Multi-Agent Debate for Software Architecture Governance: A Framework for ADR Generation, Risk Review, and Deployment Readiness

Veera Ravindra Divi
Independent Researcher
Austin, Texas, USA

Abstract— Architecture review at most organizations is a serial, expert-bound activity: a small number of senior engineers read a design, apply tacit heuristics, and produce decisions whose rationale is rarely captured in a structured, auditable form. Single large language models (LLMs) have been proposed to assist this work, but a single model that both proposes and reviews a design may reinforce assumptions from its own proposal, agree with itself, and present trade-offs as settled rather than contested. We propose **MAD-Arch** (Multi-Agent Debate for Architecture), a role-partitioned, human-in-the-loop framework for software architecture *governance* rather than mere critique. MAD-Arch separates design generation from design criticism: a proposal agent produces a baseline, a panel of adversarial reviewer roles—scalability, security, reliability, delivery, and cost—independently critiques it, the roles debate their disagreements over bounded rounds, and an arbiter consolidates the exchange into structured, version-controlled governance artifacts—Architecture Decision Records (ADRs), a security and compliance risk register, an operational-readiness checklist, and deployment-governance recommendations—together with a final recommendation carrying a transparent confidence score and an explicit list of unresolved assumptions. A human architect remains the accountable decision-maker at a mandatory approval gate, and the artifacts are designed to integrate with pull requests and CI/CD gates. We contribute (i) the generation–criticism separation as an organizing principle, (ii) a reproducible debate workflow, (iii) a rubric-based arbiter confidence model, and (iv) a conceptual evaluation protocol over three reference scenarios—a cloud-native order-processing system, an event-driven IoT telemetry platform, and an AI-powered document-processing pipeline—that compares MAD-Arch against single-model and checklist-only human review using precision/recall-style risk scoring, trade-off coverage, artifact completeness, and a redundant-recommendation rate. We report *illustrative* pilot-style values that demonstrate the protocol rather than prove effectiveness; we conducted no production deployment, and empirical validation remains future work. We position MAD-Arch as decision support rather than autonomous architecture decision-making.

Index Terms— multi-agent debate, large language models, software architecture, architecture decision records, architecture knowledge management, deployment governance, risk assessment, human-in-the-loop, DevOps, CI/CD, AI governance, software engineering

Introduction

A design review rarely fails because no plausible architecture can be drawn. It fails because the plausible architecture survives the first reading and then collapses under a constraint nobody raised in the room: a regional failover that doubles write latency, a token-refresh path that leaks across tenants, a deployment strategy that cannot roll back a schema migration, or an operational cost that only becomes visible at the third sprint. The decisive work of architecture review is not generation; it is adversarial scrutiny and the honest recording of trade-offs.

Two trends have made this scrutiny harder to supply at scale. First, the surface area of a modern design spans distributed-systems behavior, identity and data-protection regimes, reliability engineering, delivery and release governance, and cost—domains few individuals master simultaneously. Second, the rationale behind decisions is frequently lost. Teams adopt Architecture Decision Records to capture *why* a choice was made, but in practice ADRs are written after the fact, if at all, and the alternatives that were rejected—the most valuable part of the record—are summarized in a sentence or omitted.

LLMs are an obvious candidate to assist. A single capable model can read a design document and produce a reasonable critique. But a single model used as both author and reviewer exhibits a structural weakness: it may reinforce assumptions from its own proposal, surface a tidy and non-contested set of trade-offs, and express uniform

confidence whether or not the underlying reasoning is sound. The output reads as authoritative precisely when it should read as contested.

We argue that the useful unit for this problem is not a single model but a *structured disagreement* among several models playing distinct review roles, whose output is not a verdict but a set of auditable governance artifacts. We propose **MAD-Arch** (Multi-Agent Debate for Architecture), a framework in which a design-proposal agent produces a baseline, a panel of specialized reviewer roles independently critiques it from fixed adversarial perspectives, the roles debate the points where they disagree over a bounded number of rounds, and an arbiter consolidates the exchange into structured artifacts that a human architect reviews, edits, and approves. The framework is deliberately *not* autonomous: every artifact terminates at a human approval gate.

We emphasize that the contribution is not the observation that "LLMs can debate architecture." It is a specific framework design that (a) separates design *generation* from design *criticism* so that no agent grades its own work; (b) assigns reviewer roles whose objectives deliberately conflict, so that latent trade-offs surface as recorded disagreements; (c) converts those disagreements into a fixed schema of governance artifacts—ADRs, a risk register, a readiness checklist, and deployment-governance recommendations—rather than into free text; (d) attaches a transparent, rubric-based confidence score and an explicit unresolved-assumptions list to each decision; and (e) integrates the artifacts with pull requests and CI/CD gates while retaining a human approval point.

Scope and Non-Claims

This paper does not claim that MAD-Arch empirically outperforms single-model or human-review baselines. Rather, it proposes a framework, artifact schema, confidence rubric, and evaluation protocol intended to support future empirical validation. The illustrative values are included only to demonstrate how the protocol would be reported.

Concretely, this article makes the following contributions:

1. We frame architecture review as a *generation–criticism separation* problem and explain why a role-partitioned debate addresses the single-model failure modes of anchoring, silent trade-off resolution, and uncalibrated confidence (Sections IV, VI).
2. We propose the MAD-Arch framework: eight roles (seven model-driven agents plus the accountable human architect), the artifacts they exchange, and a governance loop terminating in a mandatory human approval gate (Section V), together with a repeatable debate workflow (Section VII).
3. We define precise schemas for the four governance artifacts—ADRs that record reviewer disagreement and confidence, a risk register with evidence and ownership, an evidence-backed readiness checklist mappable to CI/CD gates, and deployment-governance recommendations (Section VIII).
4. We specify a rubric-based arbiter *confidence model* that combines reviewer agreement, conflict, evidence strength, unresolved-assumption severity, and decision reversibility into a transparent score (Section IX).
5. We define a reproducible conceptual evaluation protocol over three reference scenarios with single-model and checklist-only baselines, a human adjudication rubric, risk de-duplication, and precision/recall-style scoring, and we report *illustrative* pilot-style values that demonstrate the protocol rather than prove effectiveness (Sections X, XI).
6. We document the practical concerns—prompt and configuration versioning, debate-log retention, CI/CD gating, pull-request linkage, and confidential-data handling—needed to operate the framework safely (Section XIII).

The remainder of the article covers background (Section II), related work (Section III), the problem definition (Section IV), the framework (Section V), the rationale for debate (Section VI), the debate workflow (Section VII), artifact generation (Section VIII), the confidence model (Section IX), evaluation methodology (Section X), illustrative results (Section XI), threats to validity (Section XII), implementation considerations (Section XIII), limitations (Section XIV), future work (Section XV), and conclusions (Section XVI).

Background and Motivation

For this article, an *architecture decision* is a choice among design alternatives that is costly to reverse and that constrains later choices—for example, selecting an event-sourced data model, a multi-region active-active topology, or a synchronous request path for an operation that could be asynchronous. An *Architecture Decision Record* is a short, structured document capturing the context, the decision, the alternatives considered, the consequences, and the status of such a choice.

We use *multi-agent debate* to mean a protocol in which several model instances produce independent positions on the same question and then exchange critiques across one or more rounds, optionally moderated by a separate agent that consolidates the result. The instances may be different model families, different versions, or the same model under different role prompts and sampling settings; the relevant property is that each holds a distinct vantage point and is prompted to challenge rather than confirm.

The motivation is operational. Architecture review is a bottleneck because it depends on a small pool of senior reviewers whose attention is scarce and whose reasoning is not captured. When review is rushed, risks surface in production—in incident retrospectives, security findings, or cost overruns—rather than at design time, where they are cheap to fix. A framework that broadens the review surface, forces trade-offs into the open, and emits auditable artifacts addresses both the throughput problem and the traceability problem, provided it does not substitute machine confidence for human judgment.

Related Work

We organize prior work into six lines and, for each, identify the part of the problem it addresses and the part it leaves open. We do not claim that the problem is untouched; rather, prior work addresses fragments of it, and none combines a role-partitioned, human-gated debate with the transformation of architectural disagreement into structured governance artifacts.

LLM self-reflection, self-critique, and refinement. A line of work prompts a single model to critique and iteratively revise its own output, improving quality on reasoning and coding tasks [1], [2]. For architecture review, the limiting property is that self-critique shares the generator's priors: a model that omits a failover concern while proposing a design is unlikely to surface that concern while reviewing the same design. MAD-Arch externalizes criticism into separately prompted roles and forbids any agent from grading its own proposal, which is precisely the coupling self-refinement retains.

Multi-agent debate and ensemble reasoning. Several studies show that having multiple model instances argue or that aggregating multiple sampled outputs improves factuality and reasoning [3]–[5]. This work largely targets short-form tasks with a checkable answer and measures accuracy against a ground-truth label. Architecture review differs in three ways: there is rarely a single correct answer, the relevant expertise is domain-partitioned (security versus cost versus reliability), and the desired output is a set of structured artifacts rather than a label. MAD-Arch adapts debate to this setting by fixing heterogeneous reviewer roles and routing the debate into a typed artifact schema.

LLMs for software engineering and architecture support. LLMs are increasingly studied for code generation, program repair, and developer assistance [6], [7], and more recently for design-level tasks. These efforts concentrate on producing or fixing concrete artifacts (code, tests, configurations); comparatively little addresses the upstream activity of *deciding* among architectural alternatives and recording why. MAD-Arch targets this decision and rationale layer rather than the code layer.

Architecture decision records and architecture knowledge management. Lightweight ADRs and supporting tooling have lowered the cost of capturing decisions [8], building on a broader literature on capturing and reasoning about architectural knowledge and design rationale [9]. In practice these remain manual: a person must still author the record and, most valuably, articulate the rejected alternatives, which are frequently omitted. MAD-Arch treats ADR generation as a by-product of a debate that has already enumerated and contested alternatives, so rejected options and their reasons are available rather than reconstructed after the fact.

DevOps governance, CI/CD readiness, and operational risk assessment. Continuous-delivery practice and reliability engineering define operational-readiness expectations—progressive delivery, rollback, observability, and runbooks—and link delivery practices to outcomes [11], [12], [10]. Policy-as-code and infrastructure scanners enforce many of these checks, but they operate at the artifact level (a manifest, a plan) and act after the design is fixed. MAD-Arch operates earlier, at the design-rationale level, and is complementary: its readiness checklist is designed to reference and defer to such scanners as authoritative for specific checks rather than to replace them.

AI governance, hallucination risk, and secure enterprise use. Frameworks for AI risk management and catalogs of LLM-specific risks—prompt injection, sensitive-data disclosure, over-reliance—inform the controls a production deployment requires [16], [15]. This guidance describes *what* risks exist; it does not prescribe an architecture-review workflow that operationalizes them. MAD-Arch incorporates these controls directly: a mandatory human gate, secret redaction, enterprise-tenant endpoints, and immutable debate logs for auditability.

In summary, prior work supplies the ingredients—self-critique, debate, code-level assistance, ADR capture, delivery governance, and AI-risk controls—but, to our knowledge, does not assemble them into a role-partitioned, human-gated debate workflow that transforms architectural disagreement into a fixed schema of auditable governance artifacts. That assembly is the contribution of MAD-Arch.

Problem Definition

Let a design proposal be specified by requirements R (functional and non-functional), constraints C (regulatory, budget, platform, timeline), and a baseline architecture A expressed as components, data flows, and deployment topology. An architecture review must produce a decision set D and a justification J such that D satisfies C, optimizes a weighted trade-off over qualities in R (latency, throughput, availability, security posture, cost, operability), and records J in a form a future engineer can audit.

A review *fails* when any of the following holds even though the output appears coherent:

- A binding constraint in C is violated and the violation is not surfaced (a missed risk).
- A material trade-off is resolved silently, without recording the rejected alternative and its reason (lost rationale).
- The output expresses confidence not warranted by the underlying reasoning (false confidence).

The single-model formulation collapses generation and criticism into one process that shares priors, which makes the second and third failure modes systematic rather than incidental. The problem this article addresses is: *how to organize multiple models so that criticism is independent of generation, trade-offs are forced into the record, confidence is calibrated and explicit, and a human remains the accountable decision-maker.*

Proposed Framework: MAD-Arch

Components and Roles

MAD-Arch comprises eight roles. Seven are model-driven agents; the eighth is the human architect, who owns the decision. Table I summarizes them.

Table I. MAD-Arch roles, responsibilities, and primary artifacts.

Role	Responsibility	Key inputs	Key outputs
Design Proposal Agent	Produce a baseline architecture from requirements and constraints	R, C	Baseline architecture A, assumptions list
Scalability Reviewer	Assess throughput, latency, storage growth, distributed bottlenecks, scaling strategy	A, R	Scalability findings, scaling-risk items
Security Reviewer	Assess authN/authZ, secrets,	A, C	Security findings, compliance-

Role	Responsibility	Key inputs	Key outputs
	data exposure, abuse cases, compliance		risk items
Reliability / SRE Reviewer	Assess failure modes, retries, observability, SLI/SLO, rollback, incident response	A	Reliability findings, readiness items
DevOps / Delivery Reviewer	Assess CI/CD, deployment strategy, feature flags, promotion, release governance	A, C	Delivery findings, governance recommendations
Cost & Complexity Reviewer	Assess operational cost, infrastructure overhead, cognitive load, maintainability	A, C	Cost findings, complexity-risk items
Architecture Judge / Arbiter	Consolidate debate, resolve conflicts, rank options, score confidence	All findings, debate transcript	ADRs, risk register, readiness checklist, final recommendation
Human Architect	Review, edit, approve or reject, take accountability	All artifacts	Approved/edited artifacts, sign-off

The five reviewer roles are deliberately partitioned so that their objectives conflict. The scalability reviewer will favor caching and replication; the cost reviewer will challenge the resulting infrastructure footprint; the reliability reviewer will challenge cache-invalidation correctness during failover. These conflicts are the point: they surface the trade-offs that a single perspective would resolve silently.

Input and Output Artifacts

The framework consumes a *design brief* (R, C, and optionally a draft architecture) and produces five artifacts: ADRs, a risk register, an operational-readiness checklist, deployment-governance recommendations, and a final recommendation with a confidence score and an explicit list of unresolved assumptions. Every artifact is emitted as version-controllable Markdown so it can live alongside code and be diffed in a pull request.

Governance Loop and Human Approval Point

The arbiter never commits artifacts directly. It produces *proposed* artifacts and routes them to the human architect, who may accept, edit, or reject any item, and who may send specific points back for another debate round. Only human-approved artifacts are merged. This single approval gate is what makes the framework decision support rather than an autonomous decision-maker: the confidence score and unresolved-assumptions list exist precisely to direct the architect's limited attention to the parts of the design the panel was least sure about. Figure 1 shows the end-to-end flow.

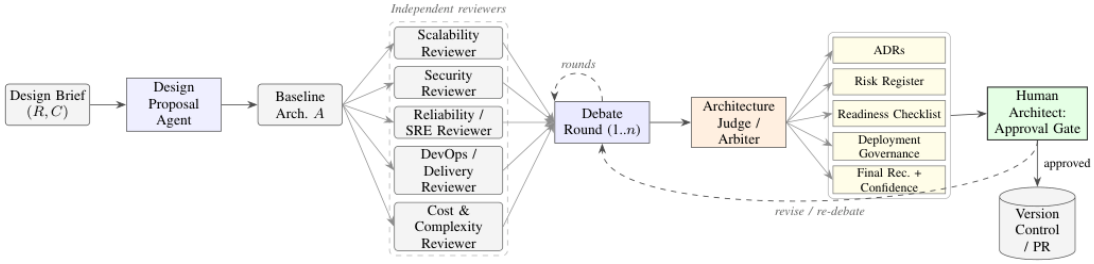

Figure 1. MAD-Arch pipeline. A design brief is turned into a baseline architecture, fanned out to five independent adversarial reviewers, reconciled over one or more debate rounds, consolidated by the arbiter into five structured artifacts, and gated on human-architect approval before reaching version control. The dashed path returns rejected items for re-debate; accountability remains with the human.

Design Rationale for Multi-Agent Architecture Review

The case for role-partitioned debate is not that more model calls are inherently better; it is that the protocol changes the *structure* of the review in ways that map directly onto the failure modes of Section IV.

- **It reduces single-model blind spots.** Independent role prompts—and, where available, different model families—do not share the same training-induced gaps, so a concern missed by one vantage point is more likely caught by another.
- **It makes criticism adversarial by construction.** Reviewers are instructed to find what is wrong, not to confirm the design. The cost reviewer is prompted to challenge the scalability reviewer's expensive proposal, which surfaces the trade-off explicitly.
- **It separates generation from criticism.** The proposal agent's job ends when the baseline is produced; it does not grade its own work. This breaks the shared-prior coupling that makes single-model self-critique weak.
- **It makes trade-offs explicit and recorded.** A disagreement between two reviewers is, by definition, a trade-off. The arbiter records both positions and the basis for resolving them, which is exactly the rejected-alternative content that manual ADRs usually omit.
- **It produces auditable artifacts.** The debate transcript, the per-reviewer findings, and the arbiter's resolution form a traceable chain from concern to decision, supporting governance and post-hoc review.
- **It calibrates confidence.** When reviewers converge, the arbiter can report high confidence; when they remain split after debate, that split is reported as a low-confidence item or an unresolved assumption rather than being averaged into false certainty.

The trade-off is real and worth stating: role-partitioned debate costs more tokens, more latency, and more orchestration complexity than a single call. Section XI characterizes that cost, and Section XIV notes where it is not worth paying.

Debate Workflow

MAD-Arch executes a repeatable ten-step workflow. We describe it at a level of detail sufficient for reproduction; the concrete model, sampling, and round-count settings used in our illustrative protocol are given in Section X.

1. **Ingest the design brief.** Requirements R and constraints C are supplied, optionally with a draft architecture. Secrets and customer identifiers are redacted before any model call (Section XIII).
2. **Generate the baseline.** The Design Proposal Agent emits a baseline architecture A and an explicit assumptions list.
3. **Instantiate roles.** The five reviewer roles and the arbiter are instantiated from versioned prompt templates; the proposal agent takes no further part in scoring its own output.
4. **Independent review (round 0).** Each reviewer critiques A from its fixed perspective without access to the other reviewers' outputs, so that initial findings are statistically independent rather than mutually anchored.
5. **Debate (rounds $1 \dots n$).** Reviewers are shown the points where their findings conflict and are asked to defend, concede, or refine each position. Rounds continue until positions stabilize (no reviewer changes a stance) or a cap $n_{\max}$ is reached, whichever comes first.
6. **Arbitration.** The arbiter consolidates the stabilized positions, resolves or explicitly flags each conflict, ranks the surviving options, and computes a confidence score per decision (Section IX).
7. **Generate ADRs** from the resolved decisions, carrying forward the contested alternatives and the reviewer disagreement.
8. **Generate the risk register** from the surfaced findings, after de-duplication (Section X).
9. **Generate the readiness checklist and deployment-governance recommendations**, mapping each gate to an enforcement point.
10. **Human approval.** The architect approves, edits, or returns specific items for re-debate; only approved artifacts are merged.

Architecture Artifact Generation

The artifacts are the framework's product. Each has a fixed structure so that downstream tooling—diffs, CI gates, dashboards—can parse and act on it.

ADR Template

Each ADR carries nine fields: status, context, decision, alternatives considered, consequences, reviewer disagreements, confidence score, unresolved assumptions, and human-approval status. The reviewer-disagreement and confidence fields are what distinguish a MAD-Arch ADR from a manually authored one: they preserve the contested reasoning rather than presenting the decision as uncontested. Figure 2 shows an example.

```
# ADR-014: Outbox pattern for
#          order-event publication
- Status: Proposed
- Human approval: PENDING <architect>
- Confidence: 0.78
- Date: 2026-06-12

## Context
Orders must publish events to downstream
fulfillment without dual-write
inconsistency between DB and broker.

## Decision
Adopt the transactional outbox pattern
with a CDC relay to the broker.

## Alternatives Considered
1. Synchronous dual write -- rejected:
no atomicity, risk of lost events.
2. Broker-first with DB projection --
rejected: complicates read-after-write.
3. Outbox + CDC -- selected: atomic with
the order transaction.

## Reviewer Disagreements
- Cost reviewer dissents: CDC relay adds
approximately 15% infrastructure cost;
proposes option 2.
- Scalability + Reliability concur with
option 3; Security/Delivery neutral.

## Consequences
+ Eliminates dual-write race; events are
exactly-once to the relay.
- Adds a CDC component and relay lag.

## Unresolved Assumptions
- Relay lag SLO (<2s) is assumed, not
yet validated under peak load. (HIGH)

## Confidence Breakdown
- Agreement a(d): 0.80
- Conflict penalty c(d): 0.20
- Evidence strength e(d): 0.70
- Unresolved-assumption severity u(d): 0.40
- Reversibility r(d): 0.60
- Final confidence: 0.78
```

Figure 2. Example ADR emitted by MAD-Arch, carrying reviewer disagreement, a confidence score, unresolved assumptions, and a human-approval field.

Risk Register Format

Table II gives the risk-register schema; the framework emits one row per identified risk.

Table II. Risk register schema (one row per identified risk).

Field	Description
ID	Stable identifier (e.g., RISK-007)
Category	Security / Scalability / Reliability / Delivery / Cost / Compliance
Description	What can go wrong, stated as a condition
Likelihood	Low / Medium / High (reviewer-assigned)
Impact	Low / Medium / High
Evidence	Brief / debate citation, scanner finding, or assumption flag
Raised by	Reviewer role(s) that surfaced it
Mitigation	Proposed control or design change
Owner	Human-assigned after review
Status	Open / Mitigated / Accepted / Deferred

Operational-Readiness Checklist Format

The readiness checklist is a list of binary, evidence-backed gates rather than prose, so it can be enforced in CI. Figure 3 shows example items, each carrying a state (pass / fail / n/a) and an evidence link.

```
[ ] SLOs defined for all user-facing
    endpoints (latency, availability)
[ ] Dashboards and alerts exist for the
    four golden signals
[ ] Rollback procedure documented and
    tested in staging
[ ] Schema migrations backward-compatible
    / reversible
[ ] Secrets in a managed store, none in
    config or images
[ ] Deployment supports progressive
    rollout (canary/blue-green)
[ ] Runbook exists for top 3 predicted
    failure modes
[ ] Load test covers projected peak +50%
```

Figure 3. Example operational-readiness checklist items. Each gate carries a pass/fail/n/a state and an evidence link; an item without evidence defaults to fail.

Each checklist item is required to be *evidence-backed*: it carries a state (pass/fail/n/a) and a link to the evidence that justifies the state—a dashboard, a load-test report, a security-scan result, a runbook page, or a staging rollback record. An item without evidence defaults to fail, so the checklist cannot be satisfied by assertion alone.

Deployment-Governance Recommendations

Deployment-governance recommendations are emitted as a separate artifact that maps each readiness gate to a concrete enforcement point, so that the checklist is operational rather than advisory. Each recommendation names (i) the gate, (ii) the enforcement mechanism—a CI job, a deployment-pipeline manual approval, an observability/alerting requirement, a security-scan stage, or a feature-flag/progressive-rollout policy—and (iii) the failure action (block merge, block promotion, or warn). For example, the readiness item "schema migrations reversible" maps to a CI job that fails the build when a migration lacks a tested down-path; "high-impact open risks owned" maps to a pipeline approval that blocks promotion until each such risk in the register has an assigned owner and status. This mapping is what allows the artifacts to participate in CI/CD gating (Section XIII).

Arbiter Confidence Model

A confidence score is useful only if it is transparent and reflects the actual state of the debate rather than the fluency of the model's prose. We therefore define confidence as an explicit rubric over five factors that the arbiter can observe directly from the debate, not as a self-reported probability. For a decision d, let:

- $a(d) \in [0,1]$ — *agreement*: the fraction of the k reviewer roles that endorse the selected option after the final debate round;
- $c(d) \in [0,1]$ — *conflict penalty*: the normalized severity of the strongest sustained dissent (0 if no reviewer dissents, 1 if a reviewer rates the decision a blocking risk);
- $e(d) \in [0,1]$ — *evidence strength*: whether the decision rests on cited facts in the brief or on scanner output (high) versus on reviewer assertion (low);
- $u(d) \in [0,1]$ — *unresolved-assumption severity*: the maximum severity among the decision's unresolved assumptions (0 none, 1 a high-severity unvalidated assumption);
- $r(d) \in [0,1]$ — *reversibility*: 1 if the decision is easily reversible, 0 if it is high-blast-radius and hard to undo.

We combine these into a bounded score

$$\mathrm{conf}(d) = \mathrm{clip}_{[0,1]}(w_a a + w_e e + w_r r - w_c c - w_u u),$$

where the non-negative weights $w_{\bullet}$ are reported alongside the score so the calculation is auditable. Unless otherwise configured, we use $w_a = w_e = w_r = w_c = w_u = 0.20$, giving equal positive weight to agreement, evidence, and reversibility, and equal penalty weight to sustained conflict and unresolved assumptions. Organizations may tune these weights based on risk tolerance; for example, a regulated deployment may assign a larger penalty to unresolved assumptions. The reversibility term encodes a deliberate asymmetry: an irreversible, high-blast-radius decision is assigned lower confidence even under reviewer agreement, because the cost of being wrong is higher and the architect's attention should be drawn to it. The score should be interpreted as a triage signal, not as a calibrated probability that the decision is correct; Equation (1) is a rubric rather than a statistical estimator. The arbiter additionally emits the per-factor values so a reviewer can see *why* a decision scored as it did—for instance, that a 0.78 reflects strong agreement discounted by an unresolved peak-load assumption.

Evaluation Methodology

MAD-Arch is a framework, not a trained model, so we evaluate it as a design-science artifact and specify a *conceptual evaluation protocol*: a reproducible procedure that another researcher could run to obtain real measurements. We deliberately separate the protocol (this section) from the illustrative values that demonstrate it (Section XI). We did not conduct a production deployment; the protocol is the contribution here, and obtaining measured results is future work (Section XV).

Reference Scenarios

- **Scenario A — Cloud-native order-processing system.** Synchronous order intake, asynchronous fulfillment, multi-region availability target, PCI-relevant payment path. Stresses consistency, failover, and compliance.
- **Scenario B — Event-driven IoT telemetry platform.** High-cardinality device ingestion, time-series storage, backpressure, intermittent connectivity. Stresses throughput, storage cost, and partial-failure handling.
- **Scenario C — AI-powered document-processing pipeline.** OCR plus LLM extraction over sensitive documents, human-in-the-loop verification, audit requirements. Stresses data exposure, cost variance, and reliability of non-deterministic components.

For each scenario, a fixed design brief (R, C) is authored in advance, and a human adjudicator constructs a *reference set* of valid risks and expected trade-offs before any condition is run, so that scoring is against a pre-registered ground truth rather than a post-hoc judgment.

Comparison Conditions

1. **Single-model review (SM):** one LLM both proposes and reviews the design in a single pass.
2. **Checklist-only human review (CL):** a human reviewer applies a fixed architecture checklist without model assistance.
3. **MAD-Arch (MA):** the full role-partitioned debate workflow with human approval.

Configuration (for reproducibility)

To make the protocol reproducible, a run must record the following; we give placeholders rather than committing to specific products, since the protocol is model-agnostic.

- *Models.* Reviewer roles use models $M_1 \dots M_5$ and the arbiter M_{arb}; record the exact family and version of each (e.g., <vendor>-<model>-<version>). The *diverse* configuration assigns different families across roles; the *homogeneous* configuration uses one family with role-specific prompts.
- *Sampling.* Record temperature and top-p per role; we suggest a low temperature (e.g., 0.2) for the arbiter to favor determinism and a moderate temperature (e.g., 0.7) for reviewers to encourage divergent critique.
- *Debate rounds.* Record $n_{\max}$ (e.g., 2) and the stabilization rule used to terminate early.
- *Prompt-template versions.* Record the version hash of each role template; every artifact stores the versions that produced it (Section XIII).
- *Seeds and repetitions.* Because outputs are non-deterministic, each condition is run N times per scenario (e.g., $N = 5$) and metrics are reported as mean $\pm$ standard deviation.

Metrics and Scoring Rubric

Let the adjudicator's reference set for a scenario contain G valid risks. For a condition's output we de-duplicate risks (below), then label each reported risk as a true positive (matches a reference risk), a false positive (invalid, non-applicable, or hallucinated), or a duplicate (merged, not counted). With TP true positives and FP false positives, we report:

$$\text{recall} = TP/G, \quad \text{precision} = TP/(TP + FP).$$

The full metric set is: (1) *valid risks identified* (TP) and risk recall; (2) *precision* on reported risks; (3) *trade-off coverage* (fraction of the reference trade-offs captured with both sides recorded); (4) *ADR completeness* (fraction of the nine ADR fields populated with substantive, non-templated content); (5) *readiness coverage* (fraction of the reference readiness checklist addressed with evidence); (6) *redundant/incorrect recommendation rate* (rejected or duplicate recommendations over total recommendations); and (7) *reviewer usefulness* (the adjudicating architect's 1–5 rating of how useful the artifacts were for the actual decision).

Table III. Evaluation scoring rubric.

Metric	Score basis	Human judgment?
Risk recall	TP / reference risks	Yes
Risk precision	TP / reported risks	Yes
Trade-off coverage	Captured trade-offs / expected trade-offs	Yes
ADR completeness	Substantive fields / 9 ADR fields	Partial
Readiness coverage	Evidence-backed checks / expected checks	Yes
Usefulness	1–5 architect rating	Yes

De-duplication and Adjudication

Risks are de-duplicated by (category,affected component,failure condition): two items that share all three are merged into one, retaining the stronger mitigation and the union of raising roles. A risk is counted *valid* only if it applies to the scenario as briefed and is not contradicted by the brief; items that restate a constraint, target an out-of-scope component, or assert a non-existent mechanism are counted as false positives. Because architecture review is partly judgment-based, adjudicator expertise and organizational context can affect which risks are considered valid. To limit adjudicator bias, outputs from the three conditions are anonymized and shuffled before scoring, and—where resources permit—two independent adjudicators score each scenario with inter-rater agreement (e.g., Cohen's κ) reported; disagreements are resolved by discussion against the pre-registered reference set. Confidence scores produced by MAD-Arch (Section IX) are evaluated separately for *calibration*: for each decision the adjudicator records whether it was ultimately sound, and confidence is compared against this outcome (e.g., via a reliability curve) rather than assumed to be correct.

Illustrative Conceptual Results and Discussion

The values in Table IV and Fig. 4 are illustrative placeholders and were not obtained from a production deployment or a controlled experiment. They are hand-constructed to demonstrate how the protocol of Section X would be reported and to convey the relative behavior the framework is *designed* to produce. They are not evidence of effectiveness; empirical validation is future work (Section XV).

Table IV. Example reporting format for the proposed evaluation protocol. Values are illustrative placeholders and should be replaced by measured values in an empirical deployment; they are not experimental evidence. Higher is better except for the last row.

Metric	SM	CL	MA
Valid risks identified (count)	9	11	**17**
Risk recall (0–1)	0.45	0.55	**0.85**
Risk precision (0–1)	0.78	**0.92**	0.86
Trade-off coverage (0–1)	0.25	0.42	**0.83**
ADR completeness (0–1)	0.55	0.70	**0.88**
Readiness coverage (0–1)	0.50	0.75	**0.85**
Reviewer usefulness (1–5)	3.1	3.6	**4.2**
Redundant/incorrect rate (0–1)	0.22	**0.08**	0.14

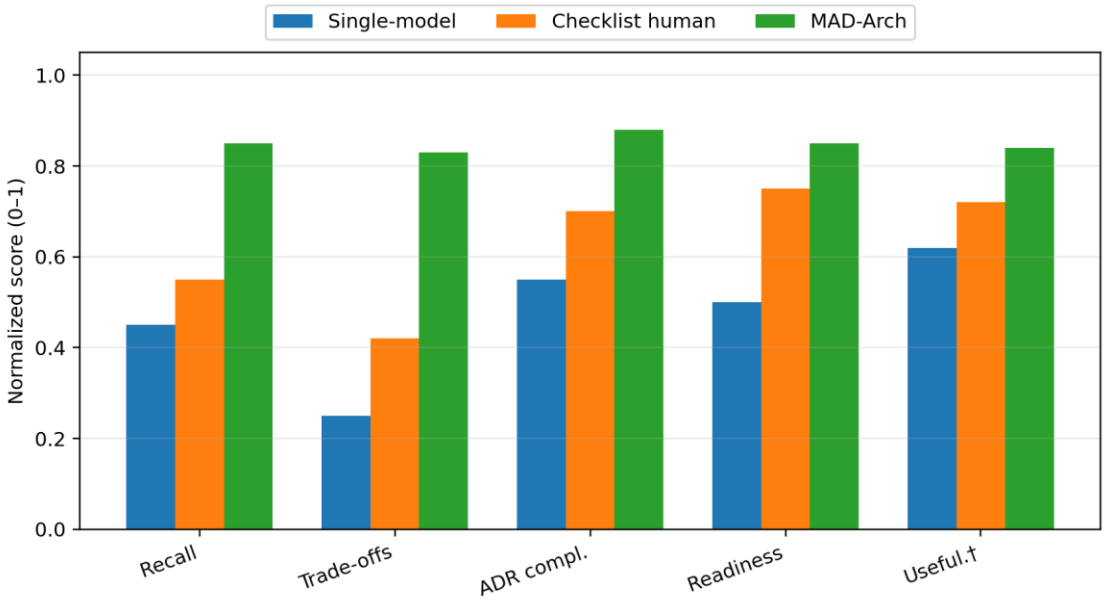

Figure 4. Example visualization template for reporting normalized quality metrics across the three review conditions. †Reviewer usefulness is rescaled from its 1–5 scale. Values are illustrative placeholders, not experimental evidence, and should be replaced by measured values in an empirical deployment.

The illustrative pattern in Fig. 4 is the behavior the framework is *designed* to produce; whether it materializes is an empirical question we do not answer here. The intended mechanism is that disagreement between adversarial roles converts an implicit trade-off into a recorded one, which would raise trade-off coverage; and that generating ADRs from a debate that already enumerated alternatives would raise ADR completeness. We also expect a characteristic tension: the same divergent critique that we hypothesize raises recall would tend to lower precision, because debate generates more material, some of it noise. In the illustrative values the redundant/incorrect rate for MAD-Arch (0.14) sits *between* the single model (0.22) and the disciplined human checklist (0.08), which is precisely why the human approval gate and the confidence score are load-bearing: they are the filter intended to keep higher recall from becoming higher downstream noise. These directional expectations are hypotheses for the protocol of Section X to test, not findings.

The cost dimension is the honest counterweight. A five-reviewer debate with one or two rounds and an arbiter pass consumes on the order of 8–12 model calls per review versus one for the single-model condition, with correspondingly higher latency and token cost. For a high-stakes, hard-to-reverse decision this overhead is small relative to the cost of a wrong architecture; for a low-risk, easily reversible change it is likely not worth paying. We therefore suggest applying the framework selectively, gated on the reversibility and blast radius of the decision under review—the same factors that enter the confidence model in Equation (1).

Threats to Validity

We discuss threats under the standard categories and note our intended mitigations; because we report no measured results, several threats apply to the *protocol* rather than to findings.

Construct validity. The risk and trade-off metrics depend on the adjudicator's reference set; a different adjudicator may define "valid" differently, and the 1–5 usefulness rating is inherently subjective. We mitigate by pre-registering the reference set before any condition runs, anonymizing and shuffling outputs before scoring, and reporting inter-rater agreement where two adjudicators are available. The confidence score is a rubric (Equation (1)), not a measured probability; we therefore evaluate its calibration separately rather than treating it as ground truth.

Internal validity. In the homogeneous configuration the reviewer roles share one model family, which partially reintroduces the shared-prior coupling that debate is meant to break: apparent "independent" agreement may reflect a common training bias rather than genuine corroboration. The diverse configuration mitigates this but raises cost and complicates prompt management. Prompt sensitivity and model-version drift are additional internal threats: a change in template wording or model version can alter outputs, which is why we version both and report them.

External validity. Three reference scenarios cannot represent the space of architectures; results on, e.g., a real-time trading system, an embedded control system, or a data-intensive analytics platform may differ materially. The framework's value may also depend on organizational context (review maturity, regulatory burden) that the scenarios do not capture.

Conclusion validity. We draw no statistical conclusions. The values in Table IV and Fig. 4 are hand-constructed to illustrate the reporting format and the hypothesized direction of effects; they are not evidence, and the relationships among them should not be interpreted as effect sizes. Establishing whether the hypothesized effects hold, and with what variance across repeated non-deterministic runs, requires the empirical study described in Section XV.

Practical Implementation Considerations

The following concerns determine whether the framework is operable and auditable in a real engineering organization, not only conceptually sound.

Prompt templates as version-controlled artifacts. Each role is defined by a prompt template that fixes its objective, the artifact schema it must emit, and its adversarial stance. Templates live in the repository, are reviewed like code, and are addressed by a version hash. Every emitted artifact records the template versions that produced it. Treating prompts as code—reviewed, diffed, and versioned—is what allows a later reader to reproduce or contest a decision rather than encounter an unexplained output.

Model-configuration tracking. Each run records the model family and version for every role and the arbiter, plus sampling parameters (temperature, top-p) and the debate-round cap. Because the same template can behave differently across model versions, this record is part of the audit trail and is required to interpret or reproduce a result; a configuration change is treated as a material change to the review.

Storing debate logs. The full transcript—baseline, per-reviewer findings, each debate round, and the arbiter's resolution and confidence breakdown—is persisted as an immutable record keyed to the design brief and the prompt/model versions. This log is the primary audit artifact: it lets a governance reviewer reconstruct *why* a decision was made, and which reviewer raised which concern, months later.

Linking artifacts to pull requests. The generated ADRs, risk register, and readiness checklist are committed as Markdown alongside the code and linked to the originating pull request, so the decision rationale travels with the change it justifies and is reviewed in the same place. Subsequent edits to an artifact are diffed like any other source change.

CI/CD gating and human-approval workflow. The readiness checklist is machine-parseable, so a CI job can fail the build when a required, evidence-backed gate is unmet, and a deployment-pipeline stage can block promotion until every high-impact risk in the register has an owner and status. Crucially, the pipeline also enforces the human gate: artifacts carry a Human approval: PENDING/APPROVED field, and merge is blocked until an authorized architect approves. The framework supplies the analysis, the pipeline supplies the enforcement, and the human supplies the accountable decision; these responsibilities are kept distinct by construction.

Secure handling of enterprise design documents. Design briefs routinely contain confidential architecture, customer, and security details. The framework is intended to run against self-hosted or enterprise-tenant model endpoints with no training retention, to redact secrets and customer identifiers before any model call, and to store debate logs in an access-controlled location. Sending an internal design to an uncontrolled public endpoint is a confidentiality incident regardless of the quality of the review returned; in the threat model of [15] this also bounds exposure to prompt-injection and data-leakage paths. These controls are preconditions for use, not optional hardening, and they align the deployment with established AI-risk guidance [16].

Auditability, confidentiality, and reproducibility together. The same mechanisms serve three goals at once: versioned prompts and configurations plus immutable logs give *reproducibility*; the human-approval field and the per-decision confidence breakdown give *auditability*; and redaction with controlled endpoints gives *confidentiality*. We regard these as the minimum viable governance surface for using LLM debate on production architecture.

Limitations

We state the framework's limitations directly, since several are intrinsic rather than incidental.

Hallucination and false confidence. The framework does not eliminate hallucination. A reviewer can invent an inapplicable risk or assert a mitigation that does not work, and a fluent but unsound argument can survive debate. The human gate and the redundant/incorrect metric exist precisely because some output will be wrong. MAD-Arch limits *where* such errors can act—no artifact reaches production without human sign-off—but it does not prevent them, and a confident-sounding wrong recommendation remains possible.

Confidence is a heuristic. The score of Equation (1) reflects the observable state of the debate, not a calibrated probability of correctness. Reviewer agreement can be spurious, particularly in the homogeneous configuration, and high agreement on a wrong premise yields misleadingly high confidence. The score should direct attention, not settle questions.

Shared-prior risk. When all roles use the same model family, their critiques are correlated, and the independence that motivates debate is only partially realized. Model diversity reduces but does not remove this, and diversity itself adds cost and operational complexity.

Sensitivity and non-determinism. Outputs are sensitive to prompt phrasing and model version, and repeated runs of the same brief can differ. Versioning and logging manage this but do not make the framework deterministic; reported results must therefore be aggregated over repetitions.

Evaluation subjectivity and limited validation. Our metrics depend on human adjudication and a small set of scenarios, and we report no production results. The framework's effectiveness is, at this stage, hypothesized rather than demonstrated.

Over-engineering and overhead. Five reviewers each advocating for their domain can collectively recommend more machinery than a problem warrants; the cost-and-complexity role and the human gate are deliberate countermeasures, not afterthoughts. The debate also incurs real token, latency, and orchestration cost, which makes the framework unsuitable for low-stakes, easily reversible changes.

Confidentiality residual risk. Even with redaction and controlled endpoints, sending design information to an LLM carries residual exposure; organizations with the strictest data-handling requirements may need stronger isolation than the framework assumes.

Not a replacement for the architect. Accountability for the decision remains entirely human. MAD-Arch is decision support: it widens the review surface and records rationale, but the responsibility for the architecture does not transfer to the system.

Future Work

The most important next step is *empirical validation*: executing the protocol of Section X on real, anonymized design reviews with multiple human adjudicators, replacing the illustrative values with measured ones and reporting variance across repeated non-deterministic runs and inter-rater agreement. A second direction is *confidence calibration*: tuning the weights of Equation (1)—or replacing the rubric with a model learned from adjudicated debates—and validating the result against held-out outcomes via reliability curves, so the score reflects actual correctness rather than debate agreement alone. A third is *tighter governance integration*: connecting the readiness checklist and risk register to existing policy engines and infrastructure scanners so the framework defers to authoritative checks where they exist and only reasons where they do not. Finally, we plan to study *adaptive depth*—varying the number of debate rounds and reviewer roles as a function of a decision's reversibility and blast radius—so that orchestration cost tracks the stakes of the decision under review.

Artifact Availability

The framework is intended to be accompanied by versioned prompt templates, scenario briefs, scoring rubrics, and sample generated artifacts. In future empirical work, these materials will be released as a reproducibility package where confidentiality constraints permit.

Conclusion

Architecture review fails not for lack of plausible designs but for lack of independent scrutiny and honest, recorded trade-offs. A single LLM used as both author and critic can inherit its own blind spots and present contested choices as settled. MAD-Arch reorganizes the work: it separates generation from criticism, assigns adversarial review roles whose objectives conflict by design, debates the disagreements, and consolidates the exchange into version-controlled ADRs, a risk register, a readiness checklist, and a final recommendation that carries an explicit confidence score and a list of what remains unresolved. The framework is human-in-the-loop by construction—every artifact terminates at an architect's approval gate, and accountability never leaves the human. Its intended value is not autonomy but auditability and breadth: it is designed to widen the review surface and force trade-offs into the record while leaving the decision where it belongs. We have specified the framework, a rubric-based confidence model, and a reproducible evaluation protocol, and we have illustrated the behavior the design aims to produce. We have not yet demonstrated that behavior empirically; doing so, and calibrating the confidence model against real outcomes, is the work we intend to pursue next.

Acknowledgments

The views and opinions expressed in this article are solely those of the authors and do not represent or reflect the views, positions, policies, or opinions of their employers or any affiliated organizations. The content is provided for informational purposes only. The authors' employers make no representations or warranties regarding the accuracy or completeness of the content and do not endorse or accept responsibility for any statements, conclusions, or materials presented herein.

[1] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, New Orleans, LA, USA, 2023, pp. 8634–8652, doi: 10.52202/075280-0377.

[2] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Welleck, B. P. Majumder, S. Gupta, A. Yazdanbakhsh, and P. Clark, "Self-Refine: Iterative refinement with self-feedback," in *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, New Orleans, LA, USA, 2023, pp. 46534–46594, doi: 10.52202/075280-2019.

[3] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, "Improving factuality and reasoning in language models through multi-agent debate," *arXiv:2305.14325*, 2023, doi: 10.48550/arXiv.2305.14325.

[4] T. Liang, Z. He, W. Jiao, X. Wang, Y. Wang, R. Wang, Y. Yang, S. Shi, and Z. Tu, "Encouraging divergent thinking in large language models through multi-agent debate," in *Proc. 2024 Conf. Empirical Methods in Natural Language Processing (EMNLP)*, Miami, FL, USA, 2024, pp. 17889–17904, doi: 10.18653/v1/2024.emnlp-main.992.

[5] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain-of-thought reasoning in language models," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2023.

[6] M. Chen et al., "Evaluating large language models trained on code," *arXiv:2107.03374*, 2021, doi: 10.48550/arXiv.2107.03374.

[7] A. Fan et al., "Large language models for software engineering: Survey and open problems," in *Proc. IEEE/ACM Int. Conf. Software Engineering: Future of Software Engineering (ICSE-FoSE)*, Melbourne, VIC, Australia, 2023, pp. 31–53, doi: 10.1109/ICSE-FoSE59343.2023.00008.

[8] M. Nygard, "Documenting architecture decisions," Cognitect Blog, Nov. 2011. [Online]. Available: https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions. Accessed: Jun. 12, 2026.

[9] P. Kruchten, P. Lago, and H. van Vliet, "Building up and reasoning about architectural knowledge," in *Quality of Software Architectures (QoSA 2006)*, C. Hofmeister, I. Crnkovic, and R. Reussner, Eds., LNCS 4214. Berlin, Germany: Springer, 2006, pp. 43–58, doi: 10.1007/11921998_8.

[10] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Boston, MA, USA: Addison-Wesley, 2010.

[11] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., *Site Reliability Engineering: How Google Runs Production Systems*. Sebastopol, CA, USA: O'Reilly Media, 2016.

[12] N. Forsgren, J. Humble, and G. Kim, *Accelerate: The Science of Lean Software and DevOps*. Portland, OR, USA: IT Revolution Press, 2018.

[13] M. Nygard, *Release It! Design and Deploy Production-Ready Software*, 2nd ed. Raleigh, NC, USA: Pragmatic Bookshelf, 2018.

[14] M. Richards and N. Ford, *Fundamentals of Software Architecture: An Engineering Approach*. Sebastopol, CA, USA: O'Reilly Media, 2020.

[15] OWASP Foundation, "OWASP Top 10 for Large Language Model Applications," 2025. [Online]. Available: https://owasp.org/www-project-top-10-for-large-language-model-applications/. Accessed: Jun. 12, 2026.

[16] E. Tabassi, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST AI 100-1, Jan. 2023, doi: 10.6028/NIST.AI.100-1.