Supplementary Material for: A Candidate Pattern Language for Resilient SME Data Pipelines: Design and Failure-Injection Evaluation

Rohit Arora[1*]

[1*]Independent Researcher, New York, NY, USA.

Corresponding author(s). E-mail(s): rohitarora.86@gmail.com;

This file accompanies the main manuscript and is not intended to be read independently of it. Section and table numbers here are prefixed with "S" and are distinct from the main manuscript's numbering. Cross-references from the main text to this file (e.g., "Supplementary Table S1") are plain-text pointers, since the two documents compile separately.

S1. Derivation Record for Candidate Mechanisms

This table restates the pattern-derivation process described narratively in the main manuscript's Research Methodology and Pattern Derivation section as an explicit record: for each candidate mechanism, the literature supporting the failure mode it addresses, the SME-suitability criterion applied to it, the resulting decision, and the specific reason for that decision. This is a restructuring of candidates already discussed narratively in the main manuscript, not the product of additional research beyond what that discussion describes.

Table S1: Derivation record for each candidate mechanism considered during pattern development.

Candidate mechanism	Sources	SME suitability	Decision	Reason
Incremental Change Capture (watermark + atomic checkpointing)	(Akidau et al. 2015, 2018; Kreps 2013; Lamport 1978; Chandy and Lamport 1985)	App-level watermark column; no managed CDC service needed	Retained	Addresses recurring full-re-extraction cost and staleness
Idempotent Replay (keyed upsert / processing ledger)	(Helland 2012; Gray and Reuter 1993; Vogels 2009; Huang and Garcia-Molina 2001)	Keyed upsert; no distributed transaction coordinator needed	Retained	Addresses duplicate rows under at-least-once delivery, the practical norm absent exactly-once messaging infrastructure
Dead-Letter Quarantine	(Hohpe and Woolf 2003; Killalea 2016; Nygard 2018; Newman 2015)	Single quarantine table with a retention policy; no dedicated triage team needed	Retained	Addresses the batch-halt-vs.-silent-drop trade-off on malformed records
Schema Drift Adapter	(Santucci 1998; Curino et al. 2013; Stonebraker and Ilyas 2018)	Static field-map layer maintainable by one developer; no schema-registry service needed	Retained	Addresses upstream schema changes breaking rigid field mappings
Backpressure-Aware Batching (AIMD)	(Jacobson 1988; Welsh et al. 2001; Reactive Streams Specification 2015; Bóner et al. 2014)	In-process batch-size controller; no dedicated load-balancing infrastructure needed	Retained	Addresses destination overload under a fixed batch size
Pipeline Health Heartbeat	(Beyer et al. 2016; Truong and Nguyen 2024)	One shared heartbeat table plus a lightweight polling watchdog; no full APM platform needed	Retained	Addresses silent stage stalls invisible to a bare liveness check

(Table S1 continued)

Candidate mechanism	Sources	SME suitability	Decision	Reason
End-to-End Reconciliation	(Redman 2001)	Scheduled aggregate queries; no dedicated audit team needed	Retained	Added specifically to close the "alive but wrong" gap the other six, including Heartbeat, cannot detect
Dedicated Change-Data-Capture Cluster	(Kreps 2013; Akidau et al. 2018)	Requires a managed streaming cluster	Rejected	Disproportionate to the SME-suitability criterion; simpler watermark mechanism retained instead
Watermark Tracking / Delete Detection (drafted separately)	(Akidau et al. 2015)	Satisfies the SME-suitability criterion only once combined	Merged into Incremental Change Capture	Forces and Solution fields proved inseparable in practice once drafted concretely

S2. Extended Baseline Comparison Details

This section expands two baseline comparisons in the main manuscript's Baseline Comparisons subsection with the full numeric breakdown and derivation reasoning that the main text states only as a headline result.

S2.1 Backpressure-Aware Batching (AIMD) vs. fixed batching

The main text reports a roughly 23× difference between fixed batching's and AIMD's median per-batch latency during a simulated 8× destination slowdown (0.484 s ± 0.001 s vs. 0.021 s ± 0.001 s), and notes that a light-touch sweep across 2×, 4×, and 8× slowdown magnitudes confirms the direction of this trade-off holds throughout. The full per-magnitude breakdown is:

- **2× slowdown:** median latency 0.126 s (fixed) vs. 0.005 s (AIMD); throughput 1,254 vs. 1,030 records/s.
- **4× slowdown:** median latency 0.246 s (fixed) vs. 0.010 s (AIMD); throughput 643 vs. 543 records/s.
- **8× slowdown:** median latency 0.486 s (fixed) vs. 0.019 s (AIMD); throughput 324 vs. 279 records/s.

The latency ratio itself stays within a roughly 24–26× band across all three magnitudes. Because this behavior follows from the linear synthetic latency model, it should not be interpreted as independent robustness evidence. In this simplified model, per-record write latency is modeled as directly proportional to batch size (a strictly linear function), so aggregate throughput over a fixed record count is close to invariant to batch-size policy, and the median-latency ratio follows almost directly from the ratio between the AIMD floor (5) and the fixed batch size (150) – $150/5 = 30$, close to

the observed 24–26× band, with the remaining difference attributable to the additive-increase ramp-up period before AIMD reaches its floor. The practical difference AIMD provides in this model is therefore a bounded worst-case blocking call after the first reactive step, not raw throughput – fixed batching's raw throughput is higher than AIMD's at every magnitude tested, because fixed batching's larger batches amortize the same total record count into fewer, larger write calls. Reading the 24–26× figure as a demonstration that the control loop behaves as designed under this specific injected model – rather than as an external validation of a real-world latency improvement – is the correct interpretation the main text argues for; a nonlinear destination-latency model (incorporating queue depth and record-level percentile latency rather than only per-batch latency) is named as future work in the main manuscript's Threats to Validity and Limitations section precisely because this linear-model result cannot, on its own, distinguish a genuine resilience benefit from an artifact of the injected latency function.

S2.2 Dead-Letter Quarantine vs. two naive alternatives

The main text reports that a competent (non-negative-control) baseline for Dead-Letter Quarantine – validating each record independently, committing valid records immediately, and discarding invalid ones with a logged reason but no retention or replay path – committed all 450 valid records immediately in the malformed-record scenario (unlike an all-or-nothing negative-control loader, which committed zero), while still underperforming Dead-Letter Quarantine on recoverability (0% vs. 100% of the 50 malformed records ever recoverable).

This 0%-vs-100% gap is a direct, mechanical consequence of what each design retains, not an incidental measurement: Dead-Letter Quarantine writes the full original payload of a rejected record into a persistent quarantine table before discarding it from the main processing path, so a later correction (in this scenario, a warehouse supervisor supplying the missing quantity) can be applied to that retained payload and replayed through the same idempotent-upsert destination write. The competent baseline's discard-and-log design keeps only the *reason* a record was rejected (in this experiment, a string describing the missing required field), not the payload itself, so there is nothing to correct and replay; recovering an equivalent record would require re-extracting it from the original source system from scratch, which may or may not still be possible depending on the source system's own retention and change-tracking behavior, a question this prototype's synthetic, in-memory source system cannot answer either way. Both designs are auditable in the sense that both log a machine-readable reason for rejection, so "auditability" alone does not distinguish them; what distinguishes them is specifically whether that log is attached to enough retained state to act on. The prototype does not measure operator effort (the human time to review, correct, and replay a batch of quarantined records) directly; the 0%-vs-100% recovery-rate gap is offered as the mechanical precondition for any recovery effort to be possible at all, not as a measurement of how much effort that recovery would take in practice.

References

Akidau T, Bradshaw R, Chambers C, et al (2015) The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. Proceedings of the VLDB Endowment 8(12):1792–1803

Akidau T, Chernyak S, Lax R (2018) Streaming Systems: The What, Where, When, and How of Large-Scale Data Processing. O'Reilly Media

Beyer B, Jones C, Petoff J, et al (eds) (2016) Site Reliability Engineering: How Google Runs Production Systems. O'Reilly Media

Bóner J, Farley D, Kuhn R, et al (2014) The reactive manifesto, version 2.0. https://www.reactivemanifesto.org, accessed August 2026

Chandy KM, Lamport L (1985) Distributed snapshots: Determining global states of distributed systems. ACM Transactions on Computer Systems 3(1):63–75. https://doi.org/10.1145/214451.214456

Curino C, Moon HJ, Deutsch A, et al (2013) Automating the database schema evolution process. The VLDB Journal 22(1):73–98. https://doi.org/10.1007/s00778-012-0302-x

Gray J, Reuter A (1993) Transaction Processing: Concepts and Techniques. Morgan Kaufmann

Helland P (2012) Idempotence is not a medical condition. Communications of the ACM 55(5):56–65. https://doi.org/10.1145/2160718.2160734

Hohpe G, Woolf B (2003) Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley

Huang Y, Garcia-Molina H (2001) Exactly-once semantics in a replicated messaging system. In: Proceedings of the 17th International Conference on Data Engineering (ICDE), pp 3–12, https://doi.org/10.1109/ICDE.2001.914808

Jacobson V (1988) Congestion avoidance and control. ACM SIGCOMM Computer Communication Review 18(4):314–329. https://doi.org/10.1145/52325.52356

Killalea T (2016) The hidden dividends of microservices. ACM Queue 14(3):25–34. https://doi.org/10.1145/2956641.2956643

Kreps J (2013) The log: What every software engineer should know about real-time data's unifying abstraction. LinkedIn Engineering Blog, https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

Lamport L (1978) Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21(7):558–565. https://doi.org/10.1145/359545.359563

Newman S (2015) Building Microservices: Designing Fine-Grained Systems. O'Reilly Media

Nygard MT (2018) Release It!: Design and Deploy Production-Ready Software, 2nd edn. Pragmatic Bookshelf

Reactive Streams Specification (2015) Reactive streams specification, version 1.0.0. https://www.reactive-streams.org, accessed August 2026

Redman TC (2001) Data Quality: The Field Guide. Digital Press

Santucci G (1998) Semantic schema refinements for multilevel schema integration. Data & Knowledge Engineering 25(3):301–326. https://doi.org/10.1016/S0169-023X(97)00024-4

Stonebraker M, Ilyas IF (2018) Data integration: The current status and the way forward. IEEE Data Engineering Bulletin 41(2):3–9

Truong HL, Nguyen NNT (2024) TENSAI: Practical and responsible observability for data quality-aware large-scale analytics. Journal of Data and Information Quality 16(4):1–43. https://doi.org/10.1145/3708014

Vogels W (2009) Eventually consistent. Communications of the ACM 52(1):40–44. https://doi.org/10.1145/1435417.1435432

Welsh M, Culler D, Brewer E (2001) SEDA: An architecture for well-conditioned, scalable internet services. ACM SIGOPS Operating Systems Review 35(5):230–243. https://doi.org/10.1145/502059.502057