

Contents

1	Empirical Research Design: A Security-by-Design Evaluation Framework for Ontology-Grounded Code Generation	2
1.1	1. Research Objective	2
1.1.1	1.1.1 What Papers 1–3 Establish and Assume	2
1.1.2	1.1.2 The Central Claim	3
1.1.3	1.1.3 What Must Be Demonstrated Empirically	4
1.1.4	1.1.4 Constrained vs. Unconstrained Execution: Evaluation Framing	4
1.1.5	1.1.5 Research Scope	5
1.2	2. Related Work and the SbD Evaluation Gap	5
1.2.1	2.1 Existing Benchmarks	5
1.2.2	2.2 The Gap	6
1.3	3. Research Questions and Hypotheses	6
1.3.1	3.1 Primary Research Questions	6
1.3.2	3.2 Secondary Research Questions	7
1.3.3	3.3 Exploratory Research Questions	7
1.4	4. The SbD Evaluation Framework	7
1.4.1	4.1 Evaluation Principles	8
1.4.2	4.2 SbD Completeness Score (M_SbD)	8
1.4.3	4.3 Multi-Tier Tool Oracle	8
1.4.4	4.4 Audit Trail Score (M_audit)	9
1.4.5	4.5 Retrieval Recall Validation (M_recall)	9
1.5	5. Experimental Conditions	10
1.5.1	5.1 Primary Groups	10
1.5.2	5.1 G2: MCP Server — Technical Summary	10
1.5.3	5.2 Ablation Groups (Phase 2)	11
1.5.4	5.3 LLMs	11
1.6	6. Task Set and Benchmark	11
1.6.1	6.1 Task Design Principles	11
1.6.2	6.2 Task Set	12
1.6.3	6.3 Per-Task Ground Truth Package	13
1.7	7. Variables and Metrics	14
1.7.1	7.1 Independent Variables	14
1.7.2	7.2 Primary Metrics	14
1.7.3	7.3 Secondary Metrics	14
1.7.4	7.4 Cost Metrics	14
1.7.5	7.5 Control Variables	15
1.8	8. Measurement and Instrumentation	15
1.8.1	8.1 M_SbD Scoring Protocol	15
1.8.2	8.2 M_audit Scoring Protocol	16
1.8.3	8.3 M_recall Validation	16
1.8.4	8.4 M_func Scoring Protocol	16
1.8.5	8.5 Cost Measurement	18
1.9	9. Experimental Controls	18
1.9.1	9.1 Fairness Controls	18
1.9.2	9.2 Risk Controls	18
1.10	10. Statistical Analysis Plan	19

1.10.1	Design	19
1.10.2	Primary Analysis (RQ1)	19
1.10.3	Sample Size and Power	19
1.10.4	Secondary and Exploratory Analyses	20
1.10.5	Multiple Comparisons Budget	20
1.11	11. Threats to Validity	20
1.11.1	Internal Validity	20
1.11.2	Construct Validity	20
1.11.3	External Validity	21
1.11.4	G2-Specific Validity	21
1.12	12. Expected Paper Structure	22
1.12.1	Recommended: Single Comprehensive Paper	22
1.13	13. Minimal Viable Study (Phase 1)	23
1.13.1	Phase 1 Infrastructure Checklist	23
1.14	14. Open Design Decisions	24
1.15	Assumptions	24
1.16	References	25

1 Empirical Research Design: A Security-by-Design Evaluation Framework for Ontology-Grounded Code Generation

Document 04 — v2 — 2026-04-07 Status: Revised experimental design Purpose: Define a rigorous, reproducible protocol for evaluating the SbD thesis of Papers 1–3

1.1 1. Research Objective

1.1.1 What Papers 1–3 Establish and Assume

Papers 1–3 form a cumulative argument and are companion papers submitted concurrently with this work; each paper’s core claims are treated as established here and are not re-derived. Reviewers are referred to the companion papers for full proofs and supporting data.

Paper 1 establishes that application security requirements across heterogeneous normative frameworks (ASVS, SSDF, CIS Controls, SLSA, CAPEC) can be reduced, without loss of coverage, to a shared ontology of 10 security slices and 234 typed instances — ControlObjectives, Practices, Mechanisms, Signals, AntiPatterns. The ontology does not introduce new requirements; it re-expresses existing normative requirements in a structured and composable form. It is a coverage-preserving normalization of external standards, fully traceable to its source frameworks. The ontology is not the source of truth — the normative frameworks are. The ontology is the structured interface through which that truth is made machine-accessible.

Paper 2 establishes that this ontology can be populated through coverage-preserving compilation of normative and empirical sources, and that 82% of apparent compliance gaps in existing framework mappings are traceability failures — not missing content. The compiled knowledge graph is complete with respect to the normative landscape it covers. This completeness is what makes deterministic retrieval possible and what justifies treating the compiled requirement set as the canonical policy expression for a given security context.

Paper 3 establishes a retrieval architecture — the MCP server — that delivers the exact set of security requirements applicable to a given code generation context, typed and provenanced, with a deterministic completeness invariant: for a given activated slice and risk level, the full applicable requirement set is always retrieved. This determinism is the mechanism under evaluation in this study.

Security-by-Design has traditionally been framed as a human-centered discipline, requiring developers to interpret and apply security practices within a given context. However, as software development increasingly incorporates automated generation systems, the locus of execution shifts from humans to machines. The limitations observed in human execution — partial coverage, inconsistency, and lack of traceability — do not disappear under automation unless explicitly mitigated. Automated systems, however, allow explicit enforcement of constraints that are inherently difficult to guarantee in human execution, enabling more consistent, complete, and auditable policy application. This creates a new requirement: security policies must be structured not only for human interpretation, but for deterministic execution by automated systems. Papers 1–3 build the infrastructure that makes this possible; this study measures whether it works.

1.1.2 The Central Claim

Execution fidelity refers to how completely, consistently, and verifiably an LLM implements a given security policy in a generated artifact. This study tests the hypothesis that increasing the strength of policy grounding improves execution fidelity, and that deterministic grounding yields reproducible improvements.

The experimental conditions represent increasing levels of grounding strength: G0 (no policy grounding), G1 (partial probabilistic grounding via similarity-based retrieval), and G2 (complete deterministic grounding via the MCP server). The underlying security policy is invariant across all three conditions — it is defined by the normative frameworks normalised in Paper 1 and compiled in Paper 2. Only the mechanism of its delivery to the LLM varies. The evaluation is not circular: the policy is held constant; the delivery mechanism is the independent variable; execution fidelity is the dependent variable.

LLMs do not lack security knowledge; they lack guarantees of completeness, consistency, and auditability under unconstrained execution. Without structured grounding, an LLM operates in an unbounded security decision space: it may apply any subset of its training-time knowledge, in any order, with no guarantee that the applicable requirements for the given context have been covered, and no mechanism to verify which requirements were considered. G1 narrows this space probabilistically but without completeness guarantees. G2 eliminates the ambiguity: ontology-grounded retrieval constrains the decision space to exactly the applicable requirement set, enabling deterministic and auditable policy application. Deterministic grounding also ensures that the same policy is delivered for the same context across runs, making improvements in execution fidelity reproducible.

This is not a tautology. If grounding trivially induced compliance — that is, if simply providing some security context were sufficient — probabilistic grounding (G1) and deterministic grounding (G2) would perform equivalently. The observed difference, if confirmed empirically, isolates the effect of grounding structure, completeness, and determinism on execution fidelity. The contribution of this work is not the definition of the policy, but the demonstration that its explicit and deterministic delivery improves how faithfully that policy is executed by an LLM.

The study does not validate the correctness or sufficiency of the policy. A hostile reviewer who characterises the evaluation as circular must first reject the premise that heterogeneous normative

frameworks can be coverage-preservingly normalised into a shared ontology — a claim established and defended in Paper 1, not assumed here.

1.1.3 What Must Be Demonstrated Empirically

Paper 3 specifies the retrieval architecture and its completeness invariant. What remains undemonstrated is whether deterministic delivery of the complete applicable requirement set produces outputs with measurably higher policy compliance — and at what cost. Specifically:

1. Does complete, deterministic policy delivery (G2) produce higher requirement-satisfaction rates than probabilistic delivery (G1) and unconstrained execution (G0)?
2. Does the resulting output carry a verifiable audit trail linking each security decision to a policy requirement?
3. Do these effects hold across artifact types (code, IaC, pipeline) or only for application code?
4. Does constraining the decision space via security grounding degrade functional correctness?
5. What is the token and latency cost of deterministic grounding relative to the alternatives?

No published benchmark measures requirement-satisfaction completeness in LLM-generated outputs against a framework-traceable, externally derived policy. **Creating this evaluation framework is itself a primary contribution of this paper**, alongside the experimental evidence.

The novelty of this work is not a better retrieval technique or a stronger prompting strategy. It is a paradigm shift: from probabilistic retrieval of security knowledge toward deterministic execution of a defined security policy. G1 approximates the policy; G2 enforces it. The difference is the object of study.

Scope of claims — what this paper does and does not prove: - *Proves*: that deterministic, complete delivery of an externally derived security policy (G2) produces higher policy execution fidelity than probabilistic delivery (G1) or unconstrained execution (G0), across code, IaC, and pipeline artifacts. - *Does not prove*: absolute software security of the generated outputs. - *Does not prove*: that the AppSec Core ontology is superior to alternative security frameworks or ontologies. - *Does not prove*: completeness of the security domain — only completeness with respect to the defined policy scope (the activated slice at L2 risk level). - *Does not prove*: that policy-compliant code is free of vulnerabilities outside the policy’s scope.

1.1.4 Constrained vs. Unconstrained Execution: Evaluation Framing

Dimension	Vulnerability avoidance (prior work)	Execution fidelity under policy (this work)
Oracle	Absence of CWE-classified patterns	Presence of policy-specified security controls
Policy source	Generic (CWE / OWASP Top 10)	External normative frameworks, normalised via ontology
Policy delivery	None / similarity-based approximation	Deterministic, completeness-invariant (G2)
Decision space	Unbounded	Bounded by activated ontology slice and risk level
Measurement	SAST/linting (negative)	Requirement-satisfaction scoring (positive)
Traceability	None	Requirement ID → normative source → implementation
Artifact scope	Application code	Code + IaC + pipeline + repo configuration

Dimension	Vulnerability avoidance (prior work)	Execution fidelity under policy (this work)
Completeness concept	“No known vulnerabilities found”	“All applicable policy requirements satisfied”
What is proved	Absence of known bad patterns	Fidelity of execution against an externally derived policy

A key insight from Paper 2: code can pass all available SAST checks while satisfying fewer than half of the applicable ControlObjectives for its security context. Execution-fidelity evaluation makes this gap observable and quantifiable.

1.1.5 Research Scope

The study evaluates three retrieval conditions (G0: no retrieval; G1: plain RAG; G2: ontology-grounded via MCP) across a task set spanning **four artifact types**: application code (Python), infrastructure-as-code (Kubernetes, Terraform, Dockerfile), CI/CD pipeline configuration (GitHub Actions), and composite tasks (code + deployment manifest). The G2 condition implements the architecture defined in Paper 3, using the MCP server as the structured retrieval interface.

1.2 2. Related Work and the SbD Evaluation Gap

1.2.1 Existing Benchmarks

SecurityEval / “Asleep at the Keyboard” [Pearce et al., IEEE S&P 2022] evaluated GitHub Copilot on 89 code generation scenarios, finding ~40% contained CWE-classified vulnerabilities. It established that LLMs generate vulnerable code at meaningful rates and defined the empirical baseline for this research thread.

LLMSecEval [Tony et al., MSR 2023] standardized this evaluation with 150 OWASP-Top-10-mapped prompts enabling reproducible cross-model comparison. A follow-on study [Tony et al., ACM TOSEM 2025] demonstrated that security-focused prompting (chain-of-thought with security examples) reduces vulnerability rates by up to 75% on some tasks — a finding that motivates the stronger structured context of G2. A recent broad empirical survey [Tihanyi et al., Empirical Software Engineering 2025] confirmed that LLM-generated code remains substantially vulnerable across models and generation strategies, reinforcing that vulnerability-avoidance measures alone are insufficient and that proactive policy enforcement is the missing layer.

CyberSecEval [Bhatt et al., Meta, arXiv:2312.04724, 2023; v2: arXiv:2404.13161, 2024] evaluates both insecure code generation and attack-assistance capability. Version 2 expanded to prompt injection and exploitation scenarios. Neither version evaluates proactive security control inclusion.

SVEN [He & Vechev, arXiv:2302.07894, 2023] uses prefix-tuning to steer generation away from vulnerability patterns. A generation-steering technique, not an evaluation framework; the SbD gap applies equally.

Sec4AI4Sec [Workshop series, IEEE EuroS&P 2023, 2024] has produced community framing around LLM-assisted security tasks but has not produced a standardized SbD evaluation dataset or framework.

IaC/pipeline static analysis (Checkov, kube-linter, tfint, hadolint, actionlint, OSSF Scorecard): Available for automated evaluation of specific artifact types. These detect known misconfigurations and insecure patterns (negative class) but cannot score whether generated IaC or pipeline code implements the full set of applicable security controls traceable to normative frameworks. They serve as partial, automated oracles for the syntactic verification class [Paper 3, §6.1].

1.2.2 The Gap

No published framework evaluates whether LLM-generated artifacts — across code, IaC, and pipeline types — **proactively implement the full set of security requirements applicable to their context, traceable to normative frameworks**. This gap is the primary motivation for the present study.

Dimension	Exists	Does Not Exist
Vulnerability avoidance (app code)	SecurityEval, LLMSecEval, CyberSecEval	—
Security-hardened generation	SVEN, structured prompting [Tony 2025]	—
IaC security static analysis	Checkov, kube-linter, tfint	IaC security <i>generation</i> benchmark
Pipeline security linting	actionlint, OSSF Scorecard	Pipeline security <i>generation</i> benchmark
Positive SbD control evaluation	None	Benchmark requiring proactive control inclusion
Framework-traceable evaluation	None	ASVS/SSDF/SLSA-traced generation oracle
Completeness-aware evaluation	None	Full requirement-family coverage scoring
Unified multi-artifact scope	None	Code + IaC + pipeline in one benchmark
Cost of security grounding	None	Token/latency overhead studies

Existing work focuses on reducing vulnerability rates in generated code, but does not address the execution guarantees of security policies under automation. As software generation becomes increasingly automated, ensuring that security policies are applied with consistent coverage and traceability becomes a central requirement — one that automated systems are uniquely positioned to enforce when properly constrained.

1.3 3. Research Questions and Hypotheses

1.3.1 Primary Research Questions

RQ1 — SbD completeness. Does ontology-grounded retrieval (G2) produce outputs that satisfy a higher proportion of applicable security requirements than baseline (G0) and plain RAG (G1)?

H1 (primary): $M_SbD(G2) > M_SbD(G1)$ and $M_SbD(G2) > M_SbD(G0)$, where M_SbD is the SbD Completeness Score (Section 4). These two contrasts are the primary inferential claims; they are tested with Bonferroni-adjusted Wilcoxon signed-rank tests (Section 10). **H1a:** The G2–G1 gap is attributable to retrieval completeness: G1 retrieves a similarity-ranked subset of applicable requirements; G2 retrieves the full applicable set by construction (verified via M_recall , Section 6). **H1b (exploratory):** $M_SbD(G1) > M_SbD(G0)$. Plain RAG is expected to improve over ungrounded generation, but this ordering is treated as exploratory — partial retrieval may not consistently outperform no retrieval across all task types. Reported descriptively; not included in the primary multiple-comparisons budget.

RQ2 — Auditability. Does G2 produce output with a more complete and verifiable audit trail than G1 and G0?

H2: $M_audit(G2)$ is non-trivially positive; $M_audit(G0) - M_audit(G1) = 0$ by structural design (no citation mechanism in G0/G1). **Note:** M_audit is reported as a qualitative property difference between G2 and G0/G1, not a statistical performance comparison. The absence of a citation mechanism in G0/G1 is a design property, not a deficit to be tested.

1.3.2 Secondary Research Questions

RQ3 — Artifact-type consistency. Are SbD completeness effects consistent across artifact types (code, IaC, pipeline, composite)?

RQ4 — Functional correctness. Does G2 maintain functional correctness comparable to G0/G1? Security grounding must not degrade the functional behaviour of the generated artifact. This is measured at two levels: structural validity (M_func_valid , tested formally) and behavioural correctness against pre-defined functional test cases ($M_func_behavioral$, reported descriptively). The concern is regression: if G2’s security grounding causes the LLM to produce functionally broken outputs, this should be observable in the per-task pass rates.

RQ5 — Cost. What is the token and latency overhead of G2 relative to G0 and G1? What is the cost per unit of SbD improvement?

1.3.3 Exploratory Research Questions

RQ6 — Cross-LLM consistency. Are the effects observed in RQ1–RQ2 consistent across LLMs? (Phase 2.)

RQ7 — Tool-detected findings. Does G2 produce output with fewer SAST/Checkov/actionlint findings than G0/G1 (standard vulnerability-avoidance dimension), or does it trade requirement completeness against tool-detectable issues?

1.4 4. The SbD Evaluation Framework

This section defines the evaluation harness. It is a primary contribution of the study independent of experimental results.

1.4.1 4.1 Evaluation Principles

1. **Positive oracle, not negative filter.** Evaluation asks “are required controls present?” not “are known bad patterns absent?” Automated tools serve as safety checks and partial oracles, not primary evaluation mechanisms.
2. **Pre-annotated, task-specific requirement sets.** Each task has a pre-annotated ground-truth requirement set R — the AppSec Core ControlObjectives and Practices applicable to that task at L2 risk level. This eliminates intent-parser uncertainty as a confound and makes the evaluation independent of the retrieval mechanism being evaluated.
3. **Multi-tier measurement.** Automated tool checks cover syntactic and known-pattern classes. Structured expert scoring covers semantic and positive-assurance classes. The two tiers are combined into M_SbD but reported separately.

1.4.2 4.2 SbD Completeness Score (M_SbD)

The primary metric is the SbD Completeness Score, defined per generated sample:

$$M_SbD = |\{r \in R : \text{satisfied}(r, \text{output})\}| / |R|$$

where: - R is the pre-annotated applicable requirement set for the task, derived from external normative frameworks (ASVS, SSDF, CIS Controls, SLISA, CAPEC) via the AppSec Core ontology. R is not defined by this study; it is the coverage-preserving projection of external standards onto the task’s security context and risk level. - $\text{satisfied}(r, \text{output})$ is evaluated by the multi-tier oracle (Section 4.3) - output is the generated artifact (code, IaC, config, or composite)

Scoring criteria for $\text{satisfied}(r, \text{output})$ (binary), from Paper 2, Definition 1: - *Scope overlap*: the output addresses the security concern identified by r - *Actionability*: the output implements the requirement, not merely references the topic - *Syntactic pre-screening*: where an automated check for r exists, it must pass

$M_SbD \in [0, 1]$. Completeness is defined relative to the specified policy scope — the set of requirements applicable to this artifact type, at L2 risk level, within the activated ontology slice. The metric makes no claim about requirements outside that scope, nor about the correctness or sufficiency of the policy itself. Aggregated across tasks, M_SbD is reported as median \pm IQR per group, with per-task and per-artifact-type breakdowns.

1.4.3 4.3 Multi-Tier Tool Oracle

Tier	Tools	Artifact types	Verification class
Tier-A — Code SAST	Semgrep (Python), Bandit	Python code	Syntactic (vulnerability patterns)
Tier-B — Code SCA	pip-audit	Python code	Dependency vulnerabilities
Tier-C — IaC analysis	Checkov, kube-linter, tflint, hadolint	K8s YAML, Terraform, Dockerfile	Syntactic (IaC misconfigurations)

Tier	Tools	Artifact types	Verification class
Tier-D — Pipeline analysis	actionlint	GitHub Actions YAML	Syntactic (workflow security)
Tier-E — Repo hygiene	OSSF Scorecard (partial)	Repo configuration tasks	Structural (branch protection, secret scanning)
Tier-F — Expert scoring	Structured checklist (per task)	All artifact types	Semantic (requirement satisfaction)

Tool tiers Tier-A through Tier-E produce automated binary pass/fail per check and cover the **syntactic verification class**: known bad patterns, known misconfigurations, and structural hygiene properties that reduce to deterministic rule matching. These feed into: - **M_tool**: proportion of automated checks passed per sample (secondary metric, RQ7) - **M_SbD pre-screening**: requirements with an associated automated check must pass that check to count as satisfied

Expert scoring (Tier-F) is necessary because requirement satisfaction in the **semantic verification class** cannot be reduced to pattern matching. Whether a generated authentication endpoint correctly implements caller identity verification, or whether a Terraform module correctly applies least-privilege access scoping, requires judgment against the requirement’s intent — not merely detection of a known insecure pattern. Automated tools can confirm the absence of known bad patterns; they cannot confirm the presence of correct security behaviour. Tier-F covers this gap. Evaluators receive artifacts with citation strings stripped (Section 8, Blinding).

1.4.4 4.4 Audit Trail Score (M_audit)

$$M_audit = |\{c \mid citations(output) : valid(c) \wedge in_retrieved(c) \wedge satisfied(c, output)\}| / |R|$$

where `citations(output)` are extracted by the citation parser (e.g., `// Satisfies: VAL-003`), and each cited requirement is validated against three criteria: 1. The ID exists in the knowledge graph (validity) 2. The ID was in the retrieved set for this (task, run) (provenance) 3. The code near the citation actually satisfies the requirement (cross-referenced with M_SbD result for r)

M_audit is a G2-specific metric. For G0/G1, it is structurally near-zero — reported as a qualitative architectural difference, not a performance result.

1.4.5 4.5 Retrieval Recall Validation (M_recall)

To validate H1a, a retrieval recall check is run per (task, group):

$$M_recall = |retrieved(task, group) \cap R| / |R|$$

For G2, M_recall is expected to be 1.0 by the completeness invariant of Paper 3 (all requirements in the activated slice are retrieved). For G1, M_recall reflects the coverage of similarity-based top-k retrieval. This is a validation check, not a hypothesis — it quantifies the structural retrieval advantage that H1a claims drives the M_SbD difference.

1.5 5. Experimental Conditions

1.5.1 Primary Groups

Group	Label	Description
G0	Baseline	LLM generates from prompt only. No retrieval, no security context.
G1	Plain RAG	LLM receives top-k (k=10) chunks retrieved by vector similarity (text-embedding-3-small) from the same corpus, without structured metadata.
G2	Ontology-grounded	Full architecture from Paper 3: slice activation → MCP-mediated structured retrieval → typed context with provenance and epistemic weights → citation instructions injected as part of MCP-structured context. The base task prompt is identical across G0, G1, and G2; citation instructions are not present in the base prompt and are not visible to G0/G1.

G2 architecture note: G2 uses the MCP server defined in Paper 3 as the retrieval interface. The MCP server is the evaluated artifact. Token consumption is measured per context injection and per MCP call (Section 7, Cost Model). No alternative implementation is used.

1.5.2 5.1 G2: MCP Server — Technical Summary

The Model Context Protocol (MCP) is an open client-server standard for structured context delivery to LLMs, released by Anthropic in November 2024 and now governed by the Linux Foundation’s Agentic AI Foundation. In this study, the MCP server exposes the AppSec Core knowledge graph (Papers 1–2) as a deterministic retrieval service. Its architecture and completeness invariant are specified in full in Paper 3; the following summarises the properties relevant to the experimental design.

The completeness invariant (Paper 3): for any valid activation tuple (slice set, risk level, artifact type), the server returns exactly the pre-compiled applicable requirement set R via ontology graph traversal — no similarity search, no top-k cutoff, no probabilistic component. $M_recall(G2) = 1.0$ holds by construction and is verified per run; any deviation aborts the run as an integrity failure. The MCP server source code, graph snapshot, and deployment configuration are published with the study data.

Structural differences between G1 and G2:

Dimension	G1 (Plain RAG)	G2 (MCP Server)
Retrieval mechanism	Vector similarity (text-embedding-3-small, top-k=10)	Ontology graph traversal + slice activation

Dimension	G1 (Plain RAG)	G2 (MCP Server)
Completeness guarantee	None — $M_recall < 1.0$ by design	1.0 by construction (Paper 3 invariant)
Response structure	Unstructured text chunks	Typed JSON with provenance, weights, citation templates
Determinism	Non-deterministic in principle; may vary across embedding model versions	Fully deterministic — same input \rightarrow identical output
Provenance	None	Full normative trace (ASVS/SSDF/CIS/SLSA/CAPEC IDs)
Citation support	None	Explicit citation templates enforced in prompt
Context ordering	Similarity rank	Epistemic weight (strong-first)

1.5.3 Ablation Groups (Phase 2)

Group	Ablation	Addresses
G2a	No epistemic weighting	Contribution of weight labels to M_SbD
G2b	No requirement IDs	Contribution of explicit IDs to M_audit
G2c	No verification feedback	Contribution of the verification loop
G0s	Security-prompted baseline	Generic security instruction without retrieval

Ablations are deferred to Phase 2.

1.5.4 LLMs

Model	Role
Claude Sonnet 4.6	Phase 1 primary
GPT-4o	Phase 2
Llama 3.1 70B	Phase 2 (open-source control)

1.6 6. Task Set and Benchmark

1.6.1 Task Design Principles

1. Tasks span four artifact types: application code, IaC, CI/CD pipeline, and composite.
2. Each task has a pre-annotated ground-truth requirement set R at L2 risk level.
3. All tasks in the primary study use **L2 risk level**. L3 variants are deferred to Phase 2 as a sensitivity analysis.

4. For tasks overlapping with LLMSecEval, the prompt is reused with an SbD pre-annotation layer added.

1.6.2 Task Set

#	Task	Artifact type	Primary slice(s)	Risk	Source	Primary RQs
T1	API end-point accepting JSON input	Code	ACO-IVF	L2	LLMSecEval	RQ1, RQ7 aligned
T2	Authentication end-point with JWT	Code	ACO-IAT	L2	LLMSecEval	RQ1, RQ7 aligned
T3	Database query with user input	Code	ACO-IVF	L2	LLMSecEval	RQ1, RQ7 aligned (L2 variant)
T4	File upload handler	Code	ACO-IVF	L2	LLMSecEval	RQ1, RQ7 aligned
T5	Kubernetes deployment manifest	IsC	ACO-SPC + ACO-RPR	L2	Custom	RQ1, RQ3
T6	GitHub Actions CI/CD workflow	Pipeline	ACO-SCBI + ACO-RPR	L2	Custom	RQ1, RQ3
T7	Secret injection in application config	Code	ACO-SPC	L2	Custom	RQ1

#	Task	Artifact type	Primary slice(s)	Risk	Source	Primary RQs
T8	Service-to-service authenticated API call	Code	ACO-ITS	L2	Custom	RQ1
T9	Logging middleware with user-sourced data	Code	ACO-SLG	L2	Custom	RQ1, RQ3
T10	Container image Dockerfile	IaC	ACO-SPC + ACO-SCBI	L2	Custom	RQ1, RQ3
T11	Terraform module for cloud storage	IaC	ACO-SPC	L2	Custom	RQ1, RQ3
T12	Composite API endpoint + K8s deployment	Composite	ACO-IVF + ACO-SPC	L2	Custom	RQ1, RQ3

Notes: - T3 uses an L2 variant of the LLMSEval database-query prompt (original is L3). The L2 variant is annotated independently. No L3 tasks appear in Phase 1. - T12 tests whether G2 maintains SbD coherence across a multi-artifact generation where security controls in the application layer and the deployment layer must be consistent. M_SbD for T12 is computed per component (API endpoint and K8s manifest separately) and reported as mean across components; the joint score (both components must satisfy their respective requirements) is reported as a supplementary metric to capture cross-layer coherence. - For LLMSEval-aligned tasks (T1–T4), prompt text is reused for comparability; the SbD annotation layer is additive.

1.6.3 Per-Task Ground Truth Package

Each task includes: - `task.md` — prompt text, identical across groups - `requirement_set.yaml` — pre-annotated R (AppSec Core IDs, risk level, applicable slice(s)) - `automated_checks.yaml` — tool checks executable against the generated output (tool, command, expected result) - `scoring_criteria.md` — per-requirement scoring guidance for expert evaluators - `expected_controls.md`

- what a well-implemented output should include (not visible to LLM) - **functional_tests/**
- pre-defined minimal functional test suite (happy path + error case); registered before the experiment; used to compute $M_{\text{func_behavioral}}$

1.7 7. Variables and Metrics

1.7.1 Independent Variables

Variable	Levels	Type
Retrieval mode	G0, G1, G2	Primary IV
Task	T1–T12	Blocking variable
LLM	Claude Sonnet 4.6 (Phase 1); +GPT-4o, Llama 70B (Phase 2)	Secondary IV
Risk level	L2 fixed; L3 in Phase 2 sensitivity analysis	Controlled

1.7.2 Primary Metrics

Metric	ID	Definition	RQ
SbD Completeness Score	M_{SbD}	$ \text{satisfied requirements} / R $	RQ1

1.7.3 Secondary Metrics

Metric	ID	Definition	RQ
Audit trail completeness	M_{audit}	Valid, provenanced, accurate citations / $ R $	RQ2
Tool check pass rate	M_{tool}	Automated checks passed / total applicable checks	RQ7
Structural validity	$M_{\text{func_valid}}$	Artifact is syntactically correct and minimally deployable (binary)	RQ4
Functional correctness	$M_{\text{func_behavioral}}$	Proportion of pre-defined functional test cases passed	RQ4
Retrieval recall	M_{recall}	$ \text{retrieved } R / R $ per group	H1a validation

1.7.4 Cost Metrics

Metric	ID	Definition
Context tokens per task	T_{ctx}	Tokens in injected context (G2: structured; G1: top-k chunks; G0: none)
MCP calls per task	N_{mcp}	Number of MCP server interactions per generation (G2 only)

Metric	ID	Definition
Tokens per MCP call	T_mcp	Mean tokens per MCP request+response pair (G2 only)
Total token overhead	T_overhead	$T_{ctx} + N_{mcp} \times T_{mcp}$ vs. G0 baseline
Overhead ratio	R_overhead	$T_{overhead}(G2) / T_{overhead}(G0)$
Latency overhead	L_overhead	Wall-clock time differential $G2 - G0$ (ms)
Cost-benefit ratio	CB	$\Delta M_SbD(G2 - G0)$ per 1,000 additional tokens

Cost metrics are fully automated via API call logging and timing. The CB ratio is the principal cost-benefit summary reported for RQ5.

1.7.5 Control Variables

- Same prompt text across G0, G1, G2 for each task
- Same decoding: temperature = 0, same max_tokens, same stop sequences
- Same context budget: G1 and G2 receive context up to the same token limit; actual distributions reported
- Same corpus: G1 uses flat chunks, G2 uses structured index from the same source
- Citation strings stripped from all outputs before expert M_SbD scoring (Section 8)
- Same evaluation tool versions (pinned; reported in appendix)

1.8 8. Measurement and Instrumentation

1.8.1 M_SbD Scoring Protocol

Step 1 — Automated pre-screening. For each (task, generated output): 1. Run applicable tool checks from the task’s `automated_checks.yaml` 2. Requirements with a linked automated check must pass that check to be scored as satisfied 3. Tool findings are logged separately as M_tool

Step 2 — Citation stripping (blinding). Before expert scoring, all citation strings (e.g., // Satisfies: VAL-003, # ACO-SPC) are stripped from generated outputs by the orchestrator. Evaluators receive artifacts without group labels or citation markers. This prevents evaluators from inferring the retrieval condition from the output content.

Step 3 — Expert scoring. For each requirement $r \in R$, the evaluator records: satisfied (1) or not satisfied (0), using `scoring_criteria.md`. Scoring is binary; partial implementation counts as not satisfied (conservative; acknowledged as a limitation).

Step 4 — Aggregation. $M_SbD = |\text{satisfied}| / |R|$.

Inter-rater reliability: 20% of samples (randomly selected, stratified by task and group) are independently scored by a second evaluator. Cohen’s kappa is computed and reported with its 95% bootstrap confidence interval; at $n = 22$ the point estimate has meaningful uncertainty and the CI is the interpretable quality indicator. A kappa point estimate 0.70 with a lower CI bound 0.60 is the acceptance threshold. If this threshold is not met, scoring criteria are revised and the affected samples re-scored. **Calibration protocol:** the kappa check and any criteria revision are completed before the research team examines aggregated group-level results; this sequencing is recorded in the

experimental log. A two-task pilot scoring session (any two tasks from T1–T12, randomly selected) is conducted with both evaluators before full execution; criteria are refined if pilot kappa falls below threshold. This pilot is logged but its scores are discarded before the full run begins.

Evaluator selection and independence: Expert evaluators are application security practitioners with a minimum of five years of hands-on experience in secure software development or security engineering, evidenced by peer-reviewed publications or equivalent professional output. Evaluators must have had no involvement in the development of the AppSec Core ontology or the SbD-ToE knowledge graph. Collaborators on other publications by the same research group are eligible provided this condition is met. Evaluator backgrounds, their relationship to the research, and any disclosed affiliations are reported in full in the paper. The primary bias control is procedural — citation stripping, absent group labels, pre-defined per-requirement scoring criteria — rather than relying solely on evaluator independence. The inter-rater kappa is the empirical quality indicator: agreement 0.70 demonstrates that the scoring criteria are sufficiently operationalized to produce consistent results regardless of evaluator background.

1.8.2 M_audit Scoring Protocol

Applied to G2 samples after M_SbD scoring is complete. The citation parser extracts requirement IDs from the original (pre-stripped) output. Each extracted ID is validated: 1. ID exists in the knowledge graph (automated lookup) 2. ID was in the retrieved set for this (task, run) (log comparison) 3. Requirement r identified by the ID was scored as satisfied in M_SbD scoring (cross-reference)

M_audit is reported descriptively for G2. No statistical comparison with G0/G1 is performed (structurally near-zero by design).

1.8.3 M_recall Validation

For each (task, group, repetition), the retrieval log records the set of requirements retrieved (G1: chunks mapped to requirement IDs post-hoc by string match; G2: directly from MCP response). $M_recall = |\text{retrieved} \cap R| / |R|$. Reported per group as mean \pm SD. This validates H1a without being a primary experimental metric. The G1 mapping is an approximation: chunks that satisfy a requirement without explicitly naming its ID are counted as misses, making G1 M_recall a lower bound on its true recall. If anything, this conservative approximation inflates the apparent G1–G2 recall gap; the G2 structural advantage over G1 is therefore not overstated by this measurement choice.

1.8.4 M_func Scoring Protocol

M_func has two distinct layers, both necessary and reported separately.

Layer 1 — Structural validity (M_func_valid): Is the artifact syntactically correct and minimally deployable? This confirms that security grounding did not produce a structurally broken output.

Artifact type	Check	Method
Code (T1–T4, T7–T9)	Parses without syntax errors; imports resolve	Python AST parse + pip check

Artifact type	Check	Method
IaC — K8s (T5)	<code>kubect1 apply --dry-run=client</code> passes	Automated
IaC — Terraform (T11)	<code>terraform validate</code> passes	Automated
IaC — Dockerfile (T10)	<code>docker build</code> succeeds	Automated
Pipeline (T6)	<code>actionlint</code> passes without errors	Automated
Composite (T12)	Both components pass their respective checks	Automated

Layer 2 — Functional correctness (M_func_behavioral): Does the artifact do what it was asked to do? This measures whether security grounding degraded the functional behaviour of the generated output — e.g., input validation so strict it rejects valid inputs, authentication that breaks the happy path, error handling that swallows exceptions. Each task has a pre-defined minimal functional test suite (2–3 test cases: happy path + basic error case), specified in `functional_tests/` as part of the ground truth package and registered before the experiment.

Artifact type	Functional test	Method
Code (T1–T4, T7–T9)	HTTP call with valid input → expected status + response body; HTTP call with invalid input → expected error response	Automated test runner against a local harness
IaC — K8s (T5)	Manifest deploys to a test cluster; pod reaches Running state; liveness probe responds	Automated against a local Kind cluster
IaC — Terraform (T11)	<code>terraform plan</code> produces the expected resource count and type against a mock provider	Automated (tfmock or localstack)
IaC — Dockerfile (T10)	Built image starts; container responds to a health-check command	Automated (<code>docker run</code> + probe)
Pipeline (T6)	Workflow executes the expected job sequence in a dry-run environment (act)	Automated
Composite (T12)	API endpoint passes its functional tests; K8s manifest passes its deployment test	Automated

M_func_behavioral is reported as **proportion of functional tests passed per sample**, per group and per task. No formal hypothesis test is applied; the metric is a regression detector. Any task where G2’s pass rate falls 10 percentage points below G0’s is flagged and described qualitatively. If such regressions are observed, the cost-benefit analysis (RQ5) accounts for them explicitly.

Note on separation from M_SbD: A K8s manifest that deploys successfully but lacks `securityContext` passes both M_func layers and fails the relevant M_SbD requirement. Whether the LLM *included* a security concern is M_SbD. Whether the LLM *broke* the functional behaviour is M_func_behavioral. The two are independent.

1.8.5 Cost Measurement

All API calls are logged via an instrumented orchestrator recording: timestamp, model, group, task, repetition, prompt tokens, completion tokens, MCP call count (G2 only), tokens per MCP call, wall-clock time per call. Monetary cost is computed post-hoc using API pricing at the time of the experiment (pricing version reported). The CB ratio is computed as $\Delta M_SbD / (T_overhead / 1000)$.

1.9 9. Experimental Controls

1.9.1 Fairness Controls

Control	Implementation
Same prompt text	Identical across G0, G1, G2 for each task
Same decoding parameters	Temperature = 0; same max_tokens
Same context budget	Both G1 and G2 subject to the same token limit; actual distribution reported
Same corpus	G1: flat chunks from the same source data as G2's structured index
Evaluator blinding	Citation strings stripped; group labels absent from score sheets
Tool version pinning	All SAST/IaC/pipeline tool versions recorded and fixed for the duration of the experiment

1.9.2 Risk Controls

Risk	Mitigation
Evaluator inference from citation presence	Citation strings stripped from all artifacts before scoring
Annotation bias (ontology authors annotate R)	20% of requirement sets reviewed by an independent security practitioner; inter-annotator agreement reported
Corpus contamination (T1–T4, LLMSecEval-aligned)	Contamination is symmetric across G0/G1/G2 (all groups use the same prompts); acknowledged as a limitation. Prompt sensitivity is similarly symmetric: any LLM-specific prompt response patterns apply equally across groups. Per-task variance is reported to detect task-specific outliers.

Risk	Mitigation
Structural self-reinforcement (G2 corpus R annotation source)	G2’s knowledge graph and R are both derived from the same normative sources. This structural advantage is explicit: the claim is that complete, typed retrieval produces better SbD outcomes. The retrieval recall metric (M_recall) quantifies the retrieval-completeness component independently.
Context budget asymmetry	If G2 structured context is shorter than G1’s top-k chunks, remaining budget is not filled on either side; token distributions per group are reported

1.10 10. Statistical Analysis Plan

1.10.1 Design

Repeated-measures: 3 groups \times 12 tasks \times 3 repetitions (Phase 1). Task is the blocking variable. Results are reported per-task and aggregated.

1.10.2 Primary Analysis (RQ1)

Overall comparison: Friedman test on M_SbD across G0, G1, G2. Non-parametric repeated-measures; appropriate for small within-task samples (n=3 reps) and bounded metric.

Primary pairwise contrast (G2 vs. G1): Wilcoxon signed-rank test on the 36 paired observations (12 tasks \times 3 repetitions), treating each (task, repetition) pair as an independent observation. Bonferroni-adjusted for 2 primary pairwise comparisons (G2 vs. G1, G2 vs. G0): adjusted $\alpha = 0.025$. Independence of repetitions within the same task is a declared assumption, defensible given T=0 and independent API calls; residual within-task correlation is reported as a robustness check. Per-task breakdowns are reported separately to confirm the effect is not driven by a subset of tasks.

Effect size: Matched-pairs rank biserial correlation r for each pairwise comparison.

Presentation: Median M_SbD \pm IQR per group, per task and overall. Box plots.

1.10.3 Sample Size and Power

Phase	Samples / group	Test	Minimum detectable effect ($\alpha=0.05$, power=0.80)
Phase 1	36 (12 tasks \times 3 reps)	Wilcoxon r (one-tailed)	0.42 (medium-large)
Phase 2	60 (12 tasks \times 5 reps)	Wilcoxon r (one-tailed)	0.33 (medium)

Papers 1–3 establish mechanistic grounds for expecting a large effect (G2 retrieves all applicable requirements; G1 retrieves a subset). Phase 1 is powered for large effects. The paper states explicitly that Phase 1 findings are framed for large-effect claims; medium-effect claims require Phase 2 replication.

1.10.4 Secondary and Exploratory Analyses

- **Per-artifact-type (RQ3):** Descriptive breakdown of M_SbD by artifact type. No hypothesis test in Phase 1 (insufficient power per stratum).
- **M_func_valid (RQ4):** Proportion of structurally valid outputs per group. Fisher’s exact test for G2 vs. G0.
- **M_func_behavioral (RQ4):** Reported descriptively — functional test pass rate per group and per task. No formal hypothesis test. Any task where G2’s pass rate is 10 percentage points below G0’s is flagged as a potential grounding-induced regression and analysed qualitatively. This threshold is a reporting convention, not a statistical claim.
- **Cost (RQ5):** Mean T_overhead \pm SD per group. CB ratio reported descriptively. No hypothesis test.
- **M_recall (H1a):** Descriptive per group (expected: G2 = 1.0, G1 < 1.0). Reported as validation, not hypothesis test.

1.10.5 Multiple Comparisons Budget

Primary: 2 pairwise (G2 vs. G1, G2 vs. G0) on M_SbD \rightarrow Bonferroni = 0.025. One additional primary test: M_tool, same 2 pairwise contrasts \rightarrow Bonferroni = 0.025. Per-task and per-artifact-type analyses are exploratory; labeled as such; no correction applied.

1.11 11. Threats to Validity

1.11.1 Internal Validity

Threat	Mitigation
Evaluator inference from citation markers	Citation strings stripped before M_SbD scoring
Annotation bias (authors annotate R) Temperature stochasticity	20% independent review; agreement (kappa) reported T=0 minimizes variation; N=3 reps; residual variance reported
Context budget asymmetry	Token budget constant; actual distributions reported per group

1.11.2 Construct Validity

Threat	Mitigation
M_SbD runtime security	Expert-scored requirement satisfaction is a proxy for implementation completeness, not a runtime security guarantee. M_tool (SAST/Checkov/actionlint) provides a complementary negative-class check. Neither alone is sufficient; the paper makes both claims explicitly.

Threat	Mitigation
R annotation completeness	Pre-annotated R reflects AppSec Core’s coverage at L2. If the ontology under-covers a domain, M_SbD under-estimates the true SbD gap. Acknowledged as a limitation tied to the ontology’s scope (Paper 1).
Binary satisfaction scoring	Partial implementation is not captured. A requirement is satisfied only if fully implemented. This is a conservative and transparent choice; acknowledged explicitly.

1.11.3 External Validity

Threat	Mitigation
Single ontology (SbD-ToE)	SbD-ToE integrates 10+ normative frameworks (Papers 1–2); it is representative of the normative landscape rather than idiosyncratic. Replication with other ontologies is future work.
Language and artifact scope	Python (application code), YAML (K8s, GitHub Actions), HCL (Terraform), Dockerfile. Other languages and artifact types are Phase 2 / future work.
LLM version sensitivity	Results are specific to model versions tested; model IDs and API versions are pinned and reported.
Task representativeness	12 tasks cover 7 of 10 AppSec Core slices. Slices ACO-ATB (Architecture Trust Boundaries), ACO-TMR (Threat Modeling), and ACO-TSV (Testing/Security Validation) are not covered in Phase 1. These slices require architectural reasoning and multi-component judgment rather than localized implementation patterns; LLMs generally perform better on pattern-based requirements than on reasoning-based ones. Their exclusion may cause Phase 1 to understate the G2 advantage on the harder requirement classes, as G2’s completeness invariant is likely most consequential where requirements are complex and non-obvious. Phase 1 findings should therefore be read as conservative lower bounds on the full-scope effect. Full slice coverage is Phase 2.

1.11.4 G2-Specific Validity

Threat	Mitigation
Apparent circularity	G2’s retrieval context and the requirement set R are both derived from the same normative frameworks via the ontology. This is not circularity — R is an externally grounded policy (traceable to ASVS, SSDF, CIS, SLSA, CAPEC); G2 delivers that policy deterministically; the study measures execution fidelity against it. Rejecting this as circular requires rejecting the premise that normative frameworks can be coverage-preservingly normalised — a claim established in Paper 1. The structural advantage of G2 over G1 is quantified via M_recall and is the claimed mechanism, not a confound.
MCP implementation fidelity	All MCP system instructions, retrieval queries, and context construction are logged, published with the study data, and mapped to the Paper 3 specification sections they implement.

1.12 12. Expected Paper Structure

1.12.1 Recommended: Single Comprehensive Paper

A single paper covering RQ1–RQ5 is the recommended structure.

Section	Content
Introduction	The SbD evaluation gap; why vulnerability avoidance is insufficient
Related Work	SecurityEval, LLMSecEval, CyberSecEval, SVEN, IaC tools (Section 2)
The SbD Evaluation Framework	Contribution 1: M_SbD, multi-tier oracle, task ground-truth packages
Experimental Design	Groups, tasks, LLMs, controls
Results: SbD Completeness	RQ1 findings (primary)
Results: Auditability	RQ2 findings
Results: Consistency and Correctness	RQ3, RQ4
Results: Cost	RQ5 cost-benefit analysis
Discussion	Effect sizes, limitations, Phase 2 agenda
Ethical Considerations	See below
Appendix: Reproducibility Statement	See below

Venue: ICSE, FSE, IEEE S&P (security track). Phase 1 results alone (108 samples, 1 LLM) are sufficient for a workshop or NIER submission. The full study (Phase 2, multi-LLM) targets a primary venue.

Ethical Considerations (to appear in paper): The study generates code and configuration artifacts under security-focused prompting conditions. No task is designed to produce exploitable

artifacts; all tasks target defensive controls (input validation, authentication, secrets management, etc.). Generated outputs are evaluated and discarded; no artifacts are deployed or published. If a generation condition unexpectedly produces artifacts with critical vulnerabilities not present in the baseline, these are reported descriptively and responsibly — not published as functional exploits. Evaluator participation is voluntary; evaluator identities are disclosed in the paper with consent. No human subjects data is collected beyond scoring judgments on generated artifacts.

Appendix: Reproducibility Statement (to appear in paper): The following materials are published alongside the paper: (1) all 12 task ground-truth packages (`task.md`, `requirement_set.yaml`, `automated_checks.yaml`, `scoring_criteria.md`, `expected_controls.md`, `functional_tests/`); (2) the orchestrator code with logging configuration; (3) pinned tool versions and configuration files; (4) raw API call logs (tokens, timing, MCP call counts) with model version identifiers; (5) scored output copies (citation-stripped, group-blinded) and inter-rater scoring sheets; (6) the `M_func_behavioral` test harness and environment configuration. The MCP server implementation is described in Paper 3; the version used is identified by commit hash.

1.13 13. Minimal Viable Study (Phase 1)

Parameter	Phase 1	Phase 2
Groups	G0, G1, G2	+ G2a, G2b, G2c, G0s
LLMs	1 (Claude Sonnet 4.6)	3
Tasks	12 (T1–T12)	12 + additional slices
Repetitions	3	5
Primary metrics	<code>M_SbD</code> , <code>M_func_valid</code> , <code>M_func_behavioral</code> , cost	+ <code>M_audit</code> detail, ablation analysis
Samples	$12 \times 3 \times 3 = \mathbf{108}$	$12 \times 7 \times 3 \times 5 = 1,260$
Power	Large effects ($r = 0.42$)	Medium effects ($r = 0.33$)

1.13.1 Phase 1 Infrastructure Checklist

Benchmark and evaluation content: - [] 12 task ground-truth packages (`task.md`, `requirement_set.yaml`, `automated_checks.yaml`, `scoring_criteria.md`, `expected_controls.md`, `functional_tests/`) - [] Per-task functional test suite: 2–3 test cases per task (happy path + error case), pre-registered before execution

Retrieval infrastructure: - [] RAG baseline: vector store (text-embedding-3-small) over the same corpus, flat chunks without metadata, $k=10$ - [] G2: MCP server from Paper 3, instrumented for per-call token and timing logging

Static analysis pipeline: - [] Tool pipeline: Semgrep, Bandit, pip-audit, kube-linter, Checkov, hadolint, tflint, actionlint (versions pinned; recorded in appendix)

Orchestration and scoring: - [] Citation parser: extract IDs from generated output; produce stripped copy for expert scoring - [] Orchestrator: (task, group, repetition) \rightarrow generate \rightarrow log context + tokens + timing \rightarrow produce scoring copy (citations stripped, group label absent) - [] Scoring interface: per-task requirement checklist, blinded, with scoring criteria - [] `M_recall` log: for G2, log retrieved requirement IDs per call; for G1, map retrieved chunks to requirement IDs

post-hoc (approximate; documented) - [] Cost aggregation: tokens \times API pricing at experiment time \rightarrow cost per (task, group)

M_func_behavioral test environments (*specification below; formal validation and toolchain decision required before execution starts*): - [] Code tasks (T1–T4, T7–T9): local HTTP harness (e.g., FastAPI test client or pytest + httpx) capable of running generated Python endpoints and executing pre-defined request/response assertions - [] K8s tasks (T5): local Kubernetes cluster (e.g., Kind) for `kubectl apply` + pod readiness + liveness probe assertions - [] Terraform tasks (T11): mock provider environment (e.g., localstack or tfmock) for `terraform plan` resource-count and type assertions - [] Dockerfile tasks (T10): Docker daemon access; `docker build` + `docker run` + health-check probe - [] Pipeline tasks (T6): local GitHub Actions runner (e.g., `act`) for workflow dry-run and job sequence validation - [] Composite tasks (T12): both the code harness and the K8s environment above

Note: The M_func_behavioral environments listed above represent the current design intent. Each environment must be validated for feasibility, reproducibility, and isolation before the experiment begins. Alternatives (e.g., container-based isolation for the HTTP harness, a shared Kind cluster vs. per-run ephemeral clusters) are open decisions to be resolved during infrastructure setup. The test harness and environment configuration are published with the study data.

1.14 14. Open Design Decisions

Decision	Options	Current recommendation
RAG chunk size	256 / 512 / 1024 tokens	512; reported; sensitivity analysis if time permits
RAG top-k	5 / 10 / 15	10; reported
Context budget	4K / 8K / 16K tokens	8K; held constant; actual distribution reported
G2 context ordering	By weight / by relevance / by requirement family	By weight (strong first), per Paper 3 §4.2
M_SbD tie-breaking	Automated check pass required before expert scoring	Yes (conservative; explicit)
Composite task (T12) scoring	M_SbD averaged across components / joint score	Averaged; joint score as supplementary
R annotation conflict resolution	Single annotator / majority vote on conflicts	20% independent review; conflicts resolved by discussion; documented

1.15 Assumptions

1. The AppSec Core ontology and structural index are correct and stable (validated in Paper 1).

2. Pre-annotated slice activations and requirement sets per task are complete and correct at L2 risk level.
 3. The MCP server implements the retrieval architecture of Paper 3 faithfully and is stable across the experimental run.
 4. SAST and IaC tools have known but acceptable false-positive/negative rates; expert scoring compensates for gaps in both directions.
 5. Temperature = 0 provides sufficient reproducibility for between-group comparison; within-group variance is measured by N repetitions.
 6. Binary M_SbD scoring is a conservative proxy; partial implementations are conservatively counted as not satisfied.
-

1.16 References

- Pearce et al. (2022). “Asleep at the Keyboard? Assessing the Security of GitHub Copilot’s Code Contributions.” *IEEE S&P 2022*. doi:10.1109/SP46214.2022.9833571
- Tony et al. (2023). “LLMSecEval: A Dataset of Prompts for Evaluating the Security of Large Language Model Code Suggestions.” *MSR 2023*. doi:10.1109/MSR59073.2023.00084
- Tony et al. (2025). “Prompting Techniques for Secure Code Generation: A Systematic Investigation.” *ACM TOSEM*, vol. 34, no. 8, art. 225. doi:10.1145/3722108
- Bhatt et al. (2023). “CyberSecEval: A Comprehensive Evaluation Framework for Cybersecurity Risks and Capabilities in LLMs.” arXiv:2312.04724
- Bhatt et al. (2024). “CyberSecEval 2.” arXiv:2404.13161
- He & Vechev (2023). “SVEN: Controlling Code Generation with Semantic Constraints.” arXiv:2302.07894
- Tihanyi et al. (2025). “How Secure is AI-Generated Code?” *Empirical Software Engineering*, vol. 30, art. 47. doi:10.1007/s10664-024-10590-1
- Sec4AI4Sec Workshop Proceedings, IEEE EuroS&P 2023, 2024.