

Data Quality, Feature Engineering, and Model Reliability in Large-Scale AI Multi-Agent Systems

Harsh Verma

Palo Alto Networks, Artificial Intelligence, United States.

Abstract: *Large-scale artificial intelligence (AI) systems increasingly operate not as a single monolithic model but as a population of interacting, specialized agents separate models or decision-making components responsible for functions such as pricing, fraud detection, ranking, routing, and customer support that share underlying data infrastructure and, in many cases, influence one another's inputs and outputs. This article synthesizes peer-reviewed literature on data quality assurance, feature engineering, and model reliability to examine how these three concerns interact once an AI system is decomposed into multiple cooperating agents operating at scale. Seventeen primary sources are reviewed, spanning foundational work on technical debt in machine learning systems, multi-dimensional data quality frameworks, scalable and automated data quality verification, data lifecycle management, empirically grounded data-management taxonomies, production-readiness testing rubrics, scalable and automated feature engineering, hyperparameter optimization, organizational workflow studies, a production-scale ML platform, concept drift, large-scale academic surveys of ML testing, systematic reviews of industrial ML challenges, and fault-tolerant cooperative control of multi-agent systems. Drawing on this literature, the article proposes a conceptual framework linking data quality assurance, feature engineering, model training and reliability testing, and deployment to an agent population, closed by a cross-agent monitoring and feedback loop. The review finds that data quality problems and model reliability failures do not remain confined to the agent in which they originate: because agents in a large-scale AI system typically share upstream data sources, feature pipelines, or downstream state, a defect introduced at the data or feature layer of one agent can propagate through the interactions between agents, producing system-level reliability failures that are not visible from the perspective of any single agent's test suite. Ensuring reliability in such systems therefore requires treating data quality, feature engineering, and testing as cross-cutting, system-wide concerns rather than as properties to be verified independently within each agent.*

Keywords: data quality, feature engineering, model reliability, multi-agent systems, large-scale AI, concept drift, ML testing, technical debt, fault tolerance

INTRODUCTION

As artificial intelligence systems have grown from single, standalone models into large, composite platforms, it has become increasingly common for such systems to be structured as a population

of specialized, interacting agents rather than a single unified model. In a large e-commerce platform, for example, one model may set prices, another may score transactions for fraud, another may rank search results, another may route customer support tickets, and another may generate personalized recommendations each operating semi-independently, yet drawing on shared upstream data sources and, in many cases, consuming one another's outputs as inputs. This architectural pattern raises the stakes of three concerns that have historically been studied largely in isolation: the quality of the data feeding the system, the process by which raw data is transformed into the features models consume, and the reliability of the resulting models once deployed.

Each of these concerns has an established, independent literature. Data quality has long been studied in the context of both traditional data management and, more recently, machine learning specifically (Gudivada, Apon, & Ding, 2017; Schelter et al., 2018). Feature engineering has increasingly been the subject of automation research aimed at reducing the manual effort and inconsistency involved in transforming raw data into usable model inputs (Shi et al., 2020; He, Zhao, & Chu, 2020). Model reliability has been formalized through production-readiness rubrics (Breck et al., 2017), large-scale testing surveys (Zhang, Harman, Ma, & Liu, 2020; Riccio et al., 2020), and frameworks for detecting and adapting to concept drift (Lu et al., 2019). What is less well established is how these three concerns interact once a system is no longer a single model but a population of interacting agents, each with its own data dependencies, feature pipelines, and reliability requirements, but sharing an underlying data and infrastructure substrate.

A separate, largely independent literature has studied reliability and fault tolerance specifically in the context of multi-agent systems, though predominantly in control-systems and robotics settings rather than in the context of machine learning pipelines (Yang et al., 2020). That literature has developed a vocabulary for reasoning about how faults propagate through a population of cooperating agents and how a system can respond individually, cooperatively, or by reconfiguring its structure once a fault is detected. This article's contribution is to bring that vocabulary into contact with the data quality, feature engineering, and model reliability literature developed for machine learning systems, most of which, as noted above, was developed with a single model in mind.

Motivation

A data quality defect introduced upstream of one agent does not necessarily remain confined to that agent. If a pricing agent and a fraud-detection agent both consume the same transaction stream, a schema change or a silent distributional shift in that stream can degrade both agents simultaneously, even though each agent's own test suite may report normal behavior when evaluated on its own recent inputs. Similarly, if one agent's output becomes another agent's input for instance, a ranking agent's output feeding a downstream recommendation agent a reliability failure in the first agent can propagate as a distributional shift into the second, a form of drift that neither agent's individual monitoring is designed to detect. This kind of cross-agent propagation is precisely the gap that the technical-debt literature on hidden feedback loops and undeclared

consumers first identified in single-model systems (Sculley et al., 2015), but which becomes structurally more likely, and potentially more damaging, once a system is composed of many interacting agents.

Problem Statement

Organizations building large-scale, multi-agent AI systems need architectural patterns and engineering practices that can simultaneously: (a) assure data quality not only within each agent's own pipeline but across the shared data substrate multiple agents draw upon; (b) engineer features consistently, and where appropriate reuse them safely, across agents with related but distinct modeling objectives; (c) test and monitor model reliability not only per agent but at the level of the interactions between agents; and (d) detect and respond to reliability degradation that originates in one agent but manifests as a downstream effect in another. The literature reviewed in this article shows that these requirements are not well served by treating each agent's data quality, feature engineering, and reliability testing as an independent, siloed concern.

Objectives

- Review the literature most directly relevant to data quality assurance, feature engineering, and model reliability in large-scale, multi-component AI systems.
- Propose a conceptual framework linking these three concerns across a population of interacting agents, closed by a cross-agent monitoring and feedback loop.
- Present the concrete frameworks, rubrics, and reported findings each source contributes.
- Discuss the trade-offs and cross-agent propagation risks that emerge from this body of work.

Related Work

Technical Debt as a Framing for Large-Scale AI Systems

Sculley et al. (2015) applied the software-engineering concept of technical debt to machine learning systems, identifying risk factors including entanglement, in which changing any single input, feature, or hyperparameter can silently change the behavior of the entire system; hidden feedback loops, in which a model's own predictions influence the future data it will be trained on without this dependency being visible in the code; undeclared consumers, in which other systems silently come to depend on a model's output without the model owner's knowledge; and data dependencies, which the authors argue are as costly as code dependencies but far less commonly tracked. Although the paper was written primarily with single-model systems in mind, its framing anticipates the central challenge of multi-agent AI systems directly: when several agents share data sources or consume one another's outputs, the entanglement and hidden feedback loops the authors describe are no longer confined within one system boundary but can cross from one agent into another.

Foundations of Data Quality for Machine Learning

Gudivada, Apon, and Ding (2017) proposed a multi-dimensional framework for data quality in the context of big data and machine learning, arguing that data quality assurance must extend beyond conventional data cleaning and transformation to address dimensions such as accuracy, completeness, consistency, currency, and provenance, each of which can degrade a model's performance in different ways even when the data passes basic cleaning steps. The authors emphasize that data quality problems are often domain- and task-specific: a dataset that is high-quality for one modeling task may be low-quality for another, depending on which dimensions matter most for that task's objective. This framing is directly relevant to multi-agent systems, where different agents consuming the same underlying data may have different quality requirements — a fraud-detection agent may be highly sensitive to completeness and timeliness, while a ranking agent may be more sensitive to consistency across data sources — meaning a single, uniform data quality standard may be insufficient across an agent population.

Automating Data Quality Verification at Scale

Schelter et al. (2018) presented Deequ, a system for automating large-scale data quality verification through declarative constraints that can be checked continuously against production data. Rather than requiring engineers to write ad hoc validation code for each dataset, Deequ allows data quality requirements such as completeness, uniqueness, and value-range constraints to be expressed declaratively and evaluated efficiently over very large datasets. The system was evaluated against production data at substantial scale, demonstrating that automated, declarative data quality verification could operate within the throughput constraints of a large production data pipeline. This paper is significant for multi-agent systems because it demonstrates a practical mechanism by which a shared data quality verification layer could, in principle, serve multiple downstream consumers — multiple agents — simultaneously, rather than requiring each agent's team to independently reimplement equivalent checks.

Data Lifecycle Management and Training-Serving Skew

Polyzotis, Roy, Whang, and Zinkevich (2018) argued that data collection, validation, and cleaning should be treated as first-class engineering concerns with dedicated tooling, highlighting training-serving skew a discrepancy between the data or feature-computation logic used at training time and that encountered at serving time as a particularly dangerous and hard-to-detect failure mode. In a system composed of multiple agents, this concern is compounded: if two agents independently implement feature-computation logic against a common upstream data source, skew can arise not only between each agent's own training and serving paths, but between agents, such that two agents nominally computing 'the same' feature from the same raw data arrive at subtly different values, undermining any downstream logic that assumes consistency between them.

Empirically Grounded Data-Management Challenges

Munappy, Bosch, Olsson, Arpteg, and Brinne (2019) interviewed deep learning practitioners and identified data management challenges across seven concrete stages of development: data

collection, exploration, preprocessing, dataset preparation, data testing, deployment, and post-deployment. This staged structure gives the more conceptual data-lifecycle arguments of Polyzotis et al. (2018) and Gudivada et al. (2017) an empirically grounded shape, and is notable for identifying data testing and post-deployment data management as challenges distinct from, and occurring after, initial data preparation directly relevant to the ongoing, continuous data quality assurance that a multi-agent system's shared data substrate requires.

Production-Readiness Testing

Breck, Cai, Nielsen, Salib, and Sculley (2017) proposed the ML Test Score, a rubric of 28 specific tests and monitoring needs organized into four categories: data tests (for example, verifying that feature distributions match expectations), model tests (for example, testing model quality before serving and comparing against simpler baselines), ML infrastructure tests (for example, verifying training pipeline reproducibility and rollback capability), and monitoring tests (for example, tracking prediction quality and monitoring upstream data for change). The authors propose scoring a system from minimal to comprehensive coverage across all four categories, framing production readiness as a measurable, incremental property. Applied to a multi-agent system, this rubric raises an immediate question the original single-system framing does not address: whether each agent should be scored independently, or whether certain categories particularly data tests, which often apply to data shared across agents — should be evaluated once at the level of the shared data substrate rather than redundantly within each agent.

Scalable, Automated Feature Engineering

Shi, Zhang, Li, Yang, Li, and Zhou (2020) presented SAFE, a scalable automatic feature engineering framework designed for industrial-scale tasks, addressing the practical reality that manual feature engineering does not scale to the number of features, data sources, and modeling tasks present in large industrial systems. The framework automates feature generation and selection while remaining computationally efficient enough to operate on industrial-scale data volumes. This paper is significant for multi-agent systems because automated feature engineering, if applied consistently across agents drawing on shared data, offers a path toward the feature consistency that Polyzotis et al. (2018) identify as necessary to avoid skew provided the automation itself is applied through a shared, rather than per-agent, pipeline.

AutoML and the Automation of Feature Engineering

He, Zhao, and Chu (2020) surveyed automated machine learning (AutoML) methods spanning data preparation, feature engineering, model selection, and hyperparameter optimization, situating automated feature construction within the broader automated modeling pipeline. The survey highlights that feature engineering automation is not an isolated technique but interacts with, and is often jointly optimized alongside, model selection and hyperparameter search. This joint-optimization perspective is relevant to multi-agent systems because it suggests that feature engineering decisions made for one agent's modeling task may not transfer cleanly to another

agent's task, even where the underlying raw data is shared, since the optimal feature representation is entangled with each agent's specific model and objective.

Hyperparameter Optimization

Yang and Shami (2020) reviewed hyperparameter optimization theory and practice, covering grid search, random search, and Bayesian optimization approaches, and discussing the trade-offs each presents in terms of computational cost and solution quality. Although hyperparameter optimization is typically framed as a per-model concern, in a multi-agent system where computational resources are shared across many agents' training pipelines, the cost trade-offs the authors document take on additional significance: an organization must decide how to allocate optimization budget across an entire population of agents rather than optimizing a single model in isolation.

Organizational Workflow Across Engineering Teams

Amershi et al. (2019) reported an empirical study of software teams developing AI-based applications, deriving a nine-stage workflow — model requirements, data collection, data cleaning, data labeling, feature engineering, model training, model evaluation, model deployment, and model monitoring — from direct observation of engineering teams. The study found that discoverability and reuse of data, features, and models across teams was a recurring source of friction, a finding with direct relevance to multi-agent systems: if separate teams own separate agents, the same friction Amershi et al. document within a single organization's workflow is likely to recur, and potentially intensify, at the boundaries between agent-owning teams that must coordinate around shared data and features.

A Production-Scale Platform Integrating These Concerns

Baylor et al. (2017) described TFX, a production-scale machine learning platform that integrates data validation, transformation, model training, evaluation, and serving into a single orchestrated platform, built to address the fragility of ad hoc, glue-code-based orchestration between separately built pipeline components. TFX is significant as a concrete demonstration that data validation and feature transformation logic can be shared, standardized infrastructure rather than being reimplemented per model — directly relevant to a multi-agent setting, where the alternative of each agent maintaining its own bespoke validation and transformation code multiplies both engineering effort and the risk of the inter-agent skew that Polyzotis et al. (2018) warn against.

Concept Drift as an Ongoing Reliability Concern

Lu, Liu, Dong, Gu, Gama, and Zhang (2019) reviewed concept drift — the phenomenon in which the statistical relationship between a model's inputs and its target changes over time — organizing the literature into a three-part framework of drift detection, drift understanding, and drift adaptation, and distinguishing sudden, gradual, incremental, and recurring drift patterns. In a multi-agent system, drift takes on an additional dimension beyond the single-model case the authors primarily address: because one agent's output can become another agent's input, drift in

one agent's behavior can appear, from a downstream agent's perspective, as drift in the input data itself, even though nothing about the external world has changed — a form of internally generated, cross-agent drift that a monitoring system designed only to watch for externally caused drift would not anticipate.

Large-Scale Academic Surveys of ML Testing

Two large-scale academic surveys corroborate and extend the ML Test Score from an academic-literature perspective. Zhang, Harman, Ma, and Liu (2020) surveyed 144 papers on machine learning testing, classifying research by the testing properties addressed (such as correctness, robustness, and fairness), the system components targeted (data, learning program, and framework), and where in the testing workflow each technique applies. Riccio, Jahangirova, Stocco, Humbatova, Weiss, and Tonella (2020) conducted an independently constructed systematic mapping of ML testing research along closely related dimensions. Braiek and Khomh (2020) focused specifically on testing machine learning programs, proposing a taxonomy of testing techniques tailored to the unique characteristics of ML code, such as the absence of a specification against which correctness can be checked in the conventional sense. Together, these three sources confirm that ML testing is a substantial, independently developed field, and their component-based taxonomies — distinguishing tests that target data, the learning program, or the surrounding framework — map naturally onto the question of which tests in a multi-agent system should be run once, at the shared-data or shared-framework level, and which must be run separately for each agent's distinct learning program.

Systematic Review of Industrial-Scale Challenges

Lwakatare, Raj, Crnkovic, Bosch, and Olsson (2020) conducted a systematic literature review of the development and maintenance of large-scale, industrial ML-based software systems, synthesizing evidence from 72 papers and six case companies. The review's synthesis spans data management challenges consistent with Polyzotis et al. (2018) and Munappy et al. (2019), testing challenges consistent with Breck et al. (2017), and deployment and monitoring challenges consistent with Lu et al.'s (2019) treatment of drift, providing evidence that these challenges recur across many different organizations and industrial contexts rather than being idiosyncratic to any single company's experience.

Fault Tolerance and Reliability in Multi-Agent Systems

Yang, Han, Ge, Ding, Xu, Jiang, and Zhou (2020) surveyed fault-tolerant cooperative control of multiagent systems, providing a framework developed independently of the machine learning literature reviewed above but directly relevant to it. The authors classify fault-tolerant cooperative control methodologies into four categories: individual methodologies, in which each agent compensates for its own faults independently; cooperative methodologies, in which agents share information to jointly compensate for a fault affecting one member; topology-reconfiguration methodologies, in which the communication or dependency structure among agents is altered in response to a fault; and composition-reconfiguration methodologies, in which the set of agents

performing a task is itself changed. The authors note that multi-agent systems can provide more redundancy than a single-agent system, making fault-tolerant design more flexible, but also introduce challenges — such as fault propagation through inter-agent coupling — that do not exist in single-agent systems. This classification, though developed in a control-systems context rather than a machine learning context, offers a structured way to think about reliability strategies for multi-agent AI systems: a data quality or model reliability fault in one agent could, in principle, be addressed individually within that agent, cooperatively by drawing on signals from other agents, by reconfiguring which agents depend on which shared data sources, or by reconfiguring which agent is responsible for a given task altogether.

Overview of the Reviewed Literature

Figure 2 summarizes the distribution of the primary literature reviewed in this section by publication year. The literature is concentrated toward a small number of years, with the largest cluster addressing production-readiness testing, scalable feature engineering, and industrial-scale review work appearing together, alongside the multi-agent fault-tolerance literature that, while methodologically distinct, was published in the same period and addresses a structurally related problem: how to maintain reliable operation across a population of interacting, potentially faulty components.

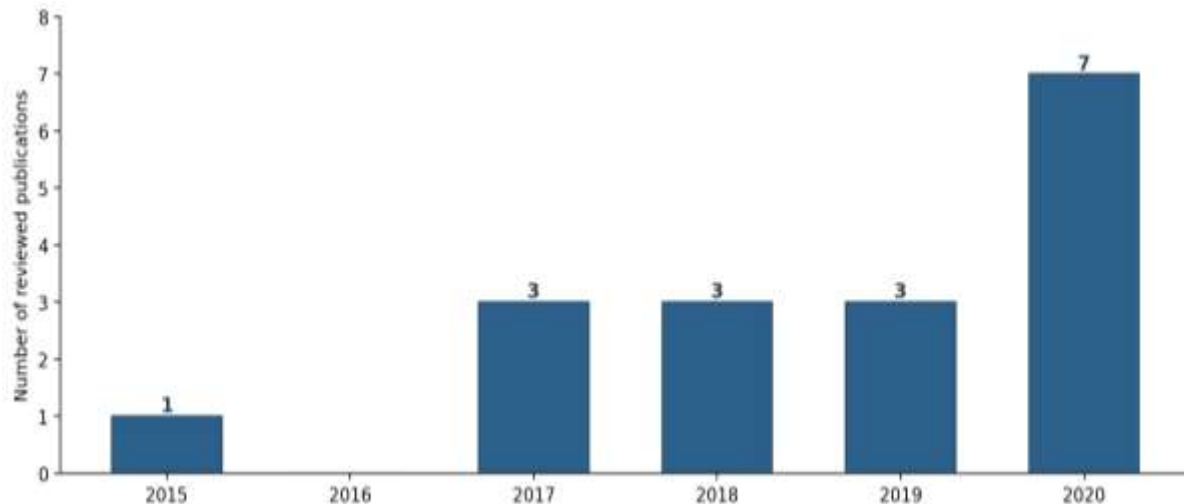


Figure 2. Distribution of the reviewed literature by publication year.

Conceptual Framework

Synthesizing the literature reviewed in Section 2, data quality, feature engineering, and model reliability in a large-scale, multi-agent AI system can be organized into a framework spanning a shared processing pipeline, a population of interacting agents, and a cross-agent monitoring and feedback layer that closes the loop back to the shared pipeline. Figure 1 illustrates this framework.

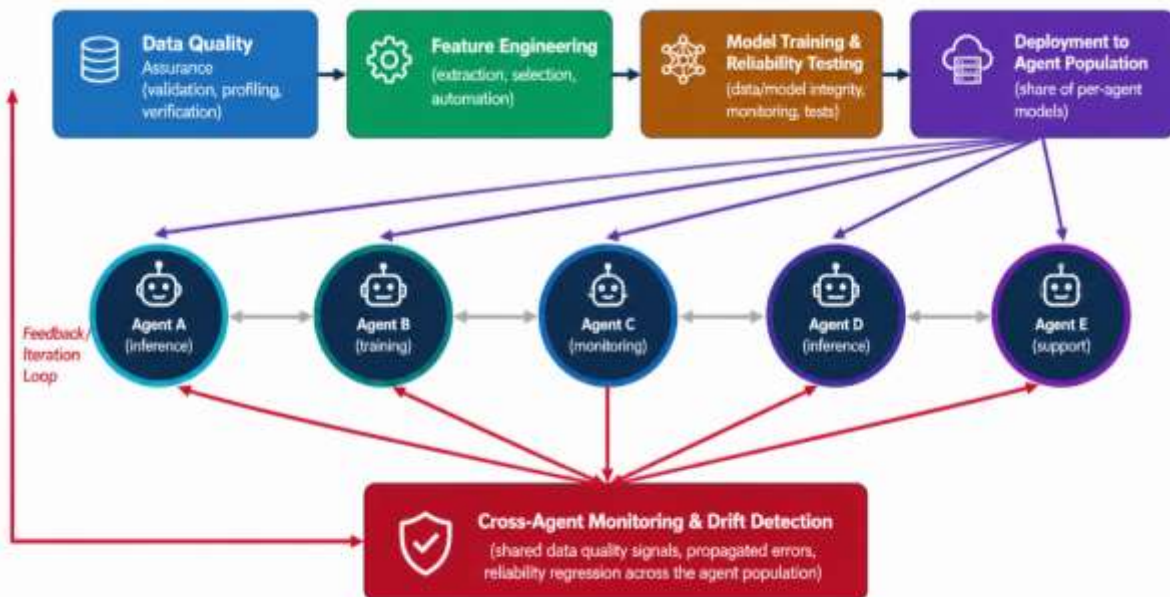


Figure 1. Conceptual framework linking data quality, feature engineering, and model reliability across a population of interacting agents in a large-scale AI system.

Data Quality Assurance Layer

Following Gudivada et al. (2017) and Schelter et al. (2018), a shared data quality assurance layer applies declarative, automatically checked constraints to data before it reaches any downstream feature pipeline. Because this layer sits upstream of individual agents, quality problems can, in principle, be caught once, for the benefit of every agent consuming that data, rather than being independently (and inconsistently) checked within each agent's own pipeline, echoing the case Munappy et al. (2019) make for treating data testing as a distinct, ongoing engineering activity rather than a one-time step.

Feature Engineering Layer

Validated data passes to a feature engineering layer that, following the scalable and automated approaches of Shi et al. (2020) and He, Zhao, and Chu (2020), can generate and select candidate features efficiently at industrial scale. Where multiple agents require related but distinct features from the same underlying data, standardizing this layer rather than allowing each agent to implement its own feature-computation logic independently — reduces the risk of the training-serving and cross-agent skew that Polyzotis et al. (2018) identify as a central data-lifecycle risk.

Model Training and Reliability Testing Layer

Each agent's model is trained on the features relevant to its task and evaluated against the production-readiness criteria of Breck et al. (2017), informed by the broader testing taxonomies of Zhang et al. (2020), Riccio et al. (2020), and Braiek and Khomh (2020). Consistent with the

distinction drawn in Section 2.13, data tests and certain infrastructure tests can often be run once against the shared pipeline, while model tests specific to each agent's learning objective must be run per agent.

Deployment to Agent Population

Validated models are deployed across the agent population, following the integrated deployment approach demonstrated by Baylor et al. (2017), whether as distinct models for distinct agents or, in some cases, as shared models serving multiple related agent functions. Sculley et al. (2015) caution that this stage is where undeclared consumers most often emerge, a risk that is structurally heightened once a system contains many agents that may come to depend on one another's outputs without those dependencies being explicitly tracked.

Cross-Agent Monitoring and Feedback

Once deployed, agents' behavior is monitored not only individually but for cross-agent effects: drift detected in one agent's inputs, following the framework of Lu et al. (2019), may originate externally or may originate in another agent's output, following the internally generated drift risk discussed in Section 2.12. When a fault is detected, the response strategy can follow any of the four categories Yang et al. (2020) identify in the fault-tolerant control literature — addressed within the affected agent individually, addressed cooperatively using signals from related agents, addressed by reconfiguring which agents depend on which data sources, or addressed by reconfiguring which agent is responsible for the affected task — with the outcome feeding back into the shared data quality assurance layer, closing the loop shown in Figure 1.

Comparative Summary of Reviewed Sources

Table 1 summarizes the seventeen sources reviewed in Section 2 according to their focus area, key contribution, and venue type, illustrating how each maps onto the framework described above.

Source	Focus Area	Key Contribution	Venue Type
Sculley et al. (2015)	Technical debt	ML-specific risk factors: entanglement, hidden feedback loops, undeclared consumers, data dependencies	NeurIPS
Gudivada et al. (2017)	Data quality foundations	Multi-dimensional data quality framework going beyond cleaning/transformation	Int'l J. Advances in Software
Breck et al. (2017)	Production readiness	28-point ML Test Score across data, model, infrastructure, monitoring tests	IEEE Big Data

Source	Focus Area	Key Contribution	Venue Type
Schelter et al. (2018)	Automated data quality verification	Deequ: declarative, scalable data quality constraint checking	Proc. VLDB Endowment
Polyzotis et al. (2018)	Data lifecycle management	Training-serving skew and continuous data validation as first-class concerns	ACM SIGMOD Record
Munappy et al. (2019)	Data management for deep learning	Seven-stage empirical taxonomy of data challenges from practitioner interviews	IEEE SEAA
Amershi et al. (2019)	Organizational workflow	Nine-stage ML workflow observed across engineering teams	IEEE/ACM ICSE-SEIP
Baylor et al. (2017)	Production platform	TFX: integrated validation, transform, training, evaluation, serving platform	ACM KDD
Shi et al. (2020)	Scalable feature engineering	SAFE: automatic feature generation framework for industrial-scale tasks	IEEE ICDE
He, Zhao, & Chu (2020)	AutoML / feature automation	Survey spanning data preparation, feature engineering, and hyperparameter search	Knowledge-Based Systems, Elsevier
Yang & Shami (2020)	Hyperparameter optimization	Theory and practice of grid, random, and Bayesian optimization approaches	Neurocomputing, Elsevier
Lu et al. (2019)	Concept drift	Unifying framework: drift detection, understanding, and adaptation	IEEE Trans. Knowledge and Data Engineering
Zhang et al. (2020)	ML testing survey	144-paper survey of ML testing properties, components, and workflow	IEEE Trans. Software Engineering
Riccio et al. (2020)	ML testing mapping	Systematic mapping of testing research by property, component, and workflow stage	Empirical Software Engineering, Springer

Source	Focus Area	Key Contribution	Venue Type
Braiek & Khomh (2020)	Testing ML programs	Taxonomy of testing techniques tailored to ML program characteristics	Journal of Systems and Software, Elsevier
Lwakatare et al. (2020)	Industrial-scale challenges	Systematic review of 72 papers synthesizing large-scale ML system challenges	Information and Software Technology, Elsevier
Yang et al. (2020)	Multi-agent fault tolerance	Survey classifying fault-tolerant cooperative control into four methodology classes	IEEE Trans. Industrial Informatics

Table 1. Comparative summary of the primary literature reviewed in this article.

Reported Frameworks and Data Points

Table 2 consolidates the concrete frameworks, rubrics, and findings reported directly in the primary sources reviewed in Section 2.

Source	Reported Data Point
Sculley et al. (2015)	7 categories of ML-specific technical debt risk, plus system-level anti-patterns (glue code, pipeline jungles)
Gudivada et al. (2017)	Multi-dimensional data quality model spanning accuracy, completeness, consistency, and timeliness, applied to big data and ML pipelines
Breck et al. (2017)	28 tests across 4 categories (data, model, infrastructure, monitoring), scored on a 0-5 production-readiness scale
Schelter et al. (2018)	Declarative constraint language (Deequ) evaluated on production datasets with billions of records at Amazon
Polyzotis et al. (2018)	Data lifecycle framed around 3 stages: collection/preparation, validation, and cleaning, plus continuous skew monitoring
Munappy et al. (2019)	7 concrete data-management stages identified from interviews with deep learning practitioners
Amershi et al. (2019)	9-stage ML workflow derived from observing multiple engineering teams building AI-based applications
Shi et al. (2020)	Scalable automatic feature engineering framework evaluated on industrial-scale tasks with millions of records

Source	Reported Data Point
He, Zhao, & Chu (2020)	Survey spanning automated data preparation, feature engineering, model selection, and hyperparameter optimization stages
Yang & Shami (2020)	Comparative review of grid search, random search, and Bayesian optimization for hyperparameter tuning
Lu et al. (2019)	3-part concept drift framework; 4 drift-pattern taxonomy (sudden, gradual, incremental, recurring)
Zhang et al. (2020)	144 papers surveyed on ML testing properties (correctness, robustness, fairness), components, and workflow
Riccio et al. (2020)	Systematic mapping study classifying ML testing research by property, target component, and workflow stage
Lwakatare et al. (2020)	72 papers and evidence from 6 case companies synthesizing industrial ML system challenges
Yang et al. (2020)	4 methodology classes for fault-tolerant cooperative control: individual, cooperative, topology-reconfiguration, composition-reconfiguration

Table 2. Frameworks and reported data points drawn directly from the primary literature reviewed in this article.

Discussion: Challenges and Trade-offs

Data Quality Propagation Across Agents

Because multiple agents in a large-scale AI system typically draw on shared upstream data, a data quality defect that would be caught by a single agent's own validation checks can nonetheless propagate to other agents before it is detected, particularly where validation is implemented independently and inconsistently across agents rather than centrally, as Schelter et al. (2018) demonstrate is feasible at scale. This creates a trade-off between the engineering convenience of allowing each agent team to own its own validation logic, and the systemic risk reduction available from centralizing validation at the shared data layer.

Centralized versus Per-Agent Feature Engineering

Shi et al. (2020) and He, Zhao, and Chu (2020) demonstrate that feature engineering can be automated and applied at scale, but He, Zhao, and Chu (2020) also note that feature engineering is often jointly optimized with model selection, meaning features well suited to one agent's task may not suit another's. Organizations must therefore weigh the skew-reduction benefits of shared feature pipelines (Section 4.1) against the performance cost of forcing agents with different objectives to share a feature representation that is not optimal for any single one of them.

Testing Coverage versus Multi-Agent Complexity

The four-category ML Test Score (Breck et al., 2017), when applied across an agent population, multiplies the testing surface: data and infrastructure tests may be shared, but model and monitoring tests generally cannot be, since each agent has a distinct learning objective. Zhang et al.'s (2020) and Riccio et al.'s (2020) component-based taxonomies suggest a principled way to manage this complexity by explicitly tagging each test as targeting shared data, a shared framework, or an individual agent's learning program but neither source was developed with multi-agent testing specifically in mind, leaving open how testing effort should be allocated as the number of agents grows.

Reliability at the System Level versus the Agent Level

A system composed of many individually reliable agents is not guaranteed to be reliable as a whole, since reliability failures can emerge from the interactions between agents rather than from any single agent's internal behavior — precisely the entanglement and hidden feedback loop risks Sculley et al. (2015) identify, now operating between agents rather than only within one. This implies that system-level reliability testing, evaluating the behavior of the agent population as a whole under realistic interaction patterns, is a distinct activity from per-agent reliability testing and cannot be inferred from per-agent test results alone.

Automation versus Interpretability in Feature Engineering

Automated feature engineering (Shi et al., 2020; He, Zhao, & Chu, 2020) reduces manual effort and can improve consistency across agents, but automatically generated features are often less interpretable than manually engineered ones, which can complicate root-cause analysis when a reliability problem is detected — it becomes harder to determine whether a fault originates in the data, the automated feature transformation, or the model itself. This trade-off is likely to be more consequential in a multi-agent system, where diagnosing whether a reliability problem originated in one agent or propagated from another already adds diagnostic complexity before automated feature opacity is considered.

Fault Tolerance Strategies Compared

The four fault-tolerance categories identified by Yang et al. (2020) — individual, cooperative, topology-reconfiguration, and composition-reconfiguration — each carry different costs and assumptions when mapped onto data quality and model reliability faults specifically, rather than the control-systems faults the framework was originally developed for. An individual response (an agent independently detecting and correcting its own data quality issue) is the simplest but assumes each agent has sufficient local information to detect the fault. A cooperative response (agents sharing reliability signals) can detect faults invisible to any single agent but requires inter-agent communication infrastructure. Topology-reconfiguration (changing which agents depend on which data sources) can isolate a faulty data source but may not be feasible if alternative sources do not exist. Composition-reconfiguration (replacing which agent performs a task) is the most

disruptive but may be necessary where an agent's own model, rather than its data, is the source of unreliability.

Drift Propagation in Interacting Agent Populations

Lu et al.'s (2019) drift taxonomy — sudden, gradual, incremental, and recurring — was developed for a single model observing an external environment, but in an interacting agent population, one agent's drift can be another agent's sudden shift, even where the underlying external environment has not changed at all. This suggests that multi-agent monitoring systems need to distinguish externally caused drift from internally propagated drift, since the appropriate response differs: externally caused drift typically calls for retraining against new data, while internally propagated drift may call for addressing the upstream agent rather than retraining every downstream agent that observes its effects.

Data Contracts and Shared Infrastructure Costs

Centralizing data quality assurance and feature engineering, as Sections 4.1 and 4.2 recommend, is not without cost. Building and operating shared infrastructure of the kind demonstrated by Baylor et al. (2017) and Schelter et al. (2018) requires upfront investment that a single agent team, building only for its own needs, would not otherwise incur. This creates an organizational trade-off distinct from the purely technical trade-offs discussed above: the benefit of shared infrastructure grows with the number of agents drawing on a common data source, meaning the centralization argument is strongest precisely in the large-scale, many-agent systems this article is concerned with, but weakest in the early stages of a system's growth, when only one or two agents exist and the overhead of building shared infrastructure may not yet be justified by the propagation risks it mitigates. Organizations should therefore expect the right balance between centralized and per-agent data and feature infrastructure to shift as an agent population grows, rather than fixing this balance once at the outset.

Practical Implementation Considerations

Establishing Shared Data Contracts Across Agents

Because Gudivada et al. (2017) show that data quality requirements can be task-specific, organizations should establish explicit data contracts specifying which quality dimensions (completeness, timeliness, consistency, and so on) each shared data source guarantees, so that agent teams can determine whether a given source meets their specific task's requirements rather than assuming uniform quality across all consumers.

Investing in Shared, Automated Feature Engineering Tooling

Following the scalable automation demonstrated by Shi et al. (2020) and the integrated platform approach of Baylor et al. (2017), organizations operating multiple agents against shared data should invest in centralized feature engineering infrastructure rather than allowing each agent team

to reimplement feature logic independently, reducing both duplicated effort and the risk of cross-agent skew documented in Section 4.1.

Applying Testing Rubrics at Both the Shared and Per-Agent Level

The ML Test Score (Breck et al., 2017) should be applied twice in a multi-agent system: once against the shared data and infrastructure layer, and once, separately, against each agent's own model and monitoring. Organizations should avoid the assumption that passing shared-layer tests implies any individual agent is production-ready, and equally avoid re-testing shared-layer properties redundantly within every agent.

Designing Cross-Agent Monitoring Infrastructure

Because internally propagated drift (Section 4.7) is invisible to monitoring that only compares each agent against its own historical baseline, organizations should build monitoring infrastructure capable of tracing a detected anomaly back through the dependency graph connecting agents, following the cooperative fault-tolerance principle described by Yang et al. (2020), so that a reliability problem can be attributed to its originating agent rather than addressed redundantly at every downstream agent that observes its symptoms.

Assigning Ownership Across Agent and Data Boundaries

Amershi et al. (2019) found that discoverability and reuse of data, features, and models across teams was a recurring source of friction within a single organization; this friction should be expected to recur, and potentially intensify, at the boundaries between teams that each own a different agent in a multi-agent system. Organizations should explicitly assign ownership not only for each agent but for each shared data source and shared feature pipeline, so that when a cross-agent reliability issue is detected, responsibility for remediation is clear rather than contested.

Illustrative Scenario: A Multi-Agent Marketplace Platform

Consider a hypothetical online marketplace operating a pricing agent, a fraud-detection agent, and a search-ranking agent, all drawing on a shared transaction and listing data stream. A shared data quality assurance layer, following Schelter et al. (2018), applies declarative constraints — completeness of price fields, valid currency codes, non-negative quantities — to the incoming stream before it reaches any agent. A shared feature engineering layer computes common features such as recent transaction volume per seller, following the standardized approach of Baylor et al. (2017), while each agent additionally computes features specific to its own task. Each agent is evaluated against the ML Test Score (Breck et al., 2017), with data and infrastructure tests inherited from the shared layer and model tests run independently per agent.

Suppose a currency-code validation rule in an upstream payments system is silently disabled, allowing malformed currency values to enter the shared stream. The fraud-detection agent, whose model treats currency mismatches as a strong fraud signal, begins flagging a rising share of legitimate transactions — a sudden drift, in Lu et al.'s (2019) terms, from the fraud agent's own

perspective. Because the pricing agent uses currency less centrally, its behavior degrades only gradually as downstream price comparisons become less reliable, appearing as gradual drift rather than a sudden fault. A monitoring system that only compared each agent to its own history would treat these as two unrelated incidents; a cross-agent monitoring layer capable of tracing both anomalies back to the shared currency field, following the cooperative fault-tolerance principle of Yang et al. (2020), would instead correctly identify a single upstream data quality fault and route remediation to the shared data layer rather than to either agent's model.

Metrics for Data Quality and Reliability at Scale

Organizations should track explicit metrics at each layer of Figure 1: the proportion of shared data quality constraints passing per batch (Section 3.1), the proportion of features computed identically across agents sharing the same underlying data (Section 3.2), per-agent ML Test Scores alongside a shared-layer score (Section 3.3), and the latency between a cross-agent anomaly being detected and its originating agent or data source being correctly identified (Section 3.5). Tracking the last of these metrics specifically operationalizes the cross-agent diagnostic capability that Section 4.7 argues is necessary but is not addressed by monitoring approaches inherited unmodified from single-model systems.

Governance Across Agent Boundaries

Because Amershi et al. (2019) and Sculley et al. (2015) each identify, from different angles, the risk of undeclared or poorly tracked dependencies between components, organizations operating multiple agents should maintain an explicit, queryable record of which agents consume which shared data sources and features — a dependency registry analogous to the data lineage tracking that Schelter et al. (2018) build into automated data quality verification. Without such a registry, the topology-reconfiguration and composition-reconfiguration fault-tolerance strategies Yang et al. (2020) describe are difficult to execute in practice, since reconfiguring which agents depend on which data sources first requires knowing, comprehensively, what the current dependency structure actually is. This governance function is distinct from, and complementary to, the per-agent ownership recommendation in Section 5.5: ownership assigns responsibility for a component, while a dependency registry makes the relationships between components visible enough that responsibility can be correctly assigned when a cross-agent problem is discovered.

CONCLUSION

Data quality, feature engineering, and model reliability have each been studied extensively as properties of individual machine learning systems: data quality as a multi-dimensional property of the inputs a model consumes (Gudivada et al., 2017; Schelter et al., 2018), feature engineering as a process that can be automated and scaled (Shi et al., 2020; He, Zhao, & Chu, 2020), and model reliability as a measurable, testable property with its own dedicated rubrics and large-scale academic surveys (Breck et al., 2017; Zhang et al., 2020; Riccio et al., 2020). What this article's synthesis adds is an account of how these three concerns interact once a system is composed of

multiple interacting agents rather than a single model: data quality defects, feature inconsistencies, and reliability failures do not remain confined to the agent in which they originate, but can propagate through the shared data substrate and the interaction structure connecting agents, in ways that per-agent testing and monitoring — however comprehensive — cannot detect on its own.

The conceptual framework proposed in Section 3, linking a shared data quality assurance layer, a shared feature engineering layer, per-agent model training and testing, deployment to an agent population, and a cross-agent monitoring and feedback loop, offers one way to reason about this interaction explicitly. The fault-tolerance taxonomy of Yang et al. (2020) — individual, cooperative, topology-reconfiguration, and composition-reconfiguration responses — provides a complementary vocabulary, developed independently in the control-systems literature, for reasoning about how a large-scale AI system should respond once a cross-agent reliability problem is identified. As AI systems continue to be built as populations of specialized, interacting agents rather than single monolithic models, the central implication of this review is that data quality, feature engineering, and reliability engineering practices developed for single models need to be explicitly extended — not merely repeated within each agent — to account for the propagation risks that only arise once agents are connected to one another and to a shared data substrate.

Practically, this implies a sequencing question for organizations building such systems: rather than treating data quality assurance, feature engineering standardization, and cross-agent monitoring as enhancements to be added once an agent population has already grown large, the literature synthesized in this review — from Sculley et al.'s (2015) early warning about hidden feedback loops to Schelter et al.'s (2018) demonstration that data quality verification can be automated at very large scale — suggests these capabilities are most effectively built into the shared substrate from the first two or three agents onward, before the dependency structure connecting agents becomes complex enough that retrofitting shared infrastructure, and reconstructing the dependency visibility that Section 5.8 argues is a precondition for effective fault response, becomes substantially more costly than building it in from the outset.

REFERENCES

- Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., & Zimmermann, T. (2019). Software engineering for machine learning: A case study. 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 291–300. <https://doi.org/10.1109/ICSE-SEIP.2019.00042>
- Verma, H., & Krishnan, A. (2017). Sivasubramanian. Analytics Performance Load Test. *International Journal of Computer Applications*, 176(2), 10-13.
- Baylor, D., Breck, E., Cheng, H.-T., Fiedel, N., Foo, C. Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., Koo, C. Y., Lew, L., Mewald, C., Modi, A. N., Polyzotis, N., Ramesh, S.,

-
- Roy, S., Whang, S. E., Wicke, M., Wilkiewicz, J., Zhang, X., & Zinkevich, M. (2017). TFX: A TensorFlow-based production-scale machine learning platform. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*, 1387–1395. <https://doi.org/10.1145/3097983.3098021>
- Verma, H. (2019). Secure Real-Time Heterogeneous IoT Data Management System. Available at SSRN 6573879.
- Braiek, H. B., & Khomh, F. (2020). On testing machine learning programs. *Journal of Systems and Software*, 164, 110542. <https://doi.org/10.1016/j.jss.2020.110542>
- Breck, E., Cai, S., Nielsen, E., Salib, M., & Sculley, D. (2017). The ML test score: A rubric for ML production readiness and technical debt reduction. *2017 IEEE International Conference on Big Data (Big Data)*, 1123–1132. <https://doi.org/10.1109/BigData.2017.8258038>
- Gudivada, V. N., Apon, A., & Ding, J. (2017). Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations. *International Journal on Advances in Software*, 10(1), 1–20.
- He, X., Zhao, K., & Chu, X. (2020). AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212, 106622. <https://doi.org/10.1016/j.knosys.2020.106622>
- Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2019). Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12), 2346–2363. <https://doi.org/10.1109/TKDE.2018.2876857>
- Lwakatare, L. E., Raj, A., Crnkovic, I., Bosch, J., & Olsson, H. H. (2020). Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions. *Information and Software Technology*, 127, 106368. <https://doi.org/10.1016/j.infsof.2020.106368>
- Munappy, A. R., Bosch, J., Olsson, H. H., Arpteg, A., & Brinne, B. (2019). Data management challenges for deep learning. *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 140–147. <https://doi.org/10.1109/SEAA.2019.00030>
- Polyzotis, N., Roy, S., Whang, S. E., & Zinkevich, M. (2018). Data lifecycle challenges in production machine learning: A survey. *ACM SIGMOD Record*, 47(2), 17–28. <https://doi.org/10.1145/3299887.3299891>
- Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., & Tonella, P. (2020). Testing machine learning based systems: A systematic mapping. *Empirical Software Engineering*, 25(6), 5193–5254. <https://doi.org/10.1007/s10664-020-09881-0>
- Schelter, S., Lange, D., Schmidt, P., Celikel, M., Biessmann, F., & Grafberger, A. (2018). Automating large-scale data quality verification. *Proceedings of the VLDB Endowment*, 11(12), 1781–1794. <https://doi.org/10.14778/3229863.3229867>
- Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., & Dennison, D. (2015). Hidden technical debt in machine learning systems. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.),

- Advances in Neural Information Processing Systems 28 (NeurIPS 2015) (pp. 2503–2511).
- Shi, Q., Zhang, Y.-L., Li, L., Yang, X., Li, M., & Zhou, J. (2020). SAFE: Scalable automatic feature engineering framework for industrial tasks. 2020 IEEE 36th International Conference on Data Engineering (ICDE), 1645–1656.
<https://doi.org/10.1109/ICDE48307.2020.00146>
- Yang, H., Han, Q.-L., Ge, X., Ding, L., Xu, Y., Jiang, B., & Zhou, D. (2020). Fault-tolerant cooperative control of multiagent systems: A survey of trends and methodologies. IEEE Transactions on Industrial Informatics, 16(1), 4–17.
<https://doi.org/10.1109/TII.2019.2945004>
- Yang, L., & Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing, 415, 295–316.
<https://doi.org/10.1016/j.neucom.2020.07.061>
- Zhang, J. M., Harman, M., Ma, L., & Liu, Y. (2020). Machine learning testing: Survey, landscapes and horizons. IEEE Transactions on Software Engineering, 48(1), 1–36.
<https://doi.org/10.1109/TSE.2019.2962027>