SvedbergOpen
DISSEMINATION OF KNOWLEDGE
International Journal of Artificial Intelligence and Machine Learning
Publisher's Home Page: https://www.svedbergopen.com/
Research Paper Open Access

A Governance Framework For Trustworthy AI-Generated Data Engineering Artifacts In Cloud Environments

Pavan Kumar Kodati

Independent Researcher, USA ORCID ID: 0009-0000-5372-1504

Abstract
AI systems are increasingly capable of generating data engineering artifacts, transformation logic, orchestration workflows, quality rules, and documentation at speeds that outpace traditional review processes. This productivity gain introduces risks related to semantic correctness, policy compliance, provenance, security, and regulatory accountability that existing controls for the software development lifecycle were not designed to address. This article proposes a governance framework for trustworthy AI-generated data engineering artifacts in cloud environments. The framework defines six control domains: provenance tracking, policy-aware generation, technical validation, human review, lifecycle management, and audit reporting. Together these domains ensure that AI-assisted engineering accelerates delivery without compromising enterprise trust or compliance obligations. The paper presents a governance problem analysis identifying four distinct risk categories, describes the framework architecture and implementation strategy, defines evaluation criteria, and outlines enterprise adoption implications. The central argument is that governance is not a constraint on AI adoption; it is the mechanism that makes AI adoption sustainable at the artifact volumes, organizational scales, and regulatory expectations that meaningful productivity gains require.

Keywords: AI-generated artifacts, audit readiness, cloud data engineering, data governance, large language models, policy compliance, provenance tracking, trustworthy AI.

1. Introduction

AI systems now generate data engineering artifacts, structured query language (SQL) transformations, extract-transform-load (ETL) scripts, orchestration directed acyclic graph (DAG) definitions, data quality rules, metadata mappings, and technical documentation at a scale and velocity that manual construction cannot match. The productivity case is real. Teams quickly create prototypes for pipeline logic, reduce repetitive coding effort, and onboard new data sources more efficiently than was practical before generative models became capable enough for engineering tasks. What rarely receives the same attention is the question that follows: how should enterprises govern these outputs (Amershi et al., 2019)?

Traditional software development lifecycle controls assume human authorship. Code review, approval workflows, audit trails, and compliance evidence generation were designed for artifacts that engineers write deliberately, with domain knowledge applied at every step. AI-generated artifacts challenge these assumptions structurally. A transformation may be syntactically valid but logically incorrect, producing plausible-looking outputs from subtly wrong logic. A generated orchestration workflow can conflict with naming conventions, retry policies, or scheduling norms that have not been completely captured in the prompt provided to the generation model. Possibilities for insecure access or revealing potentially sensitive data structures by means of AI language models' logging data can be considered side effects of the use of such models (Pearce et al., 2022). The issue does not represent something rare; along with the increased number of code generators used for software development, the rate at which AI-generated data is put to use without further examination is too fast

for anyone to manually go over each piece. The situation is an inevitable result of the current state of affairs where efficient generation of AI requires timelessness, while efficient reviewing takes time. A failure to develop a specific solution results in an unavoidable dilemma: putting AI-generated code into production unexamined or losing efficiency (Paleyes et al., 2022).

This article proposes a governance framework that resolves this tension. Six control domains—provenance tracking, policy-aware generation, technical validation, human review, lifecycle management, and audit reporting—operate in concert across the artifact lifecycle. A risk tier model calibrates the depth of each control to the risk profile of the artifact. High-risk artifacts receive full validation and multi-reviewer approval. Low-risk artifacts move faster with lighter controls, and some are eligible for automated approval after passing validation. The framework is designed for cloud-native data engineering environments and integrates with continuous integration/continuous delivery (CI/CD) pipelines, artifact repositories, and existing compliance workflows (Ashmore et al., 2021).

The contribution is fourfold: a governance problem analysis characterizing four distinct risk categories for AI-generated engineering artifacts; a six-domain governance framework with defined control points across the artifact lifecycle; a risk tier model that calibrates governance depth to artifact impact; and an evaluation framework for measuring governance effectiveness in enterprise deployments. The article proceeds as follows: Section 2 analyzes the governance problem; Section 3 presents the proposed framework; Section 4 describes the implementation strategy; Section 5 defines evaluation criteria; Chapter 6 discusses implications and limitations; Section 7 concludes.

2. Governance Problem Analysis

Trustworthy AI adoption requires more than model accuracy; it requires organizational traceability. Enterprises must be able to answer: what artifact was generated, from which inputs, by which model version, under which organizational policies, and with which reviewer approval, before that artifact entered production. In regulated sectors, financial services, healthcare, and critical infrastructure, this traceability is not optional. Audit cycles, regulatory examinations, and incident investigations depend on it. The absence of provenance records for AI-generated artifacts is not a minor gap; it is a structural compliance exposure (Avin et al., 2021).

Four risk categories require specific governance responses. Semantic correctness failures occur when generated artifacts pass syntax validation but contain logically incorrect transformation logic, wrong aggregation grain, misaligned join conditions, or quality rule thresholds that do not reflect business intent. The danger here is not that the error is obvious; it is that the output looks correct. Standards non-compliance failures occur when generated artifacts violate organizational conventions for naming, exception handling, logging patterns, or performance constraints not fully encoded in the generation prompt. Security exposure failures occur when generated code embeds insecure access patterns, exposes sensitive field names in logging, or incorporates deprecated libraries with known vulnerabilities. Documentation drift failures occur when generated documentation is not version-linked to the artifact it describes, producing a growing divergence between what documentation claims and what deployed code does (Sculley et al., 2017).

Table 1. Risk taxonomy for AI-generated data engineering artifacts showing risk type, description, example failure, and governance control required.

Risk Type	Description	Example Artifact Failure	Governance Control Required
Semantic correctness	Syntactically valid but logically incorrect transformation logic	Revenue aggregation uses wrong grain, producing inflated metrics that pass automated tests	Dynamic validation with sample data execution and business output comparison
Standards non-compliance	Violation of naming, logging, retry, or performance conventions	Generated DAG uses non-standard task naming, breaking centralized monitoring and alerting	Static linting and policy scanning against organizational rule sets

Risk Type	Description	Example Artifact Failure	Governance Control Required
Security exposure	Insecure access patterns, sensitive data in logs, deprecated libraries	Customer identifiers logged in plain text in generated ETL exception handler	Security-focused static analysis and prohibited pattern scanning
Documentation drift	Documentation not version-linked to deployed artifact	Documentation describes v1 transformation logic after v3 has been deployed to production	Lifecycle management enforcing documentation-artifact version linking at each promotion

Existing software development processes are insufficient for this artifact class for three structural reasons. First, they assume manual authorship and do not account for the volume and velocity of machine generation; reviewing every generated artifact at the depth applied to handwritten code eliminates the productivity benefit. Second, they rely on reviewer domain expertise to surface semantic errors, but the fluency of language model outputs makes semantic review harder; generated code can look authoritative even when wrong. Moreover, evidence collection does not consider artifacts that have probabilistic provenance—those in which similar prompts may yield dissimilar output each time, as well as those in which the version of the model used is equally important to include in the audit record as the authorship information (Bender et al., 2021).

This much is evident to any enterprise: AI artifacts must be treated as a unique class of assets that require unique lifecycle management tailored to their inherent risks. Treating AI artifacts the same as code written by humans and reviewing all types of artifacts in the same way will lead to bottlenecks or result in merely rubber-stamping artifacts without truly understanding their contents (Lwakatare et al., 2019).

3. Proposed Governance Framework

The proposed framework defines six control domains: provenance tracking, policy-aware generation, technical validation, human review, lifecycle management, and audit reporting. The domains are not sequential stages; they operate in parallel and interact throughout the artifact lifecycle. Provenance records are created at generation and updated at each lifecycle transition. Policy constraints are applied before generation begins and re-evaluated at validation. Audit reporting draws on all other domains continuously rather than at a single compliance checkpoint (Mitchell et al., 2019).

Provenance tracking records every generation event with sufficient detail to reconstruct the context of the output. A complete provenance record captures the metadata inputs provided to the model, the prompt template version, the model identifier and version, the generation timestamp, the identity of the user or service that initiated the request, and the identifier of every artifact version produced. This is not documentation overhead; it is the evidence base that makes every other governance function possible. When there is no provenance, then validation results cannot be mapped to any specific behavior of the model, review actions cannot be associated with artifact statuses, and audit traces cannot be gathered except through rework by hand (Gebru et al., 2021).

Constraint enforcement is built into the generation process before model invocation occurs. Templates include company constraints, permitted programming languages, naming rules, approved libraries, logging policies, disallowed operations, and security policies. Wrappers enforce these policies by preventing generation if it would violate the policy configuration before the generation call reaches the model. Guards prevent generation outputs with banned patterns from being uploaded to the repository. The goal is to shift the compliance burden upstream where it can be stopped most cheaply (Ouyang et al., 2022).

Technical validation is executed using automated tests at two levels. These include static tests, such as syntax validation, organizational rule set linting, dependency analysis, schema matching verification, and security policy pattern detection. Dynamic tests entail running the created artifacts against sample or synthetic data sets in sandboxed environments to verify that the actual results obtained match the expected results obtained from metadata, as well as validating that data quality policies deliver coverage and precision as defined by the metadata. Together, they will cover not only structural problems found through syntax validation but also semantic issues, which are often caught only during testing (Mäkinen et al., 2021).

Human review and lifecycle management operate in tandem. The reviewer interface presents each artifact alongside its complete provenance record, source metadata, prompt lineage, static and dynamic validation results, and computed risk tier with the factors that determined it. This structured evidence reduces the time reviewers spend assembling context and increases the proportion of review time spent on substantive assessment. Review decisions, comments, and any edits are stored as part of the immutable artifact history. The lifecycle process provides guidelines for the state transitions, including draft, approved, deployed, superseded, and retirement, with each step based on the approval of the reviewed version, thus ensuring that no unapproved versions get into production (Jobin et al., 2019).

Audit reporting completes the governance cycle by making all the outputs of the control domains available for review by regulators and incident investigations. The audit logs for all the artifacts will contain details such as the provenance chain; validation results, including timestamp and rule references; reviewers along with their decisions; exception decisions; justifications; and deployment history. Both point-in-time queries and longitudinal analyses are supported. This reporting capability transforms governance from a process constraint, something that slows engineering, into a compliance asset that reduces the cost and effort of audit cycles (Ashmore et al., 2021).

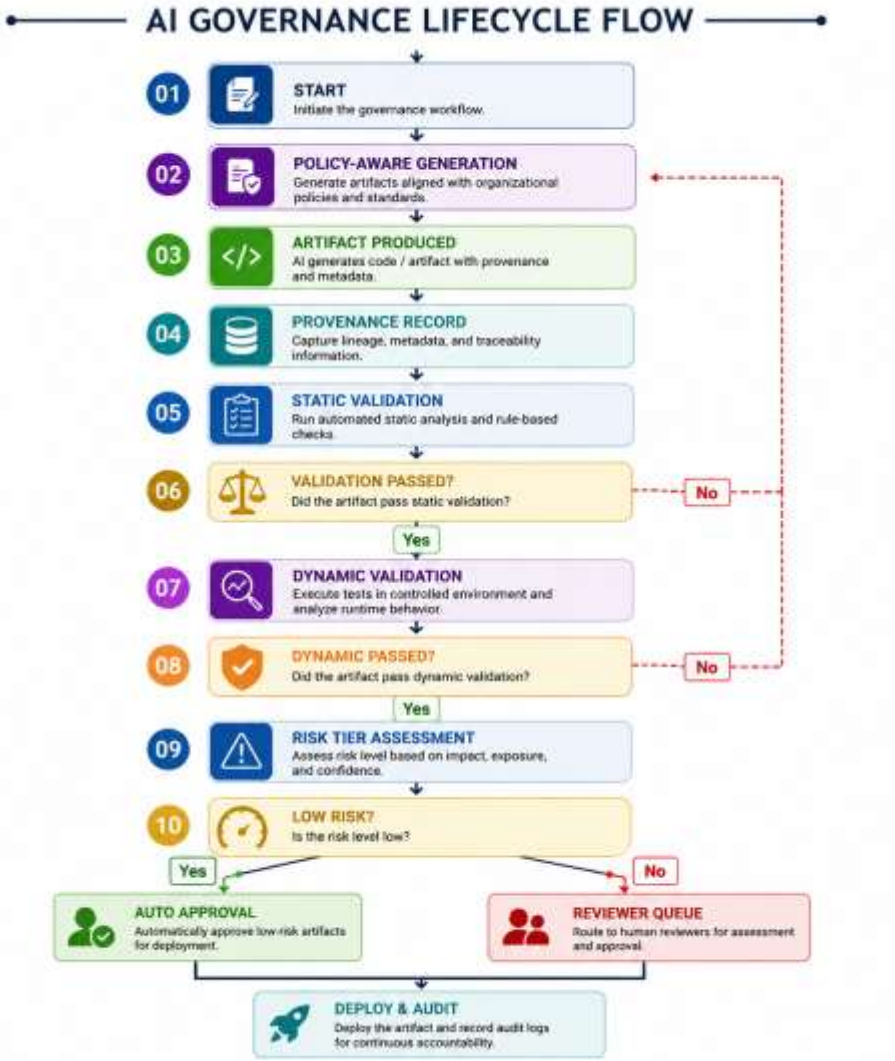

Figure 1. Proposed governance framework architecture and artifact lifecycle flow across six control domains.

4. Implementation Strategy

A practical implementation integrates the governance framework into existing CI/CD pipelines and data platform workflows rather than constructing a parallel governance system. Generated artifacts are committed to the same version-controlled repository used for manually authored code. The governance service intercepts each commit, attaches provenance metadata, triggers the validation pipeline, routes the artifact to the appropriate reviewer queue based on computed risk tier, records review outcomes, and updates artifact state at each lifecycle transition. Integration with existing tooling is not a convenience; it is a prerequisite for adoption. Under delivery pressure, engineers bypass governance systems that require them to use separate tools or duplicate workflows (Amershi et al., 2019).

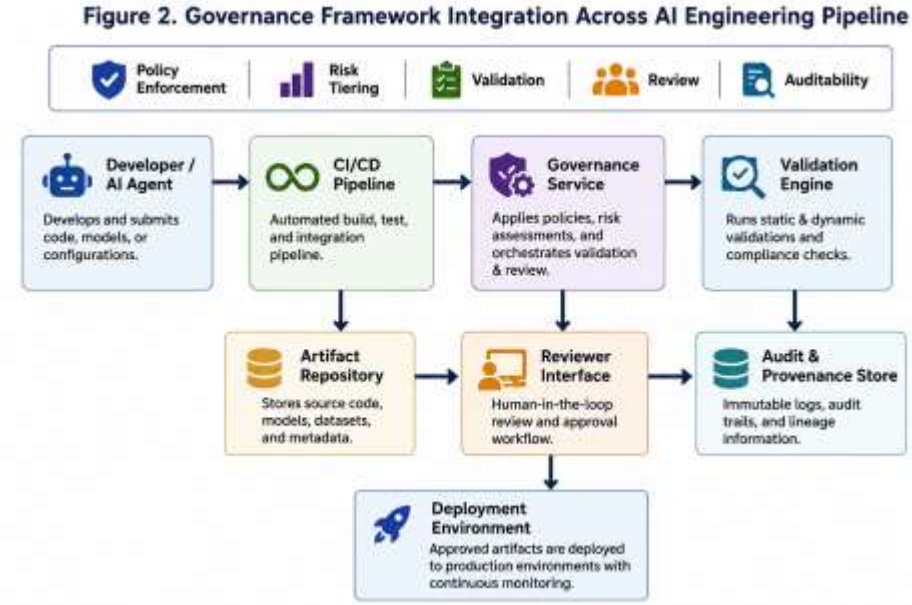

Figure 2. Governance service integration with existing CI/CD and data platform infrastructure.

4.1 Core Technical Components

The implementation is realized through seven interdependent components that collectively operationalize the six control domains described in Section 3. Each component has a defined technical boundary, specific inputs and outputs, and integration points with adjacent components.

The Governance API is the single ingestion point for all artifact submissions. It exposes REST endpoints that accept artifact payloads together with structured generation metadata, including the model identifier, prompt template version, initiating user or service identity, and timestamp. Upon receipt, the API assigns a globally unique artifact identifier, constructs an initial provenance record, publishes the artifact to an internal message bus for downstream processing by the validation and risk scoring engines, and returns a submission receipt with the assigned identifier and current processing status. All inter-component communication is authenticated using service account tokens scoped to minimum required permissions, and every API transaction is written to the audit store before the acknowledgment response is returned to the caller.

The Policy Engine enforces organizational constraints across two invocation points. At the pre-generation stage, it evaluates prompt templates and parameter overrides against a versioned rule set that encodes naming conventions, approved libraries, prohibited operations, mandatory logging patterns, access control requirements, and data handling restrictions. Rules that evaluate to a violation block generation and return a structured rejection message identifying the violated policies. At the post-generation stage, the policy engine scans generated artifact content for prohibited patterns, including hardcoded credentials, disallowed function calls, and non-compliant identifier formats. The rule set is maintained in a configuration repository with its own version history, enabling policy changes to be audited and rolled back independently of artifact changes.

The Validation Engine executes a two-stage test suite against each submitted artifact. Static validation applies syntax parsers, organizational linting rule sets, dependency graph analysis, schema conformance checks, and security-focused pattern detection. Each check produces a structured finding record containing the rule identifier, severity classification, affected line or element reference, and remediation guidance. Dynamic validation executes the artifact in an isolated sandbox environment provisioned with synthetic or anonymized

sample data representative of the target production data profile. For SQL and ETL artifacts, execution output is compared against expected result sets derived from source metadata, verifying row counts, column cardinality, aggregation grain, and business rule application. For orchestration artifacts, dynamic validation tests retry behavior under simulated failure conditions, verifies alert routing configuration, and confirms that task dependency declarations match the actual execution dependency graph. Validation results are signed with a timestamp and the validator service identity before transmission to the audit store.

The Risk Scoring Engine consumes the provenance record and validation findings for each artifact and produces a tier assignment of low, medium, or high. Tier assignment is computed from a weighted scoring function that evaluates five factors: artifact type, target deployment environment, data sensitivity classification of referenced data assets, count of downstream consumers declared in the pipeline dependency graph, and regulatory scope of the owning data domain. Weights are configurable and subject to empirical recalibration as the organization accumulates post-deployment defect data. The scoring engine exposes its weighting factors and their individual contributions alongside the final tier assignment, making tier decisions interpretable and auditable. Override requests, where a reviewer disputes the computed tier, are recorded in the audit store with the overriding user identity and a mandatory justification field.

The Reviewer Portal provides the human review interface for medium and high-tier artifacts. For each artifact pending review, the portal renders a structured evidence view containing the artifact content with inline validation annotations, the complete provenance chain showing generation context and model version, the prompt template and any applied parameter overrides, static and dynamic validation findings ranked by severity, security scan results, and the computed risk tier with factor weights. The portal enforces role-based access, routing artifacts to reviewer queues based on the qualifications declared in the reviewer registry, and requires multi-party approval for high-tier artifacts with at least one security-qualified sign-off. All review decisions, inline comments, and any content edits are recorded as immutable entries in the artifact history. The portal does not allow an artifact to transition to the approved state unless all required validation checks have passed and all required reviewer sign-offs have been recorded.

The Audit Store maintains the immutable record of every governance event across the artifact lifecycle. It stores provenance records, policy evaluation results, validation findings with full output logs, risk tier assignments with factor contributions, reviewer decisions with timestamps and identities, exception approvals with justifications, and deployment history entries. Records are written using an append-only mechanism; no record can be modified or deleted after it is written. The store supports two query modes: point-in-time queries that reconstruct the complete governance history of a specific artifact version as of a specified timestamp, and longitudinal queries that aggregate governance metrics across artifact populations over a defined time window. These query capabilities are the technical basis for the audit evidence packages produced during regulatory examination cycles.

CI/CD Integration connects the governance framework to the artifact promotion pipeline without requiring engineers to adopt separate tooling. The integration layer is implemented as pipeline stage hooks that invoke the Governance API at the artifact commit stage, poll for validation and risk tier status, enforce a governance gate that blocks promotion to the next pipeline stage until the artifact has reached the approved lifecycle state, and record the deployment event in the audit store upon successful promotion. Gate behavior is configurable by tier: low-tier artifacts that have passed all static validation checks and carry no policy violations are promoted automatically without human review; medium-tier artifacts are held at the gate until single-reviewer approval is recorded; high-tier artifacts are held until both required sign-offs are present. Artifacts that fail validation or receive rejection decisions are returned to the draft state with a structured rejection report delivered to the originating engineer through the existing pipeline notification channel.

4.2 Artifact Type Validation Procedures

The validation process differs depending on the types of artifact involved. In case of SQL and ETL artifacts, static validation is performed using parsers and linting tools that have been pre-configured with company rules related to naming, prohibited activities, mandatory logging standards, and performance constraints. Dynamic validation entails running the generated SQL on a sample data set and comparing the results obtained to the expected figures from source metadata with respect to row and column counts and business rules passed. In orchestration artifact validation, testing checks are conducted to verify the proper configuration of the retry policy, alert routes, task dependencies, and compliance with scheduling standards.

The reviewer interface is designed around structured evidence rather than raw artifact inspection. For each artifact requiring review, the interface presents: the artifact content with syntax highlighting and inline validation annotations; the provenance chain showing generation context and model version; the prompt

template and any parameter overrides applied; static validation results with rule references; dynamic validation results with sample input and output comparison; security scan findings ranked by severity; and the computed risk tier with the weighting factors. This design targets a measurable reduction in the time between artifact receipt and review decision and a measurable increase in reviewer confidence at the decision point (Paleyes et al., 2022).

The risk tier model is what makes governance scalable. Every generated artifact is assigned a tier—low, medium, or high—based on a weighted assessment of artifact type, target environment, data sensitivity classification, downstream consumer count, and regulatory scope. Low-tier artifacts, such as descriptive documentation and boilerplate configuration templates for non-production environments, are eligible for automated approval after passing static and dynamic validation without human review. Medium-tier artifacts require a single qualified reviewer. High-tier artifacts—those affecting regulated reporting pipelines, customer data transformations, or production environments with broad downstream dependencies—require full validation and approval from at least two qualified reviewers before promotion to deployed state (Sculley et al., 2017).

Table 2. The risk tier model includes the following elements: tier, artifact examples, validation requirements, approval requirements, and auto-approve eligibility.

Risk Tier	Artifact Examples	Validation Required	Approval Requirement	Auto-Approve Eligible
Low	Descriptive documentation, boilerplate config templates, non-production quality checks	Static validation only	None, automated approval after validation pass	Yes, after full static validation pass
Medium	Development environment SQL, non-regulated quality rules, internal reporting transformations	Static and dynamic validation	Single qualified reviewer	No, human review required
High	Production transformations, regulated reporting pipelines, customer data ETL, PII-touching scripts	Full validation suite including security scan	Two qualified reviewers, one with security sign-off	No, it always requires human approval

5. Evaluation Criteria

The governance process must be evaluated on the basis of five main factors. Defect prevention ratio is defined as the proportion of errors in semantics, standards adherence, and security that are detected and rectified before implementation. This metric is determined by comparing the results achieved through governance of AI-generated artifacts with those realized without governance, using AI-generated artifacts from the same generation pipeline as a baseline. Review efficiency metrics are captured in terms of the average time taken for a reviewed artifact to reach an approved state and the reviewer work effort per artifact. Provenance completeness measures the fraction of generation events for which a complete metadata record exists. The policy compliance rate assesses the fraction of generated artifacts that pass all policy checks on the first submission, a measure that improves as policy-aware generation constraints are refined. Audit readiness measures the time required to produce complete audit evidence for a sampled artifact, comparing the governed framework against manual log reconstruction (Paleyes et al., 2022).

Table 3. Evaluation metrics with measurement method, baseline approach, and expected improvement direction.

Metric	Measurement Method	Baseline	Expected Improvement
Defect prevention rate	Fraction of errors caught before deployment vs. total errors found	Ungoverned AI artifacts: errors found post-deployment	Substantial increase in pre-deployment detection rate
Review efficiency	Mean time from generation to approved state; reviewer effort per artifact	Unstructured manual review without evidence interface	Reduction in review time due to structured evidence presentation
Provenance completeness	Fraction of generation events with complete metadata capture	No systematic provenance recording in place	Near-complete capture after governance service integration
Policy compliance rate	Fraction of artifacts passing all policy checks at first submission	Ad-hoc prompt engineering without embedded policy constraints	Higher first-pass compliance with policy-aware generation active
Audit readiness score	Time to produce complete audit evidence for a sampled artifact	Manual reconstruction from logs and version history	Significant reduction in evidence preparation time

Reviewer trust operates as a second-order metric that amplifies or undermines all primary dimensions. A governance framework that reviewers do not trust will be bypassed under delivery pressure, rendering every other metric irrelevant. Trust can be measured through the reviewer override rate, how often reviewers reverse an automated risk tier assignment, the reviewer-reported confidence level at decision time, and the rate at which governed artifacts subsequently generate post-deployment defects. If override rates are high and confidence levels are low, the evidence interface is not surfacing the right information, and recalibration is required (Mitchell et al., 2019).

Comparison between an ungoverned generation, static validation alone, a full governance framework, and manually produced documents with current review processes will be made as part of a comparative evaluation. The objective of comparative evaluation is to examine the contributions of each governance layer independently while accounting for the process costs incurred by implementing the entire framework. Based on existing studies, dynamic validation would have the highest marginal impact on defect prevention, while structured evidence would have the highest marginal impact on efficiency during the reviewing process (Xia et al., 2023b).

In terms of adoption by the enterprise, measures that would include the number of artifacts that are easily accepted without undergoing significant changes after implementing generation constraints using policy awareness, reduction in defects because of AI-based code, and the time required to produce compliance Evidence can be included before conducting an audit cycle. The organization must first gather such information to provide a basis for comparison (Ashmore et al., 2021).

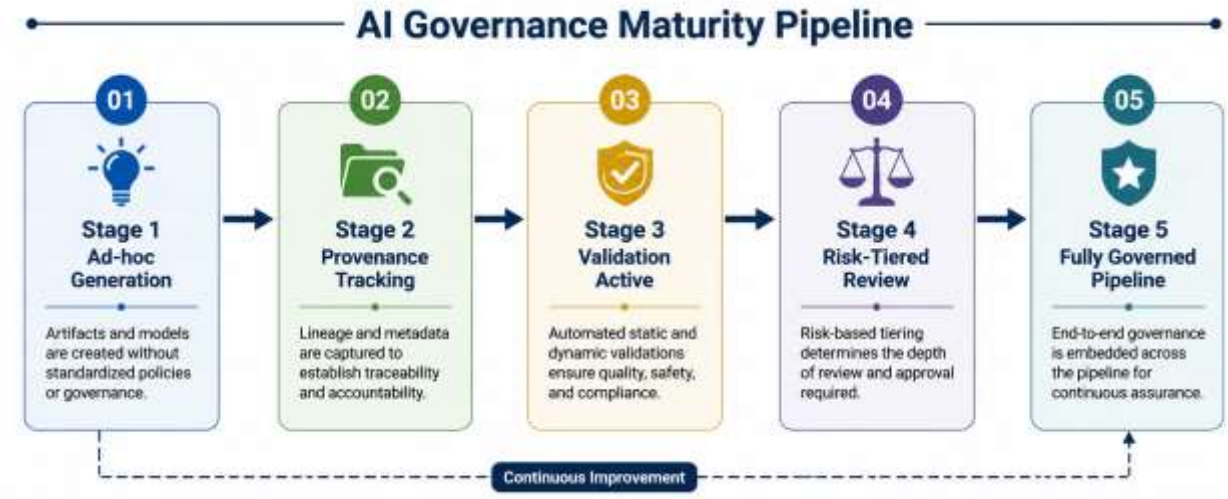

Figure 3. AI engineering governance maturity model: five stages from ad-hoc AI artifact generation to fully governed pipeline integration.

6. Discussion

The strategic framing matters as much as the technical design. Organizations that position governance as a gate on AI adoption will miscalibrate their controls, adding overhead without adding value, and ultimately undermining the adoption they intend to support. The more productive framing treats governance as the mechanism that makes AI adoption sustainable. Not because regulation requires it, though in many sectors it does, but because the alternative, ungoverned AI outputs at production scale, creates a liability that compounds faster than the productivity gains it enabled (Jobin et al., 2019).

Three limitations require direct acknowledgment. First, governance adds process overhead that may slow initial adoption if controls are poorly calibrated to actual artifact risk. Risk tier boundaries should be treated as empirical hypotheses and adjusted as the organization accumulates data on which artifact types actually produce post-deployment defects. A tier assignment that routes too many artifacts to high-tier review will create the bottleneck the framework was designed to prevent. Second, the policy-aware generation layer depends on organizational policies being accurately and completely encoded in prompt templates and model guards. Policies that exist only in informal team knowledge cannot be enforced at the generation layer, and the gap between documented policy and enforced policy is a persistent organizational challenge that the framework surfaces but cannot eliminate. Third, model versioning creates provenance complexity when the underlying language model is updated mid-deployment. The same prompt may produce meaningfully different outputs from different model versions, and provenance records must capture the model version at sufficient granularity to support retrospective analysis of artifact behavior changes (Lwakatare et al., 2019).

Table 4. Limitation analysis showing limitation, root cause, mitigation approach, and residual risk.

Limitation	Root Cause	Mitigation Approach	Residual Risk
Governance overhead may slow adoption	Risk tier miscalibration routing low-risk artifacts to high-depth review	Empirical calibration of tier boundaries using post-deployment defect data; iterative adjustment	Adoption resistance if calibration lag is long or recalibration is politically contested
Policy encoding incompleteness	Informal team knowledge not translated into prompt templates and model guards	Governance service surfaces unenforced policies; structured policy inventory required as a prerequisite.	Systematic generation of non-compliant artifacts for all uncaptured policy rules

Limitation	Root Cause	Mitigation Approach	Residual Risk
Model version provenance complexity	Language model updates changing output characteristics for identical prompts	Model version captured in provenance record at artifact level; diff-based alerting on version transitions	Retrospective analysis gaps if model versioning is coarse-grained across update cycles
Reviewer trust and bypass risk	Evidence interface perceived as compliance overhead rather than genuine decision support	Reviewer feedback loop built into governance service; override tracking and periodic interface calibration	Framework bypassed informally under delivery pressure if perceived value is insufficient

Future directions extend the framework across three dimensions. Autonomous agents present the most significant extension challenge: when AI agents both generate and deploy artifacts without human initiation, the human review domain requires fundamental redesign around policy-gate automation and post-deployment monitoring rather than pre-deployment approval. Policy-aware model orchestration embeds governance constraints in the model selection layer, routing artifact generation requests to models chosen for their policy alignment profile rather than their general capability. Risk scoring on a continuous basis involves expanding on the static tier classification approach through the development of an algorithm for updating the risk score associated with each artifact whenever the consumption scope of said artifact shifts due to changes in context (Bender et al., 2021).

7. Conclusion

AI-generated data engineering artifacts can deliver major productivity value, and that value is not realized by slowing generation, restricting which teams can use AI tools, or requiring every output to pass through the same review depth as manually authored code. It is realized by building a governance framework calibrated to the actual risk profile of different artifact types, one that makes review efficient by surfacing structured evidence rather than raw outputs and one that produces audit-ready provenance records as a byproduct of normal engineering operations rather than a separate compliance activity.

The six control domains proposed in this article are not bureaucratic layers. They apply cost-reduction mechanisms at different points in the artifact lifecycle. Provenance tracking reduces the cost of audits. Policy-aware generation reduces the cost of review by shifting the compliance burden to the point of generation. Risk tiers reduce the cost of review by calibrating depth to impact. The reviewer interface reduces the cost of review by eliminating evidence assembly. Together, they make it possible to govern AI-generated artifacts at the volume that meaningful adoption requires, without creating the bottleneck that defeats the purpose.

As AI becomes embedded in data engineering workflows, governance will evolve from optional best practice to baseline expectation, the same progression that version control and automated testing followed in previous decades of software engineering maturation. Organizations that build these capabilities now will be better positioned to expand AI-assisted engineering responsibly, demonstrate compliance in regulated environments, and sustain the organizational trust that long-term AI adoption requires.

References

1. 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. In Proceedings of the IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (pp. 291-300). https://doi.org/10.1109/icse-seip.2019.00042
2. Ashmore, R., Calinescu, R., & Paterson, C. (2021). Assuring the machine learning lifecycle: Desiderata, methods, and challenges. ACM Computing Surveys, 54(5), 1-39. https://doi.org/10.1145/3453444
3. Avin, S., et al. (2021). Filling gaps in trustworthy development of AI. Science, 374(6573), 1327-1329. https://doi.org/10.1126/science.abi7176

4. Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? In Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (pp. 610-623). https://doi.org/10.1145/3442188.3445922
5. Gebru, T., Morgenstern, J., Vecchione, B., Vaughan, J. W., Wallach, H., Daumé, H., & Crawford, K. (2021). Datasheets for datasets. Communications of the ACM, 64(12), 86-92. https://doi.org/10.1145/3458723
6. Jobin, A., Ienca, M., & Vayena, E. (2019). The global landscape of AI ethics guidelines. Nature Machine Intelligence, 1(9), 389-399. https://doi.org/10.1038/s42256-019-0088-2
7. Lwakatare, L. E., Raj, A., Crnkovic, I., Bosch, J., & Olsson, H. H. (2019). A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In Lecture Notes in Computer Science (pp. 227-243). Springer. https://doi.org/10.1007/978-3-030-19034-7_14
8. Makinen, S., Skon, J. P., Varjonen, S., & Smed, J. (2021). Who needs MLOps: What data scientists seek to accomplish and how can MLOps help? In Proceedings of the IEEE/ACM 1st Workshop on AI Engineering (pp. 109-116). https://doi.org/10.1109/wain52551.2021.00024
9. Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D., & Gebru, T. (2019). Model cards for model reporting. In Proceedings of the Conference on Fairness, Accountability, and Transparency (pp. 220-229). https://doi.org/10.1145/3287560.3287596
10. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., et al. (2022). Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems, 35, 27730-27744. https://doi.org/10.52202/068431-2011
11. Paleyes, A., Urma, R. G., & Lawrence, N. D. (2022). Challenges in deploying machine learning: A survey of case studies. ACM Computing Surveys, 55(6), 1-29. https://doi.org/10.1145/3533378
12. Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2022). Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. In Proceedings of the IEEE Symposium on Security and Privacy (pp. 754-768). https://doi.org/10.1109/sp46214.2022.9833571
13. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J. F., & Dennison, D. (2017). The ML test score: A rubric for ML production readiness and technical debt reduction. In Proceedings of the IEEE International Conference on Big Data (pp. 1395-1404). https://doi.org/10.1109/bigdata.2017.8258038
14. Xia, C. S., Wei, Y., & Zhang, L. (2023a). Automated program repair in the era of large pre-trained language models. In Proceedings of the IEEE/ACM International Conference on Software Engineering (pp. 1615-1627). https://doi.org/10.1109/icse48619.2023.00129
15. Xia, C. S., & Zhang, L. (2023b). Copiloting the copilots: Fusing large language models with completion engines for automated program repair. In Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 172-184). https://doi.org/10.1145/3611643.3616271