



(RESEARCH ARTICLE)



## RELIABILITY ENGINEERING FOR MODERN DATA AND AI SYSTEMS

Santosh Kumar Maddali \*

Sr. Staff Data Engineer.

\* Corresponding Author

World Journal of Advanced Research and Reviews, 2026, 30(01), 2664–2672

Publication history: Received on 21 February 2026; revised on 30 March 2026; accepted on 02 April 2026

Article DOI: <https://doi.org/10.30574/wjarr.2026.30.1.0853>

Copyright © 2026 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution License 4.0

### ABSTRACT

Data platforms and machine learning systems now sit on the critical path of decision making in most large organizations, yet the discipline used to keep them dependable lags far behind the discipline used for conventional software services. Site reliability engineering (SRE) gives service teams a shared vocabulary of service level indicators, service level objectives, error budgets, and blameless postmortems. Data and AI systems, however, fail in ways that those tools were not designed to catch: a pipeline can succeed while producing wrong numbers, and a model can serve predictions with low latency while its accuracy silently decays. This paper adapts reliability engineering to data and AI systems. It proposes a failure taxonomy that separates availability failures from correctness, freshness, and behavioral failures; defines a catalogue of data and model service level indicators; and describes an operating model in which data contracts, automated validation, drift monitoring, staged rollout, and incident practice are combined into a single reliability loop. The contribution is conceptual and practice oriented: it consolidates published research and reported practitioner experience into a structure that engineering teams can adopt incrementally without first building a large platform. The paper closes with limitations and with the measurement work that remains open.

**Keywords:** Data Contracts, Data Observability, Data Quality, Error Budgets, Machine Learning Operations, Model Monitoring, Service Level Objectives, Site Reliability Engineering

### 1 INTRODUCTION

The reliability of a modern enterprise increasingly depends on systems that move, transform, and learn from data. Regulatory reporting, clinical scheduling, inventory replenishment, and customer facing recommendations all run on pipelines and models rather than on hand written application logic. When these systems fail, the failure is often invisible: dashboards keep rendering, jobs report success, and prediction endpoints keep answering. The damage appears later as a wrong report, a mispriced order, or a model that has quietly become biased toward stale patterns.

Conventional reliability engineering was built for request and response services. The SRE literature from Google established a practical framework centered on service level indicators (SLIs), service level objectives (SLOs), error budgets, and structured incident response [1], [2]. These ideas transfer to data and AI systems only partially. An HTTP service is either up or down, fast or slow; a data pipeline can be up, fast, and wrong. Sculley et al. observed that machine learning systems accumulate a distinctive form of technical debt because their behavior is determined by data as much as by code [3]. Breck et al. later argued that production readiness for ML depends on tests and monitoring across data,

model, infrastructure, and operations, not on model accuracy alone [4]. Sambasivan et al. documented how data problems cascade downstream and are frequently discovered late, after they have compounded [5].

This paper addresses the gap between service reliability practice and the needs of data and AI systems. It makes three contributions. First, it proposes a failure taxonomy that distinguishes the failure modes that generic SRE tooling already catches from those that require data specific and model specific instrumentation. Second, it defines a catalogue of SLIs that make correctness, freshness, and model behavior measurable, so that SLOs and error budgets can be applied to them. Third, it describes an operating model, summarized in Figure 1, that connects design time controls (contracts and validation), deployment time controls (staged rollout and rollback), and run time controls (monitoring and incident response) into a single loop. The paper is deliberately vendor neutral and does not depend on any specific cloud platform, orchestration engine, or model registry.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the methodology and the reliability model. Section 4 covers reliability practices for data pipelines. Section 5 covers reliability practices for AI and ML systems. Section 6 describes the operating model. Section 7 discusses limitations and open questions. Section 8 concludes.

---

## 2 RELATED WORK

### 2.1 Site Reliability Engineering

The SRE books from Google define the vocabulary used throughout this paper. An SLI is a quantitative measure of some aspect of the service; an SLO is a target value or range for an SLI; and an error budget is the allowed amount of SLO violation over a window, which is used to balance reliability work against feature work [1]. The companion workbook adds practical guidance on choosing SLIs, running postmortems, and managing alerting on burn rate rather than on raw thresholds [2]. Nygard's treatment of stability patterns, including circuit breakers, bulkheads, and timeouts, describes design level defenses that remain relevant for pipeline components and model serving layers [6]. Basiri et al. formalized chaos engineering as the practice of running controlled experiments against production to build confidence in a system's ability to withstand turbulent conditions [7].

### 2.2 Data Quality and Data Management for Production Systems

Polyzotis et al. surveyed the data management challenges that arise when machine learning runs in production, including data understanding, validation, cleaning, and the management of training data over time [8]. Breck et al. described a data validation system that infers a schema from historical data and checks each new batch against it, flagging distribution skew between training and serving data [9]. Schelter et al. presented a declarative approach to data quality verification at scale, in which engineers express constraints on completeness, uniqueness, and value ranges and the system computes the corresponding metrics incrementally [10]. Kleppmann's treatment of data intensive applications provides the foundations for reasoning about consistency, replication, and stream processing guarantees that underlie pipeline correctness [11]. Akidau et al. formalized the separation of event time from processing time and the use of watermarks and triggers to reason about completeness in unbounded data, which is essential for defining freshness precisely [12].

### 2.3 Machine Learning in Production

Sculley et al. introduced the hidden technical debt framing, identifying boundary erosion, entanglement, undeclared consumers, data dependencies, and feedback loops as sources of long term maintenance cost [3]. Breck et al. proposed the ML Test Score, a rubric of tests and monitoring checks across data, model development, infrastructure, and monitoring, which many organizations use as a production readiness checklist [4]. Amershi et al. reported a case study of how software engineering practices had to adapt for ML at a large software company, noting that data discovery, management, and versioning are more difficult than for traditional code [13]. Paleyes et al. surveyed published case studies of ML deployment and organized the reported challenges by stage of the deployment workflow [14]. Shankar et al. conducted an interview study of ML engineers and found that their work follows a continual loop of data preparation, experimentation, evaluation across a multi-stage deployment, and ongoing monitoring and response [15]. Kreuzberger et al. consolidated the MLOps literature, tooling, and practitioner interviews into a definition and reference architecture built on automation, versioning, continuous monitoring, and reproducibility [16]. Zaharia et al. described MLflow as an example of platform support for tracking experiments, packaging models, and managing deployment [17]. On the behavioral side, Gama et al. and Lu et al. provide comprehensive surveys of concept drift and its detection, which underpin the drift monitoring practices discussed in Section 5 [18], [19].

## 2.4 Gap Addressed by This Paper

The literature above is rich but fragmented. Conventional SRE practice provides the operating loop and can accommodate arbitrary user visible SLIs, but it does not prescribe data specific semantic correctness or model behavior measurements. The data management and ML systems literature provides validation and monitoring techniques but rarely connects them to SLOs, error budgets, and incident response. This paper joins the two by placing data specific and model specific SLIs inside the standard SRE loop.

## 3 METHODOLOGY AND RELIABILITY MODEL

### 3.1 Method

The reliability model presented here was developed by synthesizing the published research summarized in Section 2 with the practitioner accounts reported in that literature. The literature base was assembled as a structured narrative review covering foundational SRE, distributed data processing, production data quality, MLOps, and model monitoring. Sources were selected for direct relevance to operational failure detection, reliability measurement, or production controls, with preference given to peer reviewed publications and established foundational works. The objective was representative coverage of the principal reliability mechanisms rather than exhaustive literature retrieval. The synthesis then followed three steps. First, reported failure modes from the surveyed literature were collected and grouped by the property of the system that was violated. Second, for each group, the measurement needed to detect the violation was identified and expressed as an SLI. Third, the SLIs were mapped onto the standard SRE operating loop to determine which existing practices apply unchanged, which require adaptation, and which are new. No proprietary data or confidential incident records are used in this paper; all claims are supported by the cited public sources or presented as design reasoning derived from those sources.

### 3.2 Failure Taxonomy

Table 1 presents the taxonomy. The four categories are ordered by how visible the failure is to conventional monitoring. Availability failures are loud and are caught by existing tooling. Freshness failures are partially visible, because a late job is usually noticed, but the downstream effect of stale data often is not. Correctness failures and behavioral failures are silent by default and require instrumentation that is specific to data and to models. Foidl et al. studied data pipeline quality through a literature review, expert interviews, and an analysis of developer discussions, and found that data-related issues cluster around ingestion, integration, and cleaning stages and are frequently caused by incorrect data types and compatibility problems between stages [20]; these are correctness failures in the sense used here. The central argument of this paper is that most reliability investment in data and AI systems should go toward the two silent categories, because that is where the detection gap is largest.

**Table 1: Failure taxonomy for data and AI systems**

| Category     | Description                                                             | Typical Cause                                                                             | Detected By                                                                     |
|--------------|-------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| Availability | The pipeline, store, or model endpoint is unreachable or returns errors | Infrastructure outage, quota exhaustion, dependency failure                               | Standard uptime and error rate monitoring                                       |
| Freshness    | Data or predictions are available but older than consumers expect       | Upstream delay, backfill contention, scheduler misconfiguration                           | Job completion time, watermark lag, last update timestamps                      |
| Correctness  | Data or predictions are available and timely but wrong                  | Schema change, semantic change in source, duplicate or dropped records, join key mismatch | Validation rules, reconciliation, contract checks, anomaly detection on metrics |
| Behavioral   | The model behaves differently from how it did when approved             | Concept drift, feature drift, training and serving skew, feedback loops                   | Drift statistics, shadow evaluation, delayed ground truth comparison            |

### 3.3 SLI Catalogue

Table 2 lists SLIs that make each failure category measurable. Each SLI is defined so that it can be expressed as a ratio of good events to total events over a window, which is a particularly convenient form for error budget accounting and the one recommended for SLO based alerting [1], [2]. The catalogue is intentionally small. In practice, a team should

select two to four SLIs per data product or model and resist the temptation to alert on every metric that is collected. The distinction between an SLI and a diagnostic metric matters: SLOs built on SLIs drive alerting and error budget policy, while diagnostic metrics support investigation after an SLI has moved.

**Table 2: Service level indicators for data and AI systems**

| Category     | SLI                                                                                | Definition                                                                                                                                                   |
|--------------|------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Availability | Pipeline success ratio                                                             | Scheduled runs that complete without error divided by scheduled runs, per window                                                                             |
| Availability | Serving availability                                                               | Prediction requests answered with a valid response divided by total requests                                                                                 |
| Freshness    | Freshness compliance                                                               | Consumer facing tables or feature views whose last update is within the agreed freshness bound, divided by total tables or views, sampled at fixed intervals |
| Freshness    | Event time lag                                                                     | Fraction of processing intervals in which the watermark lag stays below the agreed bound                                                                     |
| Correctness  | Validation pass ratio                                                              | Batches or micro batches that pass all contract and validation checks divided by total batches                                                               |
| Correctness  | Reconciliation match ratio                                                         | Control totals (row counts, sums, distinct keys) that match the source of truth within tolerance, divided by total controls                                  |
| Behavioral   | Feature stability (proxy SLI, used only while outcome ground truth is unavailable) | Monitored features whose distribution distance from the reference window stays below threshold, divided by monitored features                                |
| Behavioral   | Outcome quality                                                                    | Predictions whose delayed ground truth matches within the agreed tolerance, divided by predictions with available ground truth                               |

## 4 RELIABILITY PRACTICES FOR DATA PIPELINES

### 4.1 Data Contracts as the Reliability Boundary

A common root cause of silent correctness failures is an unannounced change at a producer boundary: a column is renamed, a unit changes from cents to dollars or from Celsius to Fahrenheit, or an enumeration gains a new value. Sculley et al. described unstable data dependencies as a primary source of ML technical debt [3], and the same mechanism applies to any downstream analytical table. A data contract makes the producer boundary explicit. At minimum it records the schema, the semantics of each field, the freshness commitment, the expected volume range, and the owner. Contracts are most effective when they are machine readable and enforced in the producer's deployment pipeline, so that a breaking change fails before it reaches consumers rather than after.

Schema evolution should be governed by compatibility rules. Additive changes (new optional columns) are compatible by default. Renames, type narrowing, and semantic changes are breaking and require either a versioned interface or a coordinated migration. Kleppmann discusses the compatibility properties of common serialization formats and the discipline required to evolve them safely [11]. The reliability benefit is that correctness failures are converted into deployment time failures, which are visible and cheap to fix.

### 4.2 Automated Validation at Every Boundary

Validation should run wherever data crosses a boundary: on ingestion, after each transformation stage, and before publication to consumers. The validation system described by Breck et al. infers expected constraints from history and raises anomalies when a new batch deviates [9]. The declarative approach of Schelter et al. lets engineers express constraints such as completeness, uniqueness, and value ranges and computes the metrics incrementally at scale [10]. Shankar et al. later described a production data validation approach at Meta in which each time partition of a dataset is summarized by per-feature quality metrics and compared against its history, so that a corrupted partition can be flagged before it is used for retraining [21]. In practice three layers of checks are useful. Structural checks confirm schema conformance and nullability. Statistical checks compare volume, distinct counts, and value distributions against a rolling baseline. Semantic checks encode business rules, for example that a shipment date is never earlier than its order date. Each layer catches a different class of defect, and none of them substitutes for the others.

The output of validation must be actionable. A failed check should either block publication, quarantine the offending records, or publish with a visible quality flag, according to a policy agreed with consumers. Silent logging of failures is the pattern that data cascades research warns against: problems are recorded but not acted on until they compound [5].

### 4.3 Freshness, Completeness, and Late Data

Freshness is easy to define loosely and hard to define precisely. The Dataflow model provides the vocabulary: event time is when something happened, processing time is when the system observed it, and a watermark is the system's estimate of how complete its view of event time is [12]. A freshness SLO should be stated in terms of event time, for example that ninety five percent of events are reflected in the consumer table within a fixed interval of their event time, measured continuously. Stating the objective in terms of job completion time alone hides the effect of upstream delay.

Late arriving data forces an explicit choice between latency and completeness. The stream processing literature recommends that pipelines document which choice they make, expose the choice to consumers, and make reprocessing idempotent so that late data can be incorporated without manual intervention [11], [12]. Idempotent writes and deterministic partitioning are the foundation for safe backfills and for recovery after an incident.

### 4.4 Reconciliation and Control Totals

Validation checks internal consistency; reconciliation checks agreement with an external source of truth. For operational and reporting data, control totals such as row counts, numeric sums, and distinct key counts should be computed at the source and compared at the destination. Reconciliation is one of the strongest checks for catching an entire dropped partition or a doubled load when every individual record is well formed, and it complements expected volume checks and completeness monitoring rather than replacing them. It is also the check that auditors, regulators, and business owners understand, which makes it a natural anchor for the correctness SLO.

### 4.5 Resilience Patterns in Pipeline Components

Pipelines are distributed systems, and the stability patterns from Nygard apply to them: timeouts on every external call, bounded retries with backoff, circuit breakers around flaky sources, and bulkheads that keep one failing feed from consuming the capacity needed by others [6]. Chaos engineering, in the sense of Basiri et al., can be applied cautiously to pipelines by injecting delayed partitions, malformed records, or unavailable dependencies in a staging environment to confirm that validation, alerting, and recovery behave as designed [7]. The goal is to find out how the system fails before a real incident does.

---

## 5 RELIABILITY PRACTICES FOR AI AND ML SYSTEMS

### 5.1 The Model as a Data Consumer and a Data Producer

A deployed model is simultaneously a consumer of feature data and a producer of predictions that other systems consume. All of Section 4 therefore applies to its inputs and outputs. The additional reliability concern specific to models is behavioral: the model can be available, fresh, and structurally correct while its predictions have become worse. Breck et al. emphasize that production ML reliability depends on data, model, infrastructure, and monitoring concerns beyond model accuracy alone [4], and Foidl et al. locate most reported data pipeline issues in ingestion, integration, and cleaning stages [20]; together these argue for solving pipeline reliability first and treating model specific monitoring as a layer on top.

### 5.2 Training and Serving Skew

Training and serving skew occurs when the features seen at serving time are computed differently from the features seen at training time. Common causes are separate code paths for batch and online feature computation, different handling of missing values, and time leakage in training data. The most effective defense is structural: compute features once through a shared path and log the exact feature vector that was served, so that it can be compared with the training distribution and reused for retraining. Where a shared path is not possible, skew detection compares the distribution of served features with the training reference on a schedule, as described in the data validation work of Breck et al. [9].

### 5.3 Drift Monitoring

Concept drift is a change in the relationship between inputs and the target; feature drift is a change in the input distribution alone. Gama et al. and Lu et al. survey the detection methods, which fall into distribution comparison methods, performance based methods that require ground truth, and sequential change detection methods [18], [19]. Reliability practice should treat drift statistics as diagnostic metrics and treat outcome quality against delayed ground truth as the SLI, because drift in a feature does not necessarily degrade predictions and a stable feature distribution

does not guarantee that the model remains useful. For this reason the feature stability entry in Table 2 is marked as a proxy SLI: it stands in for outcome quality only during the period before ground truth arrives, and it should be retired from SLO accounting once outcome quality can be measured. Where ground truth is delayed by days or weeks, an interim proxy such as agreement with a shadow model or a business metric with a known relationship to the prediction can be used, with the proxy validated periodically against real outcomes.

#### 5.4 Staged Rollout, Shadow Evaluation, and Rollback

Model changes should be deployed with the same discipline as service changes. Shadow evaluation runs the candidate model on live traffic without acting on its output and compares its predictions with the incumbent. Canary rollout exposes a small fraction of traffic to the candidate with automatic rollback if the serving availability or outcome quality SLI degrades. Every deployed model must be reproducible from a recorded combination of code version, training data snapshot, and configuration, which is the lineage that experiment tracking and model registry tooling is designed to capture [17]. Without that lineage, rollback restores a model artifact but not the ability to understand why it behaved as it did.

#### 5.5 Feedback Loops and Undeclared Consumers

Two of the debt categories identified by Sculley et al. deserve specific reliability treatment [3]. A feedback loop exists when a model's predictions influence the data on which it is later retrained, for example when a ranking model shapes which items receive clicks. Feedback loops can make a model appear to improve while narrowing its behavior. Mitigations include holding out a randomized slice of traffic as an unbiased reference and monitoring the diversity of served outputs. Undeclared consumers are systems that read a model's output without the model owner's knowledge; they turn any behavioral change into a potential incident for someone. Treating prediction outputs as a contracted data product, with the same ownership and change policy as a table, closes this gap.

#### 5.6 Readiness Checklists

The ML Test Score rubric provides a practical starting point for readiness review, covering data tests, model tests, infrastructure tests, and monitoring [4]. A reliability program should adopt such a rubric, tie the highest priority items to the SLIs in Table 2, and require that a model cannot be promoted to production until those items are satisfied. The rubric is more valuable as a shared standard across teams than as a score for any single model.

## 6 THE RELIABILITY OPERATING MODEL

Figure 1 shows how the controls described in Sections 4 and 5 fit together as a single loop. Design time controls (contracts and validation) sit on the left, deployment time controls (staged rollout and rollback) in the middle, and run time controls (SLIs, error budgets, and alerting) on the right. Incidents and postmortems close the loop by feeding new contract rules, validation checks, and SLI definitions back into the design time controls. The remainder of this section describes each part of the loop.

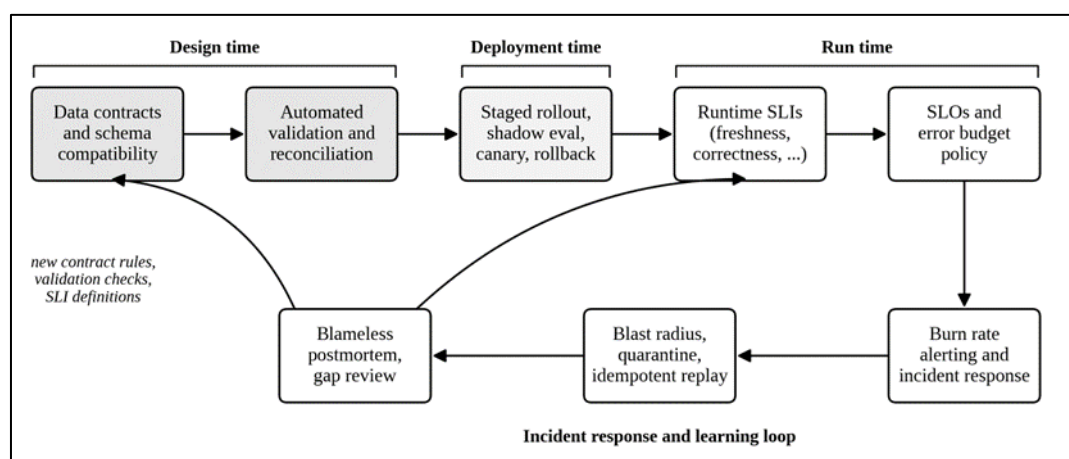


Figure 1: The reliability loop for data and AI systems

#### 6.1 SLOs and Error Budgets for Data Products

Each data product and each model should have a small set of SLOs drawn from Table 2 and agreed with its consumers. The agreement is the important part: an SLO that a consumer did not ask for and does not rely on is a source of noise. Error budgets then govern change velocity. When a product is inside its budget, teams may ship schema changes, new

features, and model retraining freely. When the budget is exhausted, changes are limited to reliability fixes until the budget recovers. This policy, adapted directly from SRE [1], gives data teams an objective way to negotiate between delivery pressure and stability.

## 6.2 Alerting on Symptoms, Not Causes

Alerts should page on SLI degradation, preferably using burn rate over multiple windows so that fast burns page quickly and slow burns are surfaced without paging [2]. Diagnostic metrics such as individual validation checks, drift statistics, and job durations should be visible on dashboards and attached to alerts as context, but should not page on their own. Data teams that page on every failed check quickly reach alert fatigue, after which real incidents are missed.

## 6.3 Incident Response for Silent Failures

Incident response for data and AI systems adds a data specific and temporal dimension to blast radius assessment. When a correctness failure is detected, responders must identify which partitions, which downstream tables, which reports, and which model training runs were affected and over what time range. Lineage metadata makes this tractable; without it, the assessment is manual and slow. Recovery then follows the standard incident structure described in the SRE literature [2], adapted to data: stop the spread by quarantining or flagging affected outputs, fix the root cause, reprocess affected partitions idempotently, and confirm through reconciliation that the corrected data matches the source of truth.

## 6.4 Postmortems and the Learning Loop

Blameless postmortems, as practiced in SRE, are the mechanism by which the taxonomy and the SLI catalogue improve over time [1]. Each postmortem should record which category in Table 1 the failure belonged to, whether an existing SLI detected it and how quickly, and which new check or contract rule would have caught it earlier. Over time, the pattern of postmortems shows where detection gaps remain and where investment should go. The interview study by Shankar et al. reports that experienced ML engineers treat monitoring and response as a continual stage of the workflow rather than a one time setup, and rely on this kind of iterative hardening rather than on any single upfront design [15].

## 6.5 Adoption Path

The model can be adopted incrementally. Drawing on the readiness rubric of Breck et al. [4] and the pipeline issue findings of Foidl et al. [20], a reasonable order is: first, publish contracts and freshness commitments for the most consumed data products; second, add structural and reconciliation checks at their publication boundary; third, define one correctness SLO and one freshness SLO per product and start recording an error budget; fourth, extend the same treatment to model inputs and outputs and add outcome quality monitoring; fifth, introduce staged rollout and shadow evaluation for model changes. Each step delivers value on its own, and no step requires a large platform investment before it can begin. The order is consistent with the MLOps reference architecture of Kreuzberger et al., in which data and model versioning, automated pipelines, and monitoring are foundational components rather than later additions [16]. Paleyes et al. note that many reported deployment failures stem from skipping precisely these foundational steps [14].

---

## 7 DISCUSSION AND LIMITATIONS

The model presented here is a synthesis rather than an empirical study. It organizes published techniques and reported practitioner accounts into a structure that can be adopted, but it does not report measured improvements in incident frequency or detection time. Those measurements are the natural next step, and they should be collected within organizations that adopt the model, with careful attention to confounding factors such as changes in team size, traffic, or system scope during the measurement period.

Several tensions remain unresolved. The first is cost: validation, reconciliation, and drift monitoring consume compute and engineering attention, and the right level of investment depends on the consequence of a wrong answer, which differs between an internal dashboard and an automated control that triggers external action. The second is ownership: contracts and SLOs require a named owner for every data product, which organizations with a long history of shared, unowned tables find difficult to establish. The third is the definition of correctness for models whose ground truth is delayed, ambiguous, or contested; the proxy metrics recommended in Section 5 are pragmatic but imperfect. The fourth is the interaction between reliability policy and experimentation velocity for AI teams, where an error budget policy that is too strict can prevent the retraining that is needed to fix a behavioral failure.

Finally, this paper deliberately avoids tying the model to particular products. That choice makes the paper durable but leaves implementation details to the reader. The practices described map onto most modern data platforms and model serving stacks, but the effort required to implement them varies considerably with the tooling in place.

## 8 CONCLUSION

Data and AI systems fail differently from conventional services: the most damaging failures are silent, and standard service centric instrumentation does not detect them without data specific and model specific extensions. This paper proposed a failure taxonomy that separates availability and freshness failures, which are generally more visible to existing tooling, from correctness and behavioral failures, which require data contracts, automated validation, reconciliation, skew and drift monitoring, and outcome quality measurement. It defined SLIs for each category so that SLOs, error budgets, and burn rate alerting can be applied uniformly, and it described an operating model (Figure 1) that links design time, deployment time, and run time controls through incident response and postmortems. The model is vendor neutral and can be adopted in stages. Future work should measure the effect of adoption on detection time and incident frequency, and should refine outcome quality measurement for models with delayed or ambiguous ground truth.

## STATEMENTS ON COMPLIANCE WITH ETHICAL STANDARDS

### *Acknowledgments*

The author would like to thank industry peers and reviewers for their constructive feedback on earlier drafts of this work.

### *Disclosure of conflict of interest*

The author declares no conflict of interest.

## REFERENCES

- [1] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., *Site Reliability Engineering: How Google Runs Production Systems*. Sebastopol, CA, USA: O'Reilly Media, 2016.
- [2] B. Beyer, N. R. Murphy, D. K. Rensin, K. Kawahara, and S. Thorne, Eds., *The Site Reliability Workbook: Practical Ways to Implement SRE*. Sebastopol, CA, USA: O'Reilly Media, 2018.
- [3] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J. F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, Montreal, Canada, 2015, pp. 2503-2511.
- [4] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ML test score: A rubric for ML production readiness and technical debt reduction," in *Proc. IEEE International Conference on Big Data*, Boston, MA, USA, 2017, pp. 1123-1132.
- [5] N. Sambasivan, S. Kapania, H. Highfill, D. Akrong, P. Paritosh, and L. M. Aroyo, "Everyone wants to do the model work, not the data work: Data cascades in high-stakes AI," in *Proc. CHI Conference on Human Factors in Computing Systems*, Yokohama, Japan, 2021.
- [6] M. T. Nygard, *Release It! Design and Deploy Production-Ready Software*, 2nd ed. Raleigh, NC, USA: Pragmatic Bookshelf, 2018.
- [7] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal, "Chaos engineering," *IEEE Software*, vol. 33, no. 3, pp. 35-41, 2016.
- [8] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data management challenges in production machine learning," in *Proc. ACM SIGMOD International Conference on Management of Data*, Chicago, IL, USA, 2017, pp. 1723-1726.
- [9] E. Breck, M. Zinkevich, N. Polyzotis, S. Whang, and S. Roy, "Data validation for machine learning," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 334-347, 2019.
- [10] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Biessmann, and A. Grafberger, "Automating large-scale data quality verification," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 1781-1794, 2018.
- [11] M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. Sebastopol, CA, USA: O'Reilly Media, 2017.
- [12] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernandez-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle, "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*, vol. 8, no. 12, pp. 1792-1803, 2015.

- [13] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in Proc. IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Montreal, Canada, 2019, pp. 291-300.
- [14] A. Paleyes, R. G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: A survey of case studies," ACM Computing Surveys, vol. 55, no. 6, article 114, 2022.
- [15] S. Shankar, R. Garcia, J. M. Hellerstein, and A. G. Parameswaran, "We have no idea how models will behave in production until production: How engineers operationalize machine learning," Proceedings of the ACM on Human-Computer Interaction, vol. 8, no. CSCW1, article 206, pp. 1-34, Apr. 2024.
- [16] D. Kreuzberger, N. Kuhl, and S. Hirschl, "Machine learning operations (MLOps): Overview, definition, and architecture," IEEE Access, vol. 11, pp. 31866-31879, 2023.
- [17] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar, "Accelerating the machine learning lifecycle with MLflow," IEEE Data Engineering Bulletin, vol. 41, no. 4, pp. 39-45, 2018.
- [18] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," ACM Computing Surveys, vol. 46, no. 4, article 44, 2014.
- [19] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," IEEE Transactions on Knowledge and Data Engineering, vol. 31, no. 12, pp. 2346-2363, 2019.
- [20] H. Foidl, V. Golendukhina, R. Ramler, and M. Felderer, "Data pipeline quality: Influencing factors, root causes of data-related issues, and processing problem areas for developers," Journal of Systems and Software, vol. 207, article 111855, 2024.
- [21] S. Shankar, L. Fawaz, K. Gyllstrom, and A. Parameswaran, "Automatic and precise data validation for machine learning," in Proc. 32nd ACM International Conference on Information and Knowledge Management (CIKM), Birmingham, UK, 2023, pp. 2198-2207.