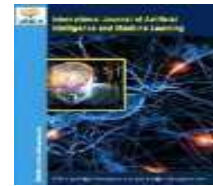




International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: <https://www.svedbergopen.com/>



Research Paper

Open Access

A Layered Governance Framework for AI-Enabled Pre-Silicon Validation Infrastructure

Abhishek Kumar Sinha

Independent Researcher, USA

Abstract

Validation organizations have already employed machine learning one decision at a time, whether a scheduling model here, a triage classifier there, or a compile-time predictor grafted onto an existing regression flow. Each new model brought a task-level gain. Such questions of composition, how an authorized party acts on the output of a model, and how a party comes to trust the output before taking a schedule-critical action are often unanswered. This article proposes that validation-infrastructure AI belongs in a single, multi-layered, platform architecture with an explicit orchestration and governance layer, between the model output and the action that gets executed, rather than as separate services adjacent to each of the many tools. Meanwhile, validation-pipeline choices are coupled in terms that a point solution cannot see: a partitioning choice shifts build time, which shifts debug fidelity, which shifts how much of a verification state space a program can credibly claim to have covered by tape-out. Drawing from MLOps literature, explainable-AI literature, and task-specialized EDA/ML literature, a six-layer reference architecture is composed for the validation pipeline: data plane, data-plane ingestion, feature and knowledge management, model service, orchestration and decision, engineer interface, and cross-cutting governance. Architectural analysis of the evaluation shows that guarantees of trustworthiness are required (more than in normal enterprise machine learning deployments) for irreversible, time-critical, tape-out-adjacent validation. These guarantees are motivated by the author's experience leading emulation infrastructure investment across four generations of flagship programs. The claim is almost a narrow one: the next real gain in validation productivity looks architectural, not like one more point-solution model.

Keywords

validation infrastructure, MLOps, model governance, orchestration layer, explainable AI, emulation, human-in-the-loop, trustworthy AI.

1. Introduction

A pre-silicon validation organization operates in time it cannot buy back. Emulation cycles, regression slots and debug hours are all vacuumed against a tape-out date that isn't movable for a scheduling algorithm and an hour spent on work that is later found to be unnecessary is an hour that downstream teams will never get back. Machine learning has found its way into nearly every stage of this pipeline: build prediction finishes before a compile even completes (Saidani et al., 2022), test case prioritization occurs in a limited execution budget (Pan et al., 2022), and even timing closure across process corners is predicted (Zhao et al., 2022). When a failure occurs, triage takes over from there (Wang et al., 2024; Papageorgiou et al., 2022). And all of those applications report a real task-level improvement on their own terms. None of them - not individually or in combination - answers the more basic organizational question of who or what has the authority to act when a recommendation from the model touches on a schedule-critical decision, nor why such authority is worth trusting.

This gap is not theoretical. The validation infrastructure at scale is a coupled system, not a shelf of tools. A partitioning decision leads to changes in build time of the emulation. Build time changes debug turnaround. Debug turnaround changes how much of the verification state space a program can honestly claim to have exercised before tape-out. A model optimized for partitioning only, with no regard for downstream build and debug impact, can return a locally-optimal recommendation that quietly degrades the risk position of the program overall. But point solutions - one model, one tool, one narrow objective - are much better at proving that machine learning works on a given validation subtask. They're much less informative about what happens when a series of such subtasks start interacting, or how much an organization should trust the model's output before doing something about it.

As this paper recommends, AI validation infrastructure should include a clear orchestration and governance layer between model output and subsequent action, rather than having separate ML services behind individual point tools. Task-level models are not the problem, and the EDA-specific machine learning literature shows their value (Nardi et al., 2020; Zhao et al., 2022). The problem is that task-level value does not compose into program-level value unless the architecture decides which model outputs are purely advisory, which may act

independently within a bounded space, and which outputs require a human validation architect in the loop before any schedule-affecting action is taken.

This article contributes a six-layer reference architecture for validation-infrastructure AI (data plane, ingestion layer, feature and knowledge management layer, model service model, orchestration and decision layer, engineer interface layer), which is different from previous reference architectures in how trustworthiness requirements differ for validation versus for enterprise usage. An enterprise recommender can tolerate a bad prediction, and the user sees the poor recommendation once and moves on. Validation-infrastructure models that misallocate emulation capacity days before a milestone or silently drift in triage accuracy across the lifecycle of a design are disruptive to a program schedule. The synthesis and the gap left in the literature are described in section 2. Section 3 presents the architecture, Section 4 shows how it would map into the validation workflow (Figure 2) and Section 5 further develops the trust and governance case. Section 6 outlines anticipated architectural change based on program-governance experience, and Section 7 offers a path to phased adoption. Section 8 concludes.

2. Literature Review

Three literatures are relevant to the problem, but all of them stop short of it.

Nardi et al. show how to use machine learning methods to optimize the compile and partitioning phase of FPGAs logic emulation systems. Zhao et al. build a machine learning model which predicts the timing at the process corners, enabling them to sample a representative subset of the most critical corners and infer the rest. (This approach, as will be shown in later sections, enables faster timing closure without sacrificing the reliability needed in an industrial process.) Saidani et al. (2022) use deep learning for continuous integration build-failure prediction, trained on tens of thousands of historical builds collected from long-lived software systems. Pan et al. (2022) survey the broader space of machine learning-based test case selection and prioritization, while Papageorgiou et al. (2022) survey machine learning-based root cause analysis for zero-defect manufacturing. Extending this further, Kaveri and Lahande (2023) study infrastructure allocation in task scheduling and load balancing for cloud computing using reinforcement learning. This is analogous to emulation capacity scheduling, the main difference being the task of scheduling computation resources. As in the previous studies, the model is evaluated on its own task-level objective (accuracy, build time, test suite efficiency, etc.), without considering how these objectives should be controlled within an interdependent pipeline.

A second literature takes the composition question seriously, yet lacks even external validation. Sculley et al. (2015), for instance, categorize the mechanisms production ML systems use to amass technical debt as boundary erosion, feature-model entanglement, hidden feedback loops, and undeclared consumers of a model's output. It provides an analysis relevant to web services, but applicable to much wider software engineering practice. A case study of a large industrial project by Amershi et al. (2019) argues that software engineering practice for machine learning is quite different from legacy practice notably in data handling and the model lifecycle. That vision is formalized in the MLOps architecture of Kreuzberger et al. (2023), which separates out the data, experimentation, and production layers of a mature deployment. Two strands capture the rest of the literature. The first is documentation to capture how a model is intended to be used, and what its known limitations are, such as model cards (Mitchell et al., 2019). The second is a catalog of open questions to be addressed in assuring an ML system over its entire lifecycle (Paterson et al., 2021). All this is mature and genuinely useful for the governance question at hand, but it is not really directed to problems where a single critical misprediction may be either terminal or time-sensitive (like a tape-out commitment). It doesn't even begin to address the question of what happens if the governed system has been incorporated into a program, whose coverage claims underwrite an irreversible commitment to manufacturing.

A third literature directly addresses trustworthiness and explainability for high-consequence AI. Its roots are in DARPA's explainable AI program, which originated from a simple operational fact: in a defense context and autonomous systems, an operator cannot act on an opaque recommendation without some account of why it was made (Gunning & Aha, 2019). A global survey of AI ethics guidelines found relative convergence around transparency, accountability, and human oversight, but very little about how to instantiate these principles (Jobin et al., 2019). The model drift literature that has since become the field's reference survey of the subject further confirms that a model's performance decays when the statistical properties of the training data diverge from those in the application environment: for a model trained on a design generation, this is equivalent to the validation model trained on one design generation and applied to the next (Gama et al., 2014). In turn, the actions oriented toward robustness, fairness, and explainability have been reviewed in a synthesis of the trustworthy-AI literature (Kaur et al., 2023). Finally, the practices of AI governance and the barriers faced by organizations in adopting them have been reviewed, first by Papagiannidis, Enholm, et al. (2023) and then by Papagiannidis, Mikalef, and Conboy (2025). This literature supplies the vocabulary a trustworthy system needs: human-in-the-loop review, drift monitoring, auditability. It rarely, if ever, encounters the risk profile

unique to semiconductor validation: an irreversible fabrication decision downstream of a coverage claim a model helped produce.

Together, these three literatures indicate a gap with a particular shape: task-level EDA and verification work shows machine learning can be applied on the individual validation subtask level, while MLOps work outlines how a machine learning system's lifecycle should be governed in general enterprise contexts. Trustworthy-AI research clarifies which structural properties a high-consequence system should have, but no framework currently integrates these three strands for validating infrastructure under a risk profile - irreversibility, schedule criticality, downstream silicon exposure - that general MLOps work does not contemplate and that trustworthy-AI research has not yet tested. This article's task is to close this integration gap, through properly contextualizing MLOps in trust- and safety-related terms.

3. Architectural Framework

Validation-infrastructure AI is a six-layer stack, with each layer having a separate responsibility, the output of each layer being the governed input to the next. Because the second claim, that even isolated point solutions cannot govern interdependent decisions, only holds if all the layers are read together, the stack cannot be considered as a disjoint set of solutions.

The lowest layer is the data plane which contains all of the telemetry from the validation estate: emulator utilization logs, regression pass/fail logs, waveforms, debug artifacts, and compilation logs, as well as process health signals. Its job is capture, retention, and lineage, not analysis: an organization that cannot reconstruct which data trained or fed a given model decision has already given up a large share of its ability to audit that decision later, a concern the model-documentation literature treats as foundational (Mitchell et al., 2019).

Ingestion sits above it, normalizing heterogeneous telemetry - different emulator vendors, different regression frameworks, different design generations - into a representation the layers above can consume without resolving the parsing problem independently. This is where schema drift across tool versions gets caught before it silently corrupts a feature computed three layers up.

The feature and knowledge layer is where telemetry from ingestion is transformed into features consumed by the models, the geometry of a partition, the failure signature history of a build, the coverage contribution of a test, the degradation history of an emulator, and institutional knowledge such as previously classified root causes, known failure signatures, and secrets for particular generations that cannot be learned by data alone. Entanglement, in Sculley et al.'s sense, was most often the case where a feature engineered for one model, unbeknownst to the teams involved, had become necessary for another model, and a change for one was affecting the other.

The models (build-outcome predictors, timing predictors, triage classifiers, and scheduling optimizers) live in the model service layer, separate from the tools, as versioned, independently deployable model services. This separation from the tools allows for governance of the models to be possible. A model wired into a scheduler's codebase, for example, cannot be separately versioned, monitored, or rolled back from the scheduler itself.

This article positions the orchestration and decision layer as the gravity center of the whole architecture and the model output as the candidate action. It also checks the action against policy - is it easily reversible, does it touch a resource that is needed to run the schedule, is its confidence score high enough for this action class? It then either performs the action, schedules it to be validated by a human validation engineer later, or defers it until more data is collected. Cross-model dependencies are handled here too, for example a partitioning recommendation is assessed against its downstream build-time and debug-fidelity consequences, rather than executed unconditionally.

Engineer interface layer surfaces model outputs, and orchestration decisions made from it, to engineers and program leads who act on or override them. This is where human-in-the-loop governance either acts as a check or merely a rubber stamp. An interface that shows a recommendation without the evidence behind it invites exactly the unreflective approval the gate exists to prevent.

Cross-cutting governance spans all five other layers. Access control defines which model is allowed to act on which resources (or other services). The history of decision changes and overrides is recorded in an audit log, and drift monitoring determines whether the model deviates from its validation-time baseline (Gama et al., 2014). The versioning discipline connects actions to specific model versions, datasets, and other nested artifacts.

Figure 1. Six-Layer Reference Architecture for AI-Enabled Validation Infrastructure

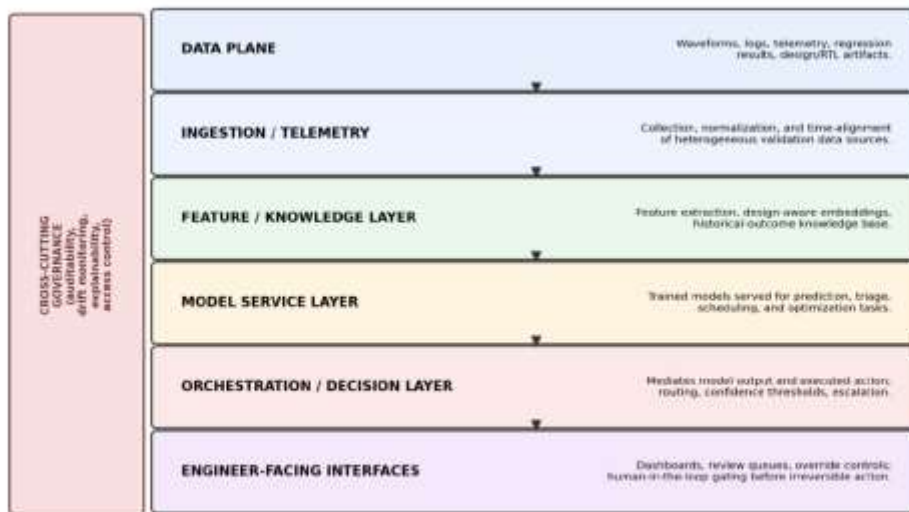


Figure 1 (referenced above; caption embedded in the image).

Table 1. Layer-by-layer function, inputs, outputs, and governance touchpoint in the six-layer reference architecture.

Layer	Function	Inputs	Outputs	Governance Touchpoint
Data plane	Holds raw validation artifacts as they are produced	Waveforms, logs, regression results, RTL/design artifacts	Structured, addressable data available to ingestion	Access control and provenance tracking for all downstream layers
Ingestion / telemetry	Collects, normalizes, and time-aligns heterogeneous sources	Raw data-plane artifacts across tools and formats	Normalized, time-aligned data streams	Data-quality checks; rejection of malformed or unattributable records
Feature / knowledge layer	Extracts design-aware features and maintains historical-outcome knowledge	Normalized telemetry; prior program outcomes	Feature vectors; queryable historical-outcome knowledge base	Versioning of feature definitions; traceability from feature back to source artifact
Model service layer	Serves trained models for prediction, triage, scheduling, optimization tasks	Feature vectors; task-specific query context	Model predictions with associated confidence signals	Model versioning; performance monitoring; drift detection
Orchestration / decision layer	Mediates between model output and executed action	Model predictions and confidence signals	Routed decisions: auto-act, escalate, or defer to engineer	Confidence thresholds; escalation policy; the primary point where governance is structurally enforced, not merely advisory

Layer	Function	Inputs	Outputs	Governance Touchpoint
Engineer-facing interfaces	Presents decisions and evidence for human review or override	Routed decisions and their supporting evidence	Engineer actions: approve, override, annotate	Human-in-the-loop gating before any irreversible or schedule-critical action

4. AI Integration Points Across the Validation Workflow

The framework earns its keep against this list of integration points already existing to some degree in real validation workflows, not as an abstract stack.

Build and compile optimization. A large SoC emulation build may run for hours, and a build that fails late in that process can waste resources already allocated to triage the build. Build outcome prediction models (Saidani et al., 2022) can reduce wasted compile cycles, but they work only if orchestration can act on these predictions before the budget expires, which is impossible in a point tool without visibility into queue-level scheduling.

Partitioning optimization. Partitioning a design across emulation hardware has implications beyond the impact of partitioning. A partition purely for build time might not allow debug observability, or a partition purely for debug observability might schedule builds longer than allowed (Nardi et al., 2020). This is the strongest example of the interdependence the article's argument relies on, and there is no model with only the partitioning objective that resolves this.

Resource allocation and scheduling. Emulation capacity is finite and costly. Scheduling policies determine which program gets to use emulation capacity, and when. Reinforcement learning scheduling in other domains for infrastructure (Kaveri & Lahande, 2023) suggests transfer learning, but when a scheduling decision is a function of many schedules' schedule risk, this is the exact class of decision orchestration is designed to gate rather than let run unilaterally.

Timing and performance prediction. Predicting timing closure across process corners without running full static timing analysis on each one can considerably compress a schedule. According to Zhao et al. (2022), it achieves 98.2% accuracy on their benchmark evaluation, which they say is an improvement of 15% over a previous state-of-the-art method they compare against. It does this by inferring a wide corner set from a small dominant subset and speeding up closure. The challenge to governance this raises is not whether the prediction is accurate on average; it is what happens to program risk on the one corner where it was not.

Debug instrumentation. Finally, applying ML to log representations (Gupta et al., 2023) and large language model-based incident interpretation (Wang et al., 2024) can point debug attention to the signals that are most likely to explain a failure, shortening the human investigation loop. That only helps if the interface layer can produce enough evidence that the advice was really checked and not just accepted on faith.

Failure triage and root-cause analysis. Automated triage (Papageorgiou et al., 2022; Wang et al., 2024) is among the more mature integration points for the workflow, and one of the most consequential to govern carefully. If the triage model misclassifies the systemic design error as a known benign signature, it can be effectively masked and coming back as a tape-out risk during production.

Infrastructure health management. The emulation estate produces telemetry data of its own, and anomaly detection of that telemetry (Vajda et al., 2024; Zhao et al., 2026) gives early warning of problems, before they cause an inconvenient failure in a program running on the emulation.

Table 2. AI integration points across the validation workflow, by decision type, risk class, and required governance control.

Integration Point	Decision Type	Risk Class	Required Governance Control
Build / compile optimization	Parameter selection for compile efficiency	Low: reversible, no correctness impact	Automated monitoring sufficient; periodic audit
Partitioning optimization	Design partitioning across emulation resources	Medium: affects downstream verification scope	Advisory recommendation with engineer confirmation before commit
Resource allocation / scheduling	Assignment of validation jobs to available infrastructure	Low-to-medium: reversible, but affects schedule throughput	Automated within bounded policy; escalation on repeated reallocation failures
Timing / performance prediction	Forecasting timing closure or performance outcomes	Medium: informs planning decisions, not sign-off	Predictions labeled as advisory; not usable as sign-off evidence without independent confirmation
Debug instrumentation	Selecting or prioritizing debug signals and probes	Medium: affects what evidence is available for triage	Engineer review of instrumentation choices before campaign execution
Failure triage / root-cause analysis	Candidate root-cause hypotheses for observed failures	High: informs schedule-critical bring-up decisions	Human-in-the-loop mandatory; model output treated as hypothesis, not conclusion
Infrastructure health management	Predictive maintenance and capacity-health signaling	Low: operational, not correctness-affecting	Automated alerting; engineer action only on sustained anomaly

5. Trustworthiness, Explainability, and Governance

Enterprise ML systems - recommendation systems, fraud classifiers, demand forecasters, etc - mostly operate in applications where bad predictions are either low-cost or recoverable (a bad recommendation shown for a short time, a fraud false positive caught and flagged). Validation infrastructure choices do not share this property. A scheduling decision that mismanaged the emulation capacity available for the last few weeks preceding a milestone will not be put right by simply rerunning the schedule, simply because the milestone is now in the past. A triage model silently drifting across a design generation, undetected until the final model claims coverage at the tape-out readiness review, has already contaminated a decision that is not cheap to undo. That is why validation-infrastructure AI needs a stricter trustworthiness standard than the one the enterprise ML literature (e.g., Kreuzberger et al., 2023; Paterson et al., 2021) generally assumes - even as that literature's tools remain necessary here.

In this architecture, human-in-the-loop gating is not a generic safety slogan, but a structural property of the orchestration layer. Some classes of action (for example, reallocating capacity across programs, closing out a coverage claim on a model's say-so, or deprioritizing a failure signature as already known) require explicit approval by a human engineer, irrespective of model confidence. Others, such as surfacing a triage suggestion or flagging a build for early review, may be ungated because their consequences are reversible within the schedule's tolerance.

Explainability follows a similar risk-based approach, and Gunning and Aha (2019) built a case for interpretability in the context of defense and autonomous systems, where an operator cannot act upon a black-box recommendation unless they understand the rationale. In validation engineering, that property is useful whenever a prediction from a model forms part of the basis of some decision, which later needs to be defended in a readiness review. If a triage classifier cannot point to a signal that led to the classification, it is not just inconvenient. Unfortunately, it is unauditible in exactly the setting where an audit may be most needed.

Drift monitoring avoids a hazard that while clearly signaled in the concept-drift literature (Gama et al., 2014), is structurally underestimated by validation organizations. Topological drift in partitions, timing drift in clocks, and signature drift in failure modes may not be straightforwardly generalizable from one design generation to the next. To assume they are, is to import a risk beyond monitoring into the program schedule. Cross-cutting governance (Section 3): For each of the models, the real-world performance should be continually compared to the baseline performance at validation time, not certified once and then forgotten.

Auditability binds the others together: The contents of model cards (Mitchell et al., 2019) and lifecycle assurance documents (Paterson et al., 2021) are not paperwork. They are how a program later reconstructs that validation decision was made, which model version made the decision, which data trained the model, and whether a person validated the decision. In cases where coverage implies an irreversible fabrication, the reconstruction process is particularly vital.

6. Results / Findings

It is an architectural analysis not an empirical experiment. It reports what the model predicts as the failure modes of a point-solution alternative from firsthand, program-governance experience, providing emulation infrastructure for four generations of flagship programs for an organization of 200+ engineers and an estimated emulation capital asset base on the order of \$1.2 billion, rather than a controlled experiment contrasting governed and ungoverned deployments. (That said, this is a single executive's cross-generation observation, and the framework should be read as such rather than a multi-organization study.)

Another failure mode is silent cross-model conflict: if the partitioning model, scheduling model, and triage model are all run as separate point solutions, the locally optimal choice of one model can silently work against another. To prevent this from happening, the orchestration layer is built in such a way that any choice that touches a shared resource must check it against the choice the model of another layer provided for that resource.

The second most common failure mode is drift amassing unmonitored into a schedule-critical surprise. A point-solution deployment typically has no standing mechanism comparing a model's current performance against its validation-time baseline; the model ships, and its accuracy gets revisited only once someone notices a problem. That said, cross-cutting governance drift monitoring is more a structural requirement than it is a discretionary habit, and across generations of design, model drift is the safer posture based on the author's experience. In each generation the workload characteristics have shifted enough that a model validated against the previous generation could not be assumed to generalize.

Unauditable authority, the third predicted failure mode, is when a model recommends an action that affects a schedule, but nobody knows which model, which version, which data set, or whether a human ever reviewed the recommendation. This is the failure mode which this article considers most likely to occur at the worst possible time: during a tape-out readiness review, when a program must defend a coverage claim that it cannot reconstruct. This risk can be reduced through a layered architecture where cross-cutting concerns of governance are not left to the decision of each point solution, which a collection of independent point tools is better positioned to do.

These three predicted failure modes bring this article's conclusion back to the perspective with which the introduction began. The defect of the point-solution pattern is not that the individual models do not work well: indeed, Section 2's review of task-level literature shows that they usually do, at least. No matter how accurate the task-level output, accuracy at this level does not, by itself, suffice to guarantee safety or accountability at the program level. An explicit orchestration and governance layer is needed to fill this gap.

7. Discussion - Operational Adoption and Maturity Roadmap

There are no organizations that would start implementing a six-layer governance architecture in one go, nor could a roadmap that pretended otherwise survive contact with a live validation program. The simplest way to start moving any organization forward is to deploy an instrumented baseline stage: build out the data plane and ingestion layers to the point where a validation program could reconstruct the validation cycle post-hoc. At this stage, no model output is yet allowed to drive action, because that is an unglamorous middle stage. There is no clear productivity gain. It is also a prerequisite for every governance mechanism in Section 5 since the program cannot audit decisions it never captured in the first place.

The second stage of advisory deployment is where the build predictions, triage suggestions, timing predictions are displayed to engineers but not acted upon. This is where the organization is also doing the calibration work to learn which model classes they trust well enough to actually take the next step, and where the engineers are developing the intuitions about model behavior that will make further autonomy tolerable rather than alarming.

Stage three is constrained autonomy. The orchestration layer may take some actions of its choosing, on certain classes of reversible actions, without a human gate (for example, auto-surfacing a triage classification). Meanwhile, schedule-critical and irreversible actions are never taken without a human gate, regardless of model confidence. This is where the core intuition of the framework manifests: autonomy increases, not by activating all of the model estate, but by activating action class by action class, risk-tier by risk-tier.

This roadmap is mapped to the themes of Section 2. This does not represent a one-to-one mapping, but rather reflects the nature of technology roadmap to be incremental, capturing the principles of the lifecycle maturity

discussed in the MLOps literature (Kreuzberger et al., 2023). In contrast, each step is a governance capability. Instrumented baseline should implement the auditability required in the trustworthy-AI literature (Kaur et al., 2023), before a model is trusted to act. The organizational adoption friction recorded in the AI-governance literature - reluctance to relinquish decision-making, ambiguous accountabilities, uneven executive sponsorship (Papagiannidis et al., 2023; Papagiannidis et al., 2025) - is, in the author's experience, sharpest when moving from advisory deployment to the restricted agency of initial inaction monitoring. That is exactly the transition where the orchestration layer's policy design meets organizational resistance, as opposed to buy-in in a slide deck.

8. Conclusion

The problem that opened this article is plain: validation organizations are using machine learning models to inform individual infrastructure decisions, but there is no architecture to govern how those decisions compose, who has the authority to act on those individual predictions, or how trust is established before a model touches a schedule-critical outcome. This article has argued throughout that better point solutions do not solve this - the task-level literature already shows individual models can be accurate - but an explicit orchestration and governance layer, mediating between model output and executed action, can, because validation-pipeline decisions are interdependent in ways no isolated tool represents on its own. The contribution offered in support is a six-layer reference architecture for Validation Infrastructure's irreversible, schedule-critical, silicon-risk-bearing consequences and a staged maturity roadmap, which does not ask an organization to leap all the way to full autonomy immediately.

The next real gain in validation productivity, on this account, will not come from one more point-solution model shaving a few more points off a narrow accuracy target. It will come from the governance architecture that lets a program trust, audit, and carefully extend autonomy to the models it already has, and, by doing so, turn scattered task-level gains into a program-level one. This article locates the next inflection of the field in that architecture.

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. *ICSE-SEIP 2019*, 291-300. <https://ieeexplore.ieee.org/document/8804457>
2. Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4), 1-37. <https://dl.acm.org/doi/10.1145/2523813>
3. Gunning, D., & Aha, D. W. (2019). DARPA's explainable artificial intelligence (XAI) program. *AI Magazine*, 40(2), 44-58. <https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2850>
4. Gupta, P., Kumar, H., Kar, D., Bhukar, K., Aggarwal, P., & Mohapatra, P. (2023). Learning representations on logs for AIOps. *IEEE CLOUD 2023*. <https://arxiv.org/abs/2308.11526>
5. Jobin, A., Ienca, M., & Vayena, E. (2019). The global landscape of AI ethics guidelines. *Nature Machine Intelligence*, 1(9), 389-399. <https://www.nature.com/articles/s42256-019-0088-2>
6. Kaur, D., Uslu, S., Rittichier, K. J., & Durrezi, A. (2023). Trustworthy artificial intelligence: A review. *ACM Computing Surveys*, 55(2). https://www.researchgate.net/publication/366788513_Trustworthy_Artificial_Intelligence_A_Review
7. Kaveri, P. R., & Lahande, P. (2023). Reinforcement learning to improve resource scheduling and load balancing in cloud computing. *SN Computer Science*, 4(2). <https://link.springer.com/article/10.1007/s42979-022-01609-9>
8. Kreuzberger, D., Kuhl, N., & Hirschl, S. (2023). Machine learning operations (MLOps): Overview, definition, and architecture. *IEEE Access*, 11, 31866-31879. <https://ieeexplore.ieee.org/document/10081336>
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. *FAT '19*, 220-229. <https://dl.acm.org/doi/10.1145/3287560.3287596>
10. Nardi, A., Armato, A., Balasubramanian, S., Bergman, E., & Rosen, S. (2020). Improving FPGA-based logic emulation systems through machine learning. *ACM TODAES*, 25(6). <https://gtcad.gatech.edu/www/papers/3399595.pdf>
11. Pan, R., Bagherzadeh, M., Ghaleb, T. A., & Briand, L. (2022). Test case selection and prioritization using machine learning: A systematic literature review. *Empirical Software Engineering*, 27(2). <https://link.springer.com/article/10.1007/s10664-021-10066-6>

12. Papageorgiou, K., Theodosiou, T., Rapti, A., Papageorgiou, E. I., Dimitriou, N., Tzovaras, D., & Margetis, G. (2022). A systematic review on machine learning methods for root cause analysis towards zero-defect manufacturing. *Frontiers in Manufacturing Technology*, 2.
https://www.researchgate.net/publication/364818190_A_systematic_review_on_machine_learning_methods_for_root_cause_analysis_towards_zero-defect_manufacturing
13. Papagiannidis, E., Enholm, I. M., Dremel, C., Mikalef, P., & Krogstie, J. (2023). Toward AI governance: Identifying best practices and potential barriers and outcomes. *Information Systems Frontiers*, 25(1), 123-141.
https://www.researchgate.net/publication/358511425_Toward_AI_Governance_Identifying_Best_Practices_and_Potential_Barriers_and_Outcomes
14. Papagiannidis, E., Mikalef, P., & Conboy, K. (2025). Responsible artificial intelligence governance: A review and research framework. *Journal of Strategic Information Systems*, 34(2).
<https://www.sciencedirect.com/science/article/pii/S0963868724000672>
15. Paterson, C., Calinescu, R., & Ashmore, R. (2021). Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *ACM Computing Surveys*, 54(5). <https://dl.acm.org/doi/10.1145/3453444>
16. Saidani, I., Ouni, A., & Mkaouer, M. W. (2022). Improving the prediction of continuous integration build failures using deep learning. *Automated Software Engineering*, 29(1).
https://www.researchgate.net/publication/357990835_Improving_the_prediction_of_continuous_integration_build_failures_using_deep_learning
17. 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. *NeurIPS* 28, 2503-2511.
https://proceedings.neurips.cc/paper_files/paper/2015/file/86df7dcfd896fcfa2674f757a2463eba-Paper.pdf
18. Vajda, D. L., Do, T. V., Berczes, T., & Farkas, K. (2024). Machine learning-based real-time anomaly detection using data pre-processing in the telemetry of server farms. *Scientific Reports*, 14.
<https://www.nature.com/articles/s41598-024-72982-z>
19. Wang, Z., Li, J., Ma, M., Li, Z., Kang, Y., Zhang, C., Bansal, C., Chintalapati, M., Rajmohan, S., Lin, Q., Zhang, D., Pei, C., & Xie, G. (2024). Large language models can provide accurate and interpretable incident triage. *IEEE ISSRE 2024*. <https://ieeexplore.ieee.org/document/10771420>
20. Zhao, X., Sun, D., Yao, Y., & Li, Y. (2026). Fault diagnosis and health monitoring method for semiconductor manufacturing equipment based on deep learning and subspace transfer. *Machines*, 14(2). <https://www.mdpi.com/2075-1702/14/2/176>
21. Zhao, Z., Zhang, S., Liu, G., Feng, C., Yang, T., Han, A., & Wang, L. (2022). Machine-learning-based multi-corner timing prediction for faster timing closure. *Electronics*, 11(10).
<https://www.mdpi.com/2079-9292/11/10/1571>