-recency mechanism (e.g. "avoid repeating anything seen in the last W generations") implements eviction by comparing a per-item count against a *cumulative, monotonically increasing* total, rather than by tracking insertion order directly. This is a variant of a classic unbounded-counter eviction bug: a bounded structure whose eviction rule is driven by an unboundedly growing reference quantity rather than by insertion order (as in a standard LRU or fixed-capacity queue).

Manifestation. Our planner’s recency window was intended to retain roughly the last $W=100$ attribute combinations generated per label. The eviction rule computed, on every update, a cutoff of (cumulative total generated for that label) $-W$, and evicted any combination whose count fell at or below that cutoff. Once the cumulative total exceeded roughly $2W$, this cutoff exceeded the count of nearly every tracked combination, since per-combination counts are small (typically 1–20) while the cumulative total grows without bound. The window was then emptied on almost every update, silently disabling the anti-repeat check without raising an error. The only visible symptom was a gradual increase in duplicate rate.

Fix. Track eviction by insertion order (an explicit `deque` holding at most the last W combinations, evicting the oldest on overflow) rather than by comparing a local count to a global running total. This makes window size invariant to how long the pipeline has been running.

4.2 Batch-Blind Coordination

Precondition. A duplicate/coordination check scores a batch of candidates against shared state (e.g. an index of previously accepted items) *before committing any batch member to that state*, when

candidates are produced in parallel. This is structurally a time-of-check-to-time-of-use (TOCTOU) race: two operations independently check a shared resource against a stale snapshot, then both commit as if their check remained valid.

Manifestation. Our deduplication stage originally scored every candidate in a batch against the FAISS index of previously accepted samples, then added all survivors only after the batch had been scored. Two near-duplicates within the same batch were therefore invisible to each other because neither was indexed when the other was checked.

Fix. Score candidates sequentially within a batch, committing each accepted candidate to the shared index immediately before scoring the next. In practice we implement this as a vectorized two-pass computation rather than a literal sequential loop: Pass 1 checks all candidates against the pre-batch index snapshot; Pass 2 computes a single pairwise-similarity matrix among Pass-1 survivors and discards any candidate that is too similar to an *earlier-indexed* survivor within the batch. Because cosine similarity is symmetric and only insertion order (not wall-clock timing) determines the correct outcome, this is equivalent to true sequential commit-then-check at a fraction of the cost.

4.3 Anchor Staleness

Precondition. A few-shot or boundary-exemplar mechanism selects deterministically from a small, static pool, returning the same exemplars regardless of how much generation has occurred. This resembles cache staleness because the selection is never refreshed as the surrounding context changes.

Manifestation. Our generator’s positive and negative boundary examples were the first three entries of a static per-label pool containing only five or six examples. Across potentially thousands of calls, the model therefore saw the same three anchors. In this narrow domain, attribute variation was layered onto similar sentence structures rather than producing genuinely new phrasing.

Fix. Draw exemplars from a rotating source: randomize selection from the static pool as an immediate mitigation, and bootstrap the exemplar pool from a recency-windowed sample (last 200 accepted samples per label) of the pipeline’s own recently accepted output, rather than the static pool alone. Sampling from only the most recent accepted output, rather than the full accumulated dataset, avoids a secondary feedback loop

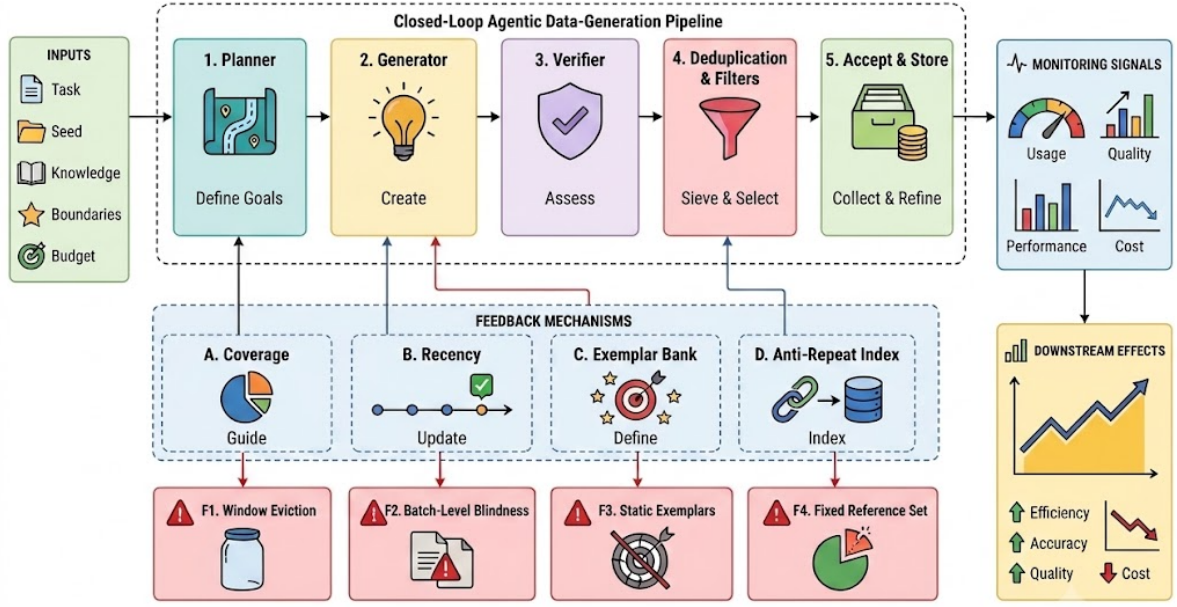


Figure 1: Closed-loop agentic data-generation pipeline.

in which the model increasingly imitates its own early-generation artifacts.

4.4 Reference-Signal Sparsity at Scale

Precondition. An anti-repetition reference set has a fixed size independent of the accumulated dataset. The constant-size sample becomes increasingly unrepresentative as the population grows.

Manifestation. The generator prompt always included three previously accepted same-label examples as a "do not repeat" signal. Early in a run, these represent a substantial fraction of the few dozen accepted samples. As the dataset grows, the same three-example reference set provides increasingly limited coverage of prior outputs, weakening the anti-repeat signal.

Fix. Scale the number of reference examples with accumulated dataset size (in our implementation, $\max(3, \lfloor n_{\text{label}}/20 \rfloor)$, capped at 15 for prompt-length reasons, where n_{label} is the number of accepted same-label samples so far). A principled alternative, which we leave to future work, is to derive the required reference-set size from a coverage argument: if a candidate’s near-duplicate lies among its k nearest neighbors with fixed probability, a fixed detection probability requires reference-set size to grow proportionally with n_{label} , which

would argue for linear rather than sublinear scaling; validating this requires assumptions about local duplicate clustering in embedding space that we have not yet tested against our own data. A second alternative is to replace random reference selection with nearest-neighbor retrieval against the same embedding index already maintained for deduplication, surfacing the examples the candidate is most at risk of duplicating rather than an arbitrary sample.

5 Early Detection via Internal-State Monitoring

Sections 4.1–4.4 diagnosed four feedback mechanisms whose failures remain hidden in aggregate metrics until their effects compound. We therefore monitor mechanism state directly to detect degradation before it becomes visible in duplicate or acceptance rates.

5.1 Feedback Health Signals

For a subset of the mechanisms diagnosed above, we define a scalar *health signal* $h_m(t) \in [0, 1]$, computed at each batch t , reflecting whether the mechanism is still performing its intended function:

- **Window occupancy** $h_{\text{window}}(t)$: the fraction of the recency window’s capacity actually

holding tracked combinations, per label, at batch t . Under correct operation this stabilizes near 1 once the window fills; under the stale-state windowing failure (Section 4.1), it collapses toward 0 as the cumulative total grows.

- **Reference coverage** $h_{\text{ref}}(t)$: the number of references actually supplied divided by the number required by the configured scaling rule, clipped to $[0, 1]$. Under correct operation this remains near 1; under the fixed-size failure (Section 4.4), it declines once the required count exceeds three.

Both signals are byproducts of the instrumentation already required to implement the corresponding fixes; monitoring them requires no additional data collection beyond what a correctly instrumented pipeline already logs.

5.2 Change-Point Detection

We apply a one-sided CUSUM statistic to each health signal to detect sustained downward drift independently of output metrics. After a warm-up period, we estimate the healthy reference mean μ_m and standard deviation σ_m from calibration batches that precede any injected fault:

$$S_m(t) = \max\{0, S_m(t-1) + (\mu_m - h_m(t)) - k\sigma_m\}, \quad (1)$$

$$S_m(0) = 0.$$

The slack parameter k and decision threshold H are held fixed across conditions and tuned on a held-out seed. A detection fires at the first batch $t_1^{(m)}$ for which $S_m(t) > H$. CUSUM itself is standard; our contribution is applying it to internal feedback state rather than aggregate output metrics.

5.3 Lead Time

For each mechanism m , we define a harm-oriented output metric $q_m(t)$, either duplicate rate $d(t)$ or rejection rate $1 - a(t)$, so that larger values always indicate worse behavior. Given a pre-registered alert threshold τ_m , the output alarm occurs at $t_2^{(m)} = \min\{t : q_m(t) \geq \tau_m\}$. We define *lead time* as

$$L_m = t_2^{(m)} - t_1^{(m)}, \quad (2)$$

measured in batches. A positive L_m means the internal-state monitor fires before the associated output metric reaches its alert threshold.

5.4 Evaluation Protocol

We compute L_m under each corresponding buggy ablation across scale-sweep runs and seeds, reporting the median and interquartile range (IQR). We also report the false-positive (FP) rate under the all-fixed condition. A useful leading indicator must provide positive lead time without firing frequently during normal operation.

6 Experiments

6.1 Research Questions

The evaluation is organized around four research questions. **RQ1 (aggregate effect)**: Does fixing all four failure modes improve duplicate rate, acceptance rate, and cost per accepted sample at production scale? **RQ2 (attribution)**: How much does fixing each mechanism contribute when the other three remain in their original state? **RQ3 (scale dependence)**: Does the cost of feedback blindness grow with accumulated dataset size, as predicted in Section 4? **RQ4 (downstream impact)**: Does fixing the generation pipeline improve a classifier trained on its output?

6.2 Task and Controlled Conditions

We evaluate on the sensitive-information-request classification task described in Section 3: four labels (otp, password, pin, normal), nine attribute dimensions, and Vietnamese-language output. Across conditions, we fixed the task configuration, label targets, prompts except for the mechanism under test, generator and verifier model versions, decoding parameters, confidence and similarity thresholds, concurrency, and target dataset size.

Settings. We use multilingual-e5-large embeddings, a FAISS IndexFlatL2 index, and verifier confidence threshold $\tau_v = 0.92$. The generator requests at most 64 samples per call. Generation and verification each allow up to eight concurrent calls. Table 1 defines six conditions. Each single-fix condition changes one mechanism relative to the all-buggy baseline, estimating its benefit while the other three remain unchanged. The all-fixed condition measures their combined effect.

6.3 Aggregate Effect and Attribution

For RQ1–RQ2, duplicate rate is the fraction of candidates reaching deduplication rejected at τ_s ; acceptance rate is the fraction of generated candidates ultimately committed. LLM calls per sample

Condition	Window eviction	Deduplication	Exemplars	References
All-buggy (baseline)	Cumulative cutoff	Batch-blind	Static first three	Fixed at three
Fix window only	Insertion order	Batch-blind	Static first three	Fixed at three
Fix dedup only	Cumulative cutoff	Sequential	Static first three	Fixed at three
Fix exemplars only	Cumulative cutoff	Batch-blind	Rotating	Fixed at three
Fix references only	Cumulative cutoff	Batch-blind	Static first three	Scaled
All-fixed	Insertion order	Sequential	Rotating	Scaled

Table 1: Experimental design matrix. Each single-fix condition changes one mechanism relative to the all-buggy baseline; the all-fixed condition changes all four.

divides generator and verifier calls by committed samples. Semantic diversity is one minus mean pairwise cosine similarity among accepted-sample embeddings. Table 2 reports mean $\pm$ standard deviation over three seeds.

At $N=4000$, all fixes together reduce duplicate rate by 19.03%, raise acceptance by 23.07%, reduce calls per sample from 2.94 to 2.18, and increase diversity from 0.711 to 0.792. Every single fix improves the reported means. Among single fixes, scaled references give the largest gains in duplication and acceptance, within-batch deduplication most reduces calls, and rotating exemplars most improves diversity.

6.4 Scale Dependence

To test RQ3, Table 3 compares all-buggy and all-fixed runs at $N \in \{500, 1000, 2000, 4000\}$ accepted samples. Checkpoint rates come from per-batch analytics snapshots with a fixed label distribution.

From $N=500$ to 4000, the duplicate-rate gap grows from 2.58% to 19.03% and the acceptance gap from 1.60% to 23.07%. These checkpoint comparisons support scale-dependent degradation over the evaluated range, addressing RQ3.

6.5 Internal Diagnostic Signals

Each batch report records window occupancy, unique exemplar count, and reference coverage. CUSUM evaluates window occupancy and reference coverage; Table 4 reports lead time and false-positive rates.

Window occupancy and reference coverage precede their output alarms by median 41 and 76 batches, respectively, with false-positive rates of 1.7% and 2.3%. These lead times offer an opportunity to intervene before output metrics reach their alert thresholds.

6.6 Downstream Classifier Validation

For RQ4, we train the same classifier on matched-size all-buggy and all-fixed datasets, then evaluate both versions on a shared human-labeled test set. Table 5 reports accuracy and macro-F1.

For RQ4, mean accuracy is 87.2% with all-fixed training data and 84.2% with all-buggy data; macro-F1 is 86.9% and 83.7%, respectively. These are descriptive comparisons; no statistical significance is claimed.

6.7 Human Audit of Duplicate Decisions

To validate τ_s , human annotators judge 100 pairs at or above 0.92 and 100 immediately below it. Table 6 reports duplicate judgments by similarity stratum.

The audit finds duplicates in 87% of sampled pairs at or above $\tau_s=0.92$ versus 19% immediately below it, supporting the threshold as a useful but imperfect separator. The below-threshold sample does not establish corpus-wide recall.

7 Discussion

The four failure modes in Section 4 share a common shape: their correctness is not directly observable from aggregate output metrics. Duplicate and acceptance rates are downstream consequences of several mechanisms, including window eviction, batch commit ordering, exemplar selection, and reference sizing. Any one can silently fail while the others continue to function. Pipelines of this shape should therefore log mechanism state, such as window occupancy, exemplar-pool diversity, and reference coverage, rather than waiting for aggregate metrics to reveal a costly problem.

These findings also motivate lightweight human review. A reviewer spot-checking output may notice repetitive patterns before an aggregate duplicate-rate metric moves. Such audits can complement automated monitoring by detecting fail-

Condition	Duplicate rate (%) ↓	Acceptance rate (%) ↑	LLM calls/sample ↓	Semantic diversity ↑
All-buggy (baseline)	61.72 ± 3.69	30.25 ± 2.1	2.94 ± 0.12	0.711 ± 0.008
Fix window only	56.17 ± 2.44	35.74 ± 1.8	2.72 ± 0.10	0.728 ± 0.009
Fix dedup only	55.83 ± 4.71	33.82 ± 1.6	2.54 ± 0.08	0.720 ± 0.007
Fix exemplars only	54.25 ± 2.93	34.57 ± 1.7	2.61 ± 0.09	0.752 ± 0.010
Fix references only	52.09 ± 3.31	37.88 ± 1.5	2.65 ± 0.07	0.741 ± 0.008
All-fixed	42.69 ± 2.61	53.32 ± 2.26	2.18 ± 0.05	0.792 ± 0.006

Table 2: Ablation results at $N=4000$, reported as mean $\pm$ standard deviation over three seeds. Bold indicates the best value in each metric column.

Accepted samples N	Duplicate rate (%) ↓		Acceptance rate (%) ↑	
	All-buggy	All-fixed	All-buggy	All-fixed
500	31.42	28.84	68.12	69.72
1000	42.66	34.16	55.82	63.27
2000	55.31	39.77	39.61	60.45
4000	61.72	42.69	30.25	53.32

Table 3: Scale-dependence checkpoints for the all-buggy and all-fixed pipelines. Bold indicates the better value at each checkpoint.

Signal	Med. L_m ↑	IQR ↓	FP ↓
Window occupancy	41	8	1.7
Reference coverage	76	14	2.3

Table 4: CUSUM detection results. Lead time and IQR are measured in batches; FP is the percentage of all-fixed runs that trigger a false alarm. Bold indicates the best value in each column.

Similarity stratum	n	Human dup. (%)
At/above 0.92	100	87.0
Immediately below 0.92	100	19.0

Table 6: Human validation of embedding-based duplicate judgments above and immediately below the cosine-similarity threshold.

Training data	Accuracy (%)	Macro-F1 (%)
All-buggy pipeline	84.2 ± 0.9	83.7 ± 1.1
All-fixed pipeline	87.2 ± 0.7	86.9 ± 0.8

Table 5: Downstream classifier performance on the shared human-labeled test set, reported as mean $\pm$ standard deviation over matched seeds. Bold indicates the better result.

ures that degrade a mechanism’s function without interrupting its execution.

8 Conclusion

We introduced *feedback blindness*: the failure of an internal feedback mechanism while every pipeline stage continues to execute and return valid output. In a production synthetic-data pipeline, we diagnosed four instances involving recency-window eviction, within-batch deduplication, boundary-exemplar selection, and anti-repeat reference sizing, and described architecture-level fixes for each. Successful component execution and aggregate metrics are insufficient evidence that a closed loop remains healthy. Monitoring mechanism state, such

as window occupancy and reference coverage, provides a more direct basis for diagnosis and early warning. Although our case study is task-specific, its architectural preconditions can be checked in other agentic data-generation pipelines.

Limitations

This study covers one pipeline, one classification task, and one language (Vietnamese). Experiments extend to $N=4000$ accepted samples, so behavior at larger scales remains unknown. Downstream validation uses one shared human-labeled test set, and the duplicate audit covers 200 pairs concentrated near the 0.92 similarity threshold. Our generalizability claim is architectural: each failure depends on a precondition that is not specific to this code-base. However, we have not validated the taxonomy in an independently implemented pipeline, another language, or a substantially different task.

Ethical Considerations

All utterances used in this study are synthetic. The pipeline is not conditioned on real customer transcripts, and the hand-authored seed examples con-

tain no personally identifiable information or customer credentials. Human annotators therefore review only synthetic text. Because the data concern requests for OTP codes, passwords, and PINs, they could be misused to imitate social-engineering language. We use them only to train defensive compliance classifiers. Any release should document the data’s synthetic provenance, intended use, limitations, and misuse risks, and omit operational details that could facilitate credential theft or impersonation.

References

- Tyler Akidau, Alex Balikov, Kaya Bekiroglu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. 2013. MillWheel: Fault-tolerant stream processing at internet scale. *Proceedings of the VLDB Endowment*, 6(11):734–746.
- Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022. [Deduplicating training data makes language models better](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8424–8445, Dublin, Ireland. Association for Computational Linguistics.
- Lan Li, Liri Fang, Bertram Ludäscher, and Vette I Torvik. 2025. [AutoDCWorkflow: LLM-based data cleaning workflow auto-generation and benchmark](#). In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 7766–7780, Suzhou, China. Association for Computational Linguistics.
- Lin Long, Rui Wang, Ruixuan Xiao, Junbo Zhao, Xiao Ding, Gang Chen, and Haobo Wang. 2024. [On LLMs-driven synthetic data generation, curation, and evaluation: A survey](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11065–11082, Bangkok, Thailand. Association for Computational Linguistics.
- Jon Saad-Falcon, Estefany Kelly Buchanan, Mayee Chen, Tzu-Heng Huang, Brendan McLaughlin, Tanvir Bhathal, Shang Zhu, Ben Athiwaratkun, Fred-eric Sala, Scott Linderman, Azalia Mirhoseini, and Christopher Ré. 2025. [Weaver: Shrinking the generation-verification gap by scaling compute for verification](#). In *Advances in Neural Information Processing Systems*, volume 38.
- Trausti Saemundsson, Hjortur Bjornsson, Gregory Chockler, and Ymir Vigfusson. 2014. [Dynamic performance profiling of cloud caches](#). In *Proceedings of the ACM Symposium on Cloud Computing*. Association for Computing Machinery.
- Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Kumar Paritosh, and Lora M. Aroyo. 2021. [“Everyone Wants to Do the Model Work, Not the Data Work”: Data cascades in high-stakes AI](#). In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery.
- D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems*, volume 28, pages 2503–2511. Curran Associates, Inc.
- Saptarshi Sengupta, Harsh Vashistha, Kristal Curtis, Akshay Mallipeddi, Abhinav Mathur, Joseph Ross, and Liang Gou. 2025. [MAG-V: A multi-agent framework for synthetic data generation and verification](#). *Preprint*, arXiv:2412.04494.
- Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. 2024. [AI models collapse when trained on recursively generated data](#). *Nature*, 631:755–759.
- Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. [Self-instruct: Aligning language models with self-generated instructions](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.
- Miriam Wanner, Benjamin Van Durme, and Mark Dredze. 2025. [DnDScore: Decontextualization and decomposition for factuality verification in long-form text generation](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 23609–23626, Suzhou, China. Association for Computational Linguistics.
- Kangning Zhang, Weiwen Liu, Wenxiang Jiao, Kounianhua Du, Yuan Lu, Weinan Zhang, and Yong Yu. 2026. [LoopTool: Closing the data-training loop for robust LLM tool calls](#). In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 21144–21164, San Diego, California, United States. Association for Computational Linguistics.