Linköping University | Department of Science and Technology
Master's thesis, 30 ECTS | Datateknik
2026 | LIU-ITN/LITH-EX-A--26/001--SE

When Agentic Workflows Help: Hybrid Retrieval for Test Case Recommendation over Heterogeneous Software Artifacts

När agentbaserade arbetsflöden hjälper: Hybridhämtning för testfallsrekommendation över heterogena mjukvaruartefakter

Berkay Orhan

Supervisor : Jiahui Geng
Examiner : Fredrik Heintz

External supervisor : Dimitris Rentas

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida `http://www.ep.liu.se/`.

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: `http://www.ep.liu.se/`.

© Berkay Orhan

Abstract

When changes are introduced into large-scale industrial software systems, engineers must identify which existing test cases are relevant—a task that requires bridging a semantic gap between unstructured change descriptions and test artifacts distributed across multiple tools and repositories. Retrieval-Augmented Generation (RAG) can match natural-language queries against external evidence, but standard pipelines often retrieve plausible but incorrect context, fabricate identifiers, and fail to decompose multi-step queries.

This thesis designs, implements, and evaluates an agentic hybrid retrieval assistant for this setting. The assistant combines dense semantic retrieval, sparse lexical matching, structured relational queries, and bounded graph navigation over a shared corpus of industrial test artifacts; an agentic workflow lets the system choose retrieval tools step by step, while constrained tool interfaces limit what it can query and return. The title's contrast—*when agentic workflows help*—refers to which of two agentic workflows helps under which conditions; an agentic-versus-non-agentic comparison is outside the study's scope.

The system extends the *DeepAgents* agent harness with a polyglot retrieval substrate, layered tool guardrails, a per-session virtual filesystem for offloading tool results, and a three-tier evaluation pipeline whose composite scoring protocol assesses task correctness alongside traditional information retrieval metrics. Language-model backbone, agentic workflow, and knowledge-augmentation mode vary across conditions; the extension components are held constant. The evaluation comprises 936 benchmark executions across 18 configurations (three large language models, two agentic workflows, and three agent-skill configurations) on 52 evaluation scenarios over a corpus of 11,366 industrial test cases in a telecommunications environment.

Repeated evaluations by the automated judge on identical inputs show high agreement (Cohen's $\kappa = 0.844$, Pearson $r = 0.986$), while cross-family correlations ($\rho = 0.27$–0.53) show that judge-based and retrieval-based metrics capture complementary quality signals. Agent-skill effects are model-dependent: under Plan-and-Execute, on-demand skill access improved DeepSeek-V3.2 (+11.6 percentage points) while GLM-5.1 declined (-7.7 percentage points); with one model per behavioral profile and a single run per configuration, these contrasts are directional observations, consistent with two non-exclusive candidate mechanisms (context-budget relief and autonomous skill selection). The orchestrator-worker pattern improves aggregate functional pass rate for GLM-5.1—an advantage that persists, attenuated from 15.4 to 8.2 percentage points, after excluding a model–serving interaction failure—is near-parity for DeepSeek-V3.2, and does not improve Qwen3-Coder under this criterion. Tool-call success rates range from 86% to 100%, and no configuration achieves acceptable abstention accuracy on unanswerable queries (42.9–71.4% missed-abstention rates).

These results are consistent with effective agentic retrieval depending on model-specific tool-use training, runtime reasoning behavior, and serving conditions, and they show that single-metric evaluation misses quality aspects visible only to the complementary metric family.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my supervisor Jiahui Geng for his guidance on research question formulation, project alignment, and pushing for empirical rigor throughout this thesis. I am equally grateful to my company supervisor Dimitris Rentas, who was with me at every step, providing domain expertise, practical feedback, and consistent support through our regular meetings. Finally, I thank my examiner Fredrik Heintz for his constructive input during the examination process.

1 Introduction

Large-scale enterprise software systems evolve under sustained delivery pressure. Continuous Integration and Continuous Deployment (CI/CD) practices shorten iteration cycles, but they also increase how often teams must assess the impact of each change and validate that existing behavior has not been unintentionally altered [1]. In mature CI-driven projects, regression-validation cost can be driven as much by human effort in test selection, maintenance, and failure investigation as by raw execution time [2]. In CI/CD settings, this validation effort can constrain the speed at which teams integrate and release changes [3].

Regression testing research offers techniques for reducing verification cost while preserving confidence, including selection, minimization, and diversity-based prioritization [4, 5, 6]. In practice, however, many industrial scoping decisions happen before any automated test-selection algorithm can be applied. Engineers must first identify existing tests that may be related to a requirement, defect report, or change request. They must also justify those recommendations to stakeholders. This thesis addresses that earlier, knowledge-intensive step.

Several terms recur throughout the thesis. An *artifact* is a software-engineering work object, such as a test case, requirement, or design document. A *corpus* is the full collection of such artifacts used by the retrieval system. A *trace link* is an explicit recorded relationship between artifacts, for example a requirement linked to one or more test cases. *Retrieval-Augmented Generation* (RAG) means that a language model answers by using retrieved external evidence rather than relying only on its internal training data. In this thesis, *hybrid retrieval* means combining several retrieval modes: dense retrieval for embedding-based semantic similarity, sparse lexical retrieval for keyword and exact-term matching, structured relational queries for fields and filters, and graph traversal for explicit artifact relationships. An *agentic assistant* is a system that answers by selecting and executing retrieval tools step by step before producing a final response. *Evidence grounding* means that a recommendation can be traced back to retrieved artifacts or relation paths that a practitioner can inspect.

1.1 Problem Setting

Test scope analysis as test case recommendation Regression testing cost remains a bottleneck in CI/CD workflows [5, 7]. In this thesis, *test scope analysis* denotes the activity of identifying and recommending relevant *existing* test cases when a new feature, defect report, or change request is introduced. The emphasis is on helping engineers *discover* legacy tests that may

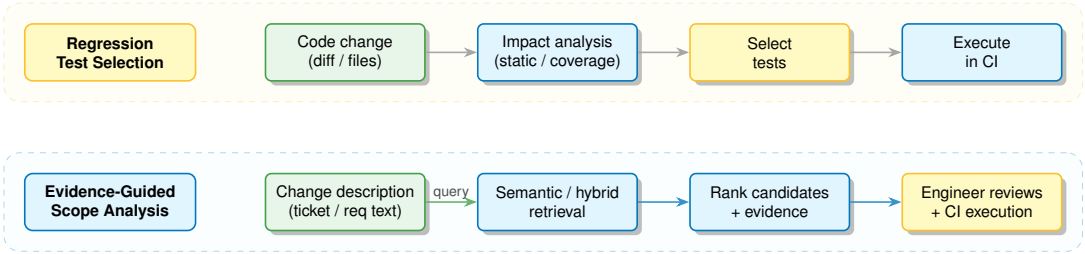


Figure 1.1: Contrast between traditional Regression Test Selection (RTS) and test scope analysis framed as semantic retrieval with evidence-backed recommendations.

cover the change, including tests not trivially discoverable through keywords or explicit trace links.

This framing differs from traditional Regression Test Selection (RTS), which typically filters a test suite for execution using program analysis or coverage-based criteria [4, 5]. While RTS is essential for execution-time efficiency, industrial scoping often begins with a semantic retrieval problem: mapping an unstructured change description to a set of tests and to the evidence that explains the connection. In the literature, this is closely related to *test case recommendation* and to traceability link recovery between requirements and tests [8].

Evaluation of such retrieval systems commonly relies on information retrieval (IR) metrics such as Normalized Discounted Cumulative Gain (NDCG), Precision@k, and Recall@k [9, 10]. In an industrial setting, however, retrieval quality alone is insufficient: practitioners also require transparent evidence for why a test was suggested, and they need predictable system behavior under ambiguous, incomplete, or noisy inputs.

Traceability, heterogeneous artifacts, and the semantic gap Large-scale organizations typically store software-engineering artifacts across multiple tools, repositories, and heterogeneous formats: requirements and backlog items, test cases and execution history, defect reports, and (where available) explicit trace links between these artifacts [11]. Trace links support change impact analysis and coverage arguments, yet in practice they are frequently missing or stale [12, 13]. When trace links are incomplete or burdensome to maintain, practitioners fall back on manual search and informal knowledge [14, 15].

A central difficulty is the *semantic gap* between natural-language change descriptions and structured test assets. A work item may state "fix race condition in handover," while test assets are described through domain-specific identifiers and step-level procedures. Classical IR approaches for trace recovery can struggle when there is limited lexical overlap, when terminology shifts across teams, or when the relevant information is distributed across multiple artifacts [12]. Recent reviews of traceability research highlight two facts: IR-based baselines are mature, but maintaining usable traces at industrial scale remains difficult in long-lived systems [13].

1.2 Motivation

In industrial continuous delivery, maintaining test confidence without excessive execution forces trade-offs between rapid feedback, acceptable risk, and resource constraints [5]. Scoping decisions face three pressures: the risk of omitting fault-revealing tests, the productivity cost of slow CI feedback when excessive test work is triggered per change [5, 3], and the need for recommendations that practitioners can inspect and trust.

Under-scoping lets regressions escape to later stages or users [5]. Over-scoping lengthens CI builds and delays feedback, reducing developer productivity [3]. When recommendations lack inspectable rationale, practitioners cannot calibrate trust in the system [16, 17].

Prior to LLMs, trace recovery and test recommendation relied on supervised classifiers (logistic regression, SVMs, tree-based models) trained on IR similarity features and structural signals to predict trace links or fault-relevant artifacts. These approaches improved over lexical baselines but depended on manual feature engineering and generalized poorly across domains with shifting technologies or sparse labeled data [12].

Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG) offer a practical mechanism for semantic matching between change descriptions and legacy tests [18, 19, 20]. At the same time, knowledge graphs provide a structured, queryable representation of relationships between artifacts (e.g., requirement–test traces, component ownership, or execution history), supporting deterministic traversal and provenance-aware reasoning [21, 22]. Hybrid approaches that combine vector-based retrieval with graph-based context expansion have been proposed under the umbrella of *GraphRAG* [23].

Naive adoption of LLM-centric retrieval in engineering workflows has known limitations. Standard RAG pipelines can retrieve plausible but incorrect context [24, 25], and generative components can hallucinate identifiers or fabricate rationales [26]. Most pipelines also rely on dense retrieval over flat text chunks, treating artifacts as independent documents.

Software-engineering artifacts are interrelated objects embedded in dependency and traceability structures. They combine natural language with identifiers, configuration signals, and explicit links. Purely semantic chunk similarity can miss structural relations and exact identifier matches central to impact analysis [12, 13]. Conversely, purely graph-based approaches are brittle when queries are underspecified free text and the schema does not encode all relevant meaning. Effective retrieval systems therefore integrate multiple knowledge types rather than relying on a single approach [25, 24]. Four design requirements follow:

(i) Hybrid retrieval over heterogeneous knowledge types. Industrial scoping draws on unstructured text, semi-structured fields, and explicit relations across tools and lifecycle stages. Integrating multiple knowledge types is central for robust retrieval in complex domains [25]. Unstructured industrial artifacts further require preprocessing and enrichment (e.g., extracting structured fields from test steps, normalizing terminology) to become reliably retrievable.

(ii) Structured representation and enrichment before retrieval. Retrieval effectiveness depends on upstream representation choices: what constitutes a retrievable unit, what metadata is preserved, and how content is indexed for sparse and dense access [24]. This thesis adopts an "enrich-first" stance: artifacts are transformed into structured, provenance-aware representations before embedding and indexing, reducing ambiguity and improving retrieval precision. LLM-driven information extraction can convert unstructured text into schema-aligned knowledge objects [27], and LLM-empowered knowledge graph construction supports entity and relation extraction for structured reasoning layers [28].

(iii) Agentic workflow with explicit reasoning trajectories and constrained tool interfaces. Scope analysis queries are frequently underspecified and multi-step (e.g., "Does this change affect billing, and if so, which legacy gateway tests cover it?"). A static retrieve-and-summarize pipeline cannot decompose such intents, perform intermediate lookups, or adapt based on partial evidence. Agentic architectures handle multi-step decomposition by iteratively selecting retrieval actions (tool calls), incorporating feedback, and maintaining a reasoning trajectory that links intermediate evidence to the final recommendation [29]. The trajectory exposes how conclusions derive from retrieved artifacts, supporting interpretability. LiSSA, a link-recovery system for software traceability, illustrates this direction by combining retrieval with explainable outputs [30].

(iv) Architectural constraints and polyglot persistence. In industrial software-engineering organizations, artifacts are not merely technically heterogeneous; they are organizationally distributed. Requirements live in backlog systems (Jira, Azure DevOps), test cases in test-management platforms (TestRail, qTest, internal equivalents), source code in version control (Git), defects in issue trackers, design documents in wikis or documentation portals, and execution history in CI dashboards (Figure 1.2).

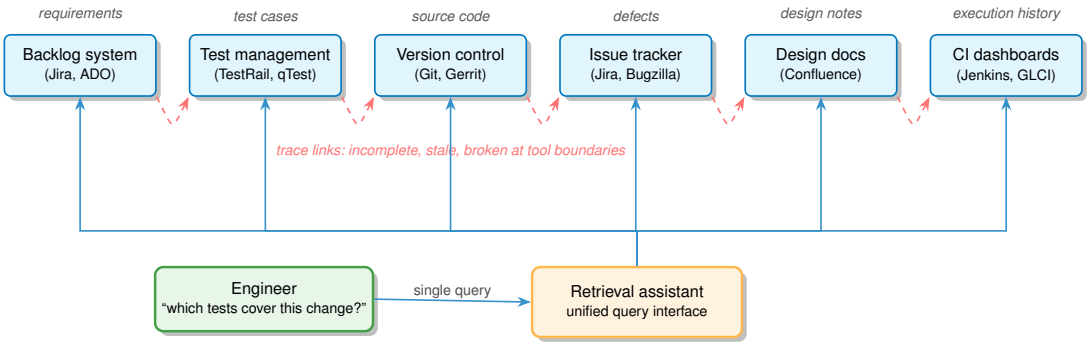


Figure 1.2: A typical industrial software-engineering data landscape. Requirements, test cases, source code, defects, design documents, and execution history each reside in tool-specific stores with distinct access patterns. Trace links cross tool boundaries and are frequently incomplete. A unified retrieval interface over this landscape is the steady-state requirement that motivates polyglot persistence in this thesis.

This distribution reflects different stakeholder needs across product, QA, and engineering teams. It also reflects acquisition history, per-tool access-control and audit requirements, and the cost of migrating mature workflows. The consequence for engineers is daily context-switching across five to ten tools. Trace links often stop at tool boundaries, and cross-tool questions such as "find all test cases referencing this requirement, sorted by last failure time" are practically infeasible without a unified query layer.

Even within a single tool, access patterns vary across exact-identifier lookup, free-text search, structural traversal, and field-based filtering, each with different performance characteristics. *Polyglot persistence* [31] combines specialized data stores rather than forcing all queries into one backend. LLM interfaces over heterogeneous backends (relational, vector, graph) under a unified orchestration layer support this pattern [32]. In this setting, polyglot persistence follows from the data landscape rather than from implementation convenience. The contribution of this thesis is not new storage primitives, but the design and evaluation of a retrieval assistant built on these storage choices.

1.3 Aim

This thesis designs, implements, and evaluates an agentic retrieval assistant for test scope analysis in an industrial setting, focusing on reliability-oriented variations in agent workflow design, tool-interface constraints, and knowledge-delivery mechanisms. It investigates how workflow design and tool-interface constraints influence reliable tool use and evidence-grounded recommendations, and which language-model characteristics most affect tool-use reliability.

The resulting system, referred to as an *Agentic Hybrid Retrieval* assistant, integrates dense semantic retrieval, sparse lexical matching, structured relational queries, and bounded graph-based navigation within an agentic workflow that enforces constrained tool interfaces and explicit evidence aggregation. It retrieves relevant legacy test cases, exposes traceability evidence, and provides explainable rationales for its recommendations.

The evaluation considers retrieval effectiveness, operational reliability, and trust-oriented automated proxies such as evidence grounding, abstention behavior, and response quality. It proposes a benchmark and scoring protocol that relates IR metrics and LLM-as-a-judge scores.

Scope of the evaluation. The benchmark targets a real-world industrial corpus rather than a small synthetic dataset. The underlying knowledge base comprises 11,366 production test cases with structured metadata and 1,365 linked backlog entities (requirements, design documents,

specification items) connected through 15,222 reference links. These links form the knowledge graph over which retrieval is performed.

Evaluation is organized around 52 hand-crafted evaluation scenarios (queries, not test cases). Each scenario carries database-derived ground truth: expected artifact identifiers and entity references pulled from the canonical database, agent-reviewed reference answers (approximately one-third manually audited; Section 3.3), and known confusion traps. The scenarios are stratified, meaning deliberately distributed, across five retrieval families (lookup, search, traceability, impact, and comparison) and three complexity tiers (single-hop, multi-hop, and reasoning). A single scenario can touch anywhere from 1 to 97 test cases. An impact-analysis scenario, for example, checks whether the agent correctly identifies all 83 test cases linked to a given architectural component.

Each scenario is exercised across 18 configurations (3 language models $\times$ 2 agentic workflows $\times$ 3 knowledge-delivery modes), yielding 936 evaluation runs in total. The 52-scenario topic set is consistent with established IR evaluation practice: TREC ad hoc tracks have long used topic sets of approximately 50 queries [33] (Section 3.3 develops this argument).

Scope premises and contribution boundary. Two design premises bound the empirical scope. First, hybrid retrieval is a task-derived requirement: the five scenario families introduced in Section 3.3 jointly demand exact-identifier joins, reverse-edge enumeration, multi-hop graph access, filtered set difference, and dense–sparse fusion. No single retrieval modality covers this distribution, so polyglot hybrid retrieval is presupposed by the benchmark rather than tested by it. Second, the agentic architecture is similarly task-derived: the scenarios are designed to require iterative decomposition, intermediate lookups, and adaptive tool selection, capabilities that motivate an agentic architecture over a static retrieve-and-summarize pipeline; such a non-agentic baseline was therefore not included in the comparison, and quantifying this premise empirically is left to future work. The agentic architecture is also an established pattern for tool-mediated retrieval over heterogeneous backends [29, 32, 34]. Within these premises, the empirical contribution isolates how knowledge-augmentation mode and orchestration strategy interact across three language-model backbones. The title phrase *When Agentic Workflows Help* accordingly refers to *which* workflow—ReAct versus orchestrator-worker Plan-and-Execute—helps under which knowledge-mode and model conditions, not to an agentic-versus-non-agentic comparison. The complete execution environment around the language model is referred to as the *agent harness*, following Hashimoto [35] and Seong et al. [36]. This thesis adopts the DeepAgents harness [37] as foundation and contributes the retrieval-domain extensions detailed in Section 3.2.

1.4 Research questions

To avoid ambiguity, the following additional terms are defined as used throughout the research questions. *Representation* refers to which artifact types and fields are stored, which text fields are indexed for dense or sparse retrieval, and which relationship types are explicitly queryable with provenance. An *agent workflow* combines two design choices: how agent components are wired (for example, a single-agent loop or an orchestrator-worker system), and how reasoning and tool calls are sequenced (for example, ReAct or plan-then-execute). The workflow determines how the assistant reaches an answer, including stopping and fallback rules. The *tool interface* is the exposed set of tool actions and their input/output constraints. Finally, *language-model characteristics* are properties fixed at model selection time that may affect tool-use behavior. These include post-training methodology, architecture, effective context length, and tool-calling policy prior.

Based on these definitions, the following research questions are formulated:

- **RQ1:** How can a benchmark and scoring protocol for agentic hybrid retrieval across structured and unstructured heterogeneous artifacts be designed and validated, and to what extent do LLM-as-a-judge evaluations agree with traditional IR metrics?
- **RQ2:** How does the mode of domain-knowledge augmentation affect task correctness, retrieval effectiveness, and evidence grounding in agentic hybrid retrieval over heterogeneous industrial artifacts, and how does the agentic workflow moderate these effects?
- **RQ3:** How do language-model characteristics influence reliable tool use and evidence-grounded retrieval performance in this setting?

1.5 Delimitations

This study is delimited to the context of an industrial partner's proprietary software development environment. In terms of data, the study utilizes proprietary datasets provided by the industrial partner, including structured exports from a test management system and a requirements/backlog system, their associated trace links, and unstructured text fields such as descriptions and test instructions. The thesis reports only paraphrased examples and aggregate evaluation results. Functionally, the system focuses exclusively on *test scope analysis* (identifying relevant existing tests) and does not include automatic generation of new test cases or execution of tests. Regarding deployment, the contribution is evaluated as a decision-support assistant for engineers; full organizational rollout, governance processes, and long-term maintenance strategies are outside the scope of this thesis.

1.6 Thesis outline

Chapter 2 provides background on regression testing, traceability recovery, retrieval and representation foundations, graph-augmented retrieval, and agentic architectures. Chapter 3 describes the system design, data pipeline, retrieval interfaces, agent runtime, and the benchmark-based evaluation protocol. Chapter 4 reports quantitative results organized by research question, covering scoring protocol validation, knowledge-configuration and workflow comparisons, and cross-model reliability. Chapter 5 interprets the findings, relates them to prior work, and discusses limitations. Chapter 6 summarizes the contributions and outlines directions for future work.

2 Background and Foundations

This chapter provides background for the research questions in Chapter 1. It first establishes the software engineering context (rapid delivery, regression cost, and traceability decay), then covers retrieval, representation, graph-based, and agentic building blocks relevant to evidence-grounded recommendation under industrial constraints.

2.1 Software engineering context and problem framing

Progress and scaling pressures in modern software engineering. CI/CD practices shorten feedback loops and increase delivery velocity [38], but they intensify the need for continuous change-impact assessment through regression testing [5]. At scale, the pressure is not only to execute tests but to deliver feedback that fits within CI time budgets. Empirical studies report recurring regression costs in CI environments, including flaky tests and pipeline inefficiencies [2, 7]. Long-running builds further constrain what is feasible and motivate targeted validation [3].

Why regression cost and artifact fragmentation have intensified. Regression cost extends beyond runtime to recurring knowledge work: deciding which tests are relevant, interpreting failures, and understanding behavior across interacting components. In large organizations, team boundaries, partial system ownership, and evidence spread across heterogeneous tools (work items, change requests, defect reports, test specifications, CI history) compound this effort. Sustainable testing depends on both automation and organizational practices that keep test assets maintainable [1]. In large-scale agile settings, misalignment between requirements engineering and system testing contributes to information gaps and traceability decay [11].

Scope analysis versus classical regression test selection. RTS research selects test subsets based on program analysis, coverage, or change impact criteria to reduce execution cost [4, 5]. CI/CD strains purely analysis-driven approaches: frequent small changes, heterogeneous non-code edits, and operational constraints that make heavyweight analyses brittle [7]. This thesis frames *test scope analysis* as a *test case recommendation* and evidence-retrieval task: given a change description, identify candidate legacy tests and provide explicit evidence explaining

why those tests are plausible validators [6]. The emphasis is on retrieval effectiveness and evidence grounding, not only execution-time optimization.

Traceability as a partial signal under realistic conditions. Traceability links are rarely complete or consistently maintained and become stale as artifacts evolve [13]. Scope analysis therefore combines semantic retrieval over unstructured text with use of explicit relations where they exist. Systematic mapping studies highlight IR as a practical foundation for recovering links across heterogeneous artifact types [12]. A recurring obstacle is the *semantic gap*. Relevant information may be distributed across fields, encoded in domain-specific terminology, or expressed with limited lexical overlap. This motivates hybrid strategies that combine textual meaning, exact identifiers, and explicit relations.

2.2 Retrieval and representation foundations

Because traces are incomplete and terminology varies across teams, effective scope analysis requires complementary retrieval strategies with controllable access to heterogeneous evidence. In software engineering, queries mix descriptive language with exact identifiers (component names, ticket IDs) and constraint intent ("tests linked to requirement X"). This section covers retrieval approaches and representation practices for such mixed queries.

Lexical retrieval, sparse signals, and exact-match vocabulary. Lexical (sparse) retrieval remains essential because engineering artifacts contain exact strings that carry meaning: identifiers, abbreviations, and standardized domain terms. BM25 and full-text indexing provide robust exact-match behavior with transparent query operators [39, 10]. For scope analysis, lexical retrieval is the most dependable path for evidence tied to specific IDs, subsystem names, and enumerated labels, where semantic similarity alone is unreliable.

Dense retrieval and semantic similarity. Dense retrieval represents queries and documents as learned embeddings and retrieves candidates via nearest-neighbor similarity [40]. This maps high-level change descriptions to semantically related tests even when terminology differs, which is common when developers describe intent at a higher level than test specifications. Dense retrieval can be brittle for rare tokens and identifiers and may return plausible but irrelevant matches when domain nuance is not captured, motivating combination with lexical retrieval and evidence-oriented post-processing.

Hybrid retrieval and rank fusion. Hybrid retrieval combines sparse and dense retrievers for both lexical precision and semantic generalization. Because scores across retrieval families are not directly comparable, rank-based fusion is standard. Reciprocal Rank Fusion (RRF) combines ranked lists without score normalization [41]:

$$\mathrm{RRF}(d) = \sum_{r \in \mathcal{R}} \frac{1}{K + \mathrm{rank}_r(d)}$$

where $\mathcal{R}$ is the set of ranked lists and K is a smoothing constant (typically 60). This fusion promotes candidates supported by multiple retrieval views while reducing sensitivity to score calibration.

Structured querying and constraint-based access. Not all scope questions are answered by similarity search. Many are constraint queries over structured fields: filtering by component, ownership, timeframe, or link type. Structured querying over record-like and relationship data complements similarity search by offering deterministic retrieval paths with explicit provenance. The relevance here is not a specific query language but the role of structured

access as an auditable interface: bounded access, validation, and reproducible retrieval steps within a broader workflow.

RAG and evidence grounding. Retrieval-Augmented Generation (RAG) grounds outputs in external evidence rather than relying solely on parametric model memory [18]. For scope analysis, the generation task is decision-support explanation: recommend candidate tests, cite evidence from artifacts and relations, and communicate uncertainty when evidence is weak. RAG introduces coupled risks: retrieval may be irrelevant or incomplete, and generation may be unfaithful even when retrieval is correct (hallucinated identifiers, overstated claims, fabricated rationales). Reliability-focused RAG therefore emphasizes *evidence grounding* (recommendations traceable to explicit retrieved artifacts or relation paths) alongside retrieval quality. Robust systems operate over multiple knowledge types (unstructured text, semi-structured records, structured relations), motivating careful representation and integration choices [25].

Representation and enrichment for heterogeneous artifacts. Retrieval quality is bounded by what is indexed. Industrial artifacts are noisy, inconsistent, or overly long, and important signals may be implicit. An *enrich-first* perspective transforms artifacts into structured, normalized, provenance-aware representations before indexing. Generative information extraction produces schema-guided structured outputs from text [27]: concise summaries, tags, component hints, and candidate relations retrievable deterministically or usable as additional signals. LLM-empowered knowledge graph construction extends this to ontology definition, knowledge extraction, and entity resolution [28]. For scope analysis, enrichment both increases signal-to-noise ratio for retrieval and improves explainability by making evidence citable (linking recommendations to extracted fields or explicit relations rather than opaque text fragments).

2.3 Graph and knowledge-oriented retrieval

Knowledge graphs represent relationships between artifacts explicitly (requirement–test links, test–procedure references, defect–requirement relations) [42]. Unlike similarity-based retrieval, graph retrieval yields deterministic evidence trails as paths and neighborhoods, which is valuable when explanations must show *why* a test is recommended.

Graph-augmented retrieval (*GraphRAG*) integrates graph structure into context construction so generation can use both textual and relationship evidence [23]. For scope analysis, the benefit is structured reasoning over explicit relations: controlled context expansion, entity disambiguation, and auditable evidence chains inspectable by engineers.

Two prerequisites recur. First, the system must resolve mentions to graph nodes (entity linking), handling aliases and ambiguous identifiers [43]. Second, graph access must be bounded: depth limits, neighbor caps, and explicit traversal strategies prevent uncontrolled expansion and reduce the risk of conditioning generation on irrelevant context. These constraints matter in industrial settings where the graph is large and explanations must remain concise and verifiable.

2.4 Agentic systems, architecture, and evaluation

Scope analysis questions are multi-step: resolving ambiguous identifiers, selecting a retrieval mode, retrieving candidates across evidence sources, and assembling a grounded recommendation. Agentic systems interleave reasoning with tool invocation, meaning that they alternate between deciding what to do next and calling a tool that observes external data. ReAct (Reasoning and Acting) formalizes this alternation between reasoning and observable actions (tool calls) [29]. For industrial decision support, the motivation is not open-ended autonomy but

the ability to route between retrieval modes (lexical, dense, structured, graph) and assemble evidence trails in a controlled manner.

Execution topology and cognitive control patterns. Agent architectures can be characterized along two orthogonal axes: *execution topology*, which describes how components are wired (e.g., a single-agent loop or a central orchestrator coordinating subordinate workers), and *cognitive control pattern*, which describes the reasoning strategy governing action selection [44]. The *orchestrator-worker* topology places a central component that dynamically decomposes tasks, delegates sub-tasks to worker agents, and synthesizes their results [34]. This topology is agnostic to the cognitive pattern: the orchestrator may plan and then execute, route tasks by type, or coordinate adversarial verification. *Plan-and-execute* is a cognitive pattern in which a planner first produces a decomposed task plan and an executor carries out each step, with optional replanning between steps [45]. When plan-and-execute is implemented atop an orchestrator-worker topology, the planner serves as the orchestrator and each execution step is delegated to a worker sub-agent with its own isolated context window. This thesis evaluates both a single-agent ReAct workflow and a plan-and-execute workflow implemented as an orchestrator-worker system, enabling comparison between reactive and deliberative control under the same tool registry and guardrail constraints.

Tool interfaces and reliability mechanisms. A *tool interface* is the set of retrieval and query actions exposed to an agent and their input/output constraints. For hybrid retrieval over software artifacts, typical tools include lexical search, embedding-based search, hybrid fusion, graph neighborhood/path retrieval, and read-only structured filtering. Reliability risks (malformed queries, overly broad queries, compounding errors, ungrounded synthesis) motivate three design choices: (i) constrained access via typed schemas and bounded result sizes, (ii) evidence-first answering with explicit citations to retrieved items or relation paths, and (iii) failure-aware behavior where the assistant reduces claim strength or abstains when evidence is insufficient.

Agent virtual filesystem and workspace persistence. Multi-step agentic workflows produce intermediate and final artifacts, such as ranked lists, evidence tables, and exported reports. These artifacts accumulate across tool calls within a session. A *virtual filesystem* provides a persistent, scoped workspace where the agent can write, read, list, search, and organize them through file-oriented tool interfaces exposed by a pluggable backend [46].

This differs from transient tool return values. A normal tool result is consumed once and then discarded from the context window. An artifact in the virtual filesystem remains addressable throughout the session and can be inspected by the end-user. The DeepAgents framework supports multiple backend implementations, including in-memory state, local disk, cloud storage, and database storage [46]. For deployments requiring durability and concurrent access, the documentation recommends implementing a custom virtual filesystem over PostgreSQL or S3 [46]. Sandbox backends add isolated code execution through a shell tool within a security boundary [47]. Retrieval-focused agents that do not generate or execute code can instead use a non-sandbox filesystem backend, preserving workspace persistence without the overhead of full sandboxing.

Context engineering and working-memory management. An LLM agent's context window functions as finite working memory: every system prompt token, conversation turn, tool call, and tool result competes for the same fixed budget. Because transformer attention computes pairwise relationships across all tokens, the model's ability to accurately recall and reason over context degrades as token count grows; this phenomenon, termed *context rot*, means that context must be treated as a resource with diminishing marginal returns rather than a buffer to fill [48]. The guiding principle of context engineering is therefore to maintain the

smallest set of high-signal tokens that maximizes the likelihood of the desired outcome [48]. In retrieval-heavy workflows, this budget is consumed rapidly; a single database query or vector search can return thousands of tokens, and multi-step reasoning chains compound the problem across successive tool calls. Without active management, the agent either loses early evidence to context rot or exhausts its context window before completing the task. Three complementary strategies address this constraint [49, 48].

Offloading. Offloading targets large tool results. When a tool call produces output exceeding a token threshold, the framework writes the full result to the virtual filesystem and substitutes the context-window entry with a compact reference (file path and a short preview). The agent can then retrieve specific portions on demand through the filesystem's read and search operations rather than retaining the entire result in working memory. This converts the interaction model from push (entire result injected into context) to pull (agent reads what it needs, when it needs it). The agent maintains lightweight references and loads data into context just in time [48], making its effective working memory the full filesystem rather than the fixed-size context window.

Summarization. Summarization, also termed *compaction* [48], targets conversation history. When the accumulated context approaches the model's window limit and no further offloading is possible, an LLM-generated summary replaces older messages while preserving task intent, artifacts produced, and next steps. The original messages are written to the filesystem as a canonical record, so the agent can recover specific details through file search if needed. This allows long-running sessions to continue beyond what the raw context window would permit.

Context isolation. Context isolation targets multi-step sub-tasks. When the main agent delegates work to a sub-agent, each sub-agent operates in its own fresh context window and may consume tens of thousands of tokens through extensive tool use, but returns only a condensed result to the main agent [48]. This quarantines context-heavy work (iterative search refinement, multi-hop graph traversal, large result set processing) from the main agent's working memory.

These three strategies operate at different granularities (individual tool results, conversation history, and task-level delegation) and compose naturally: a sub-agent can offload its own retrieval results and be summarized independently, while the main agent's context remains clean. For retrieval-intensive agentic systems, this layered approach is a prerequisite for sustaining multi-step evidence assembly within practical context-window limits.

Agent skills and structured procedural knowledge. Tool interfaces define *what* an agent can do; equally important is the procedural knowledge governing *when* and *how* to use each tool. *Agent skills* are modular packages of curated procedural knowledge, such as domain conventions, retrieval strategies, and response policies. They augment agent behavior at inference time without modifying model weights [50, 51].

RAG retrieves external evidence for the current question. Fine-tuning changes the model weights so that a behavior becomes part of the model. Skills occupy a third position: they provide structured, human-curated instructions that the agent can load when needed. Through metadata-driven activation and progressive disclosure, the agent loads only a skill's metadata at startup and retrieves full instructions on demand. This keeps token consumption proportional to task complexity. SkillsBench, a benchmark of agent skills across 86 tasks and 11 domains, found that curated skills raised pass rates by 16.2 percentage points on average, though with substantial domain variation (+4.5 pp for software engineering, +51.9 pp for healthcare). Focused skill sets (2–3 modules) outperformed full documentation by nearly $4\times$, and models could not reliably self-generate effective skills [52]. Whether curated skills yield comparable gains for industrial hybrid retrieval over software artifacts remains an open empirical question, as existing benchmarks do not cover this domain.

Polyglot persistence as an architectural pattern. Hybrid retrieval implies heterogeneous access patterns with distinct performance and storage requirements. *Polyglot persistence* uses multiple specialized data stores within a single system rather than forcing all workloads into one database [31]. LLMs can act as natural-language interfaces over heterogeneous backends, translating intent into structured queries and aggregating results across stores [32]. The relevance for this thesis is the interaction between orchestration flexibility and reliability: as queries route across multiple stores, the design must keep access bounded and outputs evidence-grounded.

Evaluation principles and metrics. The research questions require evaluation along three dimensions: retrieval effectiveness, operational efficiency (latency/cost/tool calls), and trust-oriented properties (grounding, failure behavior). Because scope analysis produces ranked recommendations, rank-aware IR metrics apply [9, 10]. NDCG rewards relevant results more when they appear early in the ranked list. Recall@k measures how many known relevant items appear in the top k results, while Precision@k measures how much of the top k is relevant. MRR measures how early the first correct result appears [53]. Beyond retrieval relevance, reliability-oriented evaluation of agentic systems requires:

- **Evidence coverage:** whether each recommendation is supported by retrieved artifacts or explicit relation paths.
- **Faithfulness/consistency:** whether claims are entailed by the retrieved evidence rather than invented.
- **Tool-trajectory correctness:** whether tool calls are valid, bounded, and aligned with the intended workflow.
- **Abstention quality:** whether the system declines or requests clarification when evidence is insufficient, instead of producing confident but unsupported outputs. Selective prediction [54] and unanswerable question detection [55] formalize this as a binary decision problem; recent work extends these concepts to RAG evaluation settings [56].

Scenario-based benchmarks suit agentic workflows because they evaluate both ranking outcomes and intermediate decision behavior across multi-step retrieval and evidence assembly.

LLM-as-a-Judge evaluation. Traditional IR metrics operate on ranked identifier lists and cannot assess response completeness, coherence, or evidence grounding. LLM-as-a-Judge uses a separate language model to evaluate system outputs against reference answers and scoring rubrics, meaning written criteria for how to assign scores [57]. This introduces evaluator-specific risks: verbosity bias, position bias, self-enhancement bias, and hallucinated rationales [58]. Mitigations include model separation (different model family for judging), structured output constraints to prevent scoring drift [59], and calibration protocols that re-judge a sample under identical conditions to quantify intra-judge consistency. In IR evaluation specifically, LLM-based relevance assessment is an established research line: Faggioli et al. chart its prospects and risks [60], Thomas et al. report production-scale agreement with searcher preferences [61], and UMBRELA provides an open-source reproduction of a production relevance assessor [62]. Automated judge frameworks for RAG pipelines include ARES [63] and RAGAS [64].

Model characteristics as fixed selection criteria. Modern LLMs acquire tool-calling behavior through post-training that combines supervised fine-tuning on demonstration data with reinforcement learning. A key finding from recent work is that tool-calling termination behavior is primarily a *trained policy* rather than a capability that emerges from general model scale: without explicit training on when to stop calling tools, models default to over-calling [65, 66].

Post-training data composition and reward design therefore determine a model's *policy prior* (whether it tends toward exhaustive exploration or efficient termination on first sufficient evidence) [67, 68, 69, 70].

This has a direct consequence for model selection: tool-use reliability is a distinct behavioral axis that standard accuracy benchmarks do not capture. Evaluations such as the Berkeley Function Calling Leaderboard (BFCL) and τ-bench consistently show that models with strong general performance can exhibit systematic over-calling, and that the two dimensions are independently variable [71, 72, 66]. Beyond policy, architecture also matters: Mixture-of-Experts (MoE) architectures activate only a subset of parameters per token, whereas dense architectures engage all parameters for every prediction. Whether this design affects behavioral consistency in multi-step tool-calling workflows remains an open empirical question, treated in this thesis as a selection-relevant variable under RQ3.

Post-training divergence among matched-parameter models. When MoE models share comparable active-parameter counts, their developers' technical reports attribute behavioral differences in agentic settings primarily to post-training design rather than to architecture or raw scale [73, 74, 75]. These are vendor reports rather than controlled cross-model studies, so this thesis treats them as hypothesis-generating background whose behavioral predictions are examined empirically under RQ3. Three dimensions of post-training design are particularly consequential for tool-use behavior.

First, *reinforcement-learning pipeline structure* matters. DeepSeek-V3.2 reports that a single merged RL stage jointly optimizing reasoning, tool use, and alignment avoids catastrophic forgetting across capabilities [73]. Such a stage may also carry length biases associated with group-relative policy optimization (GRPO) [76], whose reward normalization has been shown to artificially increase response length [77]. Sequential multi-stage RL, such as separate reasoning, agentic, and alignment phases, preserves per-stage specialist performance. The tradeoff is that it can create distribution seams when the model must autonomously select among learned behaviors at inference time.

Second, *tool-call serialization format* matters. Tool invocations may be encoded as structured JSON with dedicated tokenizer-level special tokens, template-based XML, or custom domain-specific grammars. These choices affect both schema adherence and compatibility with third-party inference frameworks. Custom formats that deviate from widely supported conventions introduce parsing fragility independent of the model's underlying capability.

Third, *runtime reasoning mode* matters. Models that preserve chain-of-thought traces across consecutive tool calls maintain a continuous reasoning ledger that supports iterative refinement. Models that operate in a non-thinking mode, suppressing internal reasoning tokens, must resolve multi-step tool orchestration through shallower pattern matching. These three dimensions interact with each other and with the task environment, making their effects difficult to predict from model specifications alone and motivating empirical comparison under controlled conditions.

3 Method

The study comprised three activities: system design and implementation, data pipeline construction, and benchmark-based evaluation, each mapped to the research questions in Chapter 1 and grounded in Chapter 2.

Anonymization note. To protect proprietary context, internal source-system names and source-specific identifiers are abstracted in this chapter. The enterprise artifact source is denoted as *Source A*; an independent reference-data stream is denoted as the *Procedure Catalog*. Schema prefixes in pseudonymized listings use neutral names (`core`, `raw`, `canon`, `enrichment`, `proc`, `rag`). Pseudonymization changes labels only and does not alter structural or evaluation properties. Technical architecture, retrieval interfaces, and evaluation logic are reported as implemented.

3.1 Methodological Stance, Design, and Pre-study

The study proceeded in four phases: (1) pre-study and benchmark design (problem operationalization, artifact scoping, scenario design); (2) system and pipeline design (polyglot retrieval architecture, ingestion versioning, guarded tool interfaces); (3) implementation of the end-to-end data pipeline, agent runtime, and reliability controls; and (4) evaluation through scenario-based execution, judge-based assessment, deterministic IR scoring, and result aggregation. Separating design from implementation enabled independent evaluation of each.

Problem operationalization. Test scope analysis was operationalized as a retrieval-and-evidence task. In this context, *operationalized* means that the practical engineering activity was converted into measurable benchmark behavior: given a change-oriented query, recommend relevant existing test artifacts and provide inspectable evidence for each recommendation [5, 8].

Each evaluation query therefore has two required outputs: a ranked set of candidate artifacts, and an evidence trail traceable to retrieved records, links, and tool outputs.

Artifact scope and representation policy. The implemented scope included artifacts from one proprietary enterprise system (Source A) and a Procedure Catalog. The Procedure Catalog

contained protocol specifications, message definitions, and use-case mappings, and supported protocol-oriented traceability checks.

Artifacts were organized in a layered storage architecture, following the layered ingestion pattern described by [25]. At the base, a *raw layer* (append-only) preserved ingested source payloads for replay, versioned by ingest run. Above it, a *canonical layer* stored normalized relational records with stable identifiers and typed fields, one row per entity without ingest-run partitioning.

An optional *enrichment layer* consolidated structured metadata extracted via two independent paths: deterministic regex extraction and LLM-based extraction. Where the two paths produced conflicting values, the deterministic result took precedence and conflicts were logged for downstream inspection (see Section 3.2.1). A *RAG layer* held chunked retrieval units and vector embeddings derived from canonical and enrichment data. Finally, a *graph layer* provided a derived graph projection enabling relationship traversal within configurable depth bounds.

Derived outputs (enrichment, RAG chunks, graph projection) were treated as evidence artifacts, not as primary sources of truth.

Evaluation scenario design. The benchmark was implemented as a generated scenario set stratified by retrieval family, complexity, and answerability (Section 3.3). Stratified here means that scenarios were deliberately distributed across these categories instead of sampled as one undifferentiated pool. Each scenario specified explicit expected behavior: expected artifact identifiers, expected entity references, and a detailed reference answer. Binary relevance labels were derived from expected identifiers to support IR scoring.

3.2 Retrieval Harness for Heterogeneous Software Artifacts

Per the scope premises in Section 1.3, hybrid retrieval and the agentic architecture are fixed; this section details the harness contract held constant across conditions. Following the harness-engineering framing of Hashimoto [35] and Seong et al. [36], AGENT = MODEL + HARNESS. The harness is the code and configuration around the language model that determines what the agent perceives, how it acts, and how its work is orchestrated and verified. The harness used here is built on the *DeepAgents* framework [37, 46, 49], which provides the agent loop, middleware stack, tool registry abstraction, subagent and handoff patterns, and a pluggable filesystem `BackendProtocol`. DeepAgents is not a contribution of this thesis.

This thesis contributes the retrieval-domain extensions that adapt DeepAgents to hybrid retrieval over heterogeneous industrial software-engineering artifacts. The extensions span four layers, detailed in the subsections that follow: (i) a polyglot retrieval substrate beneath the harness, integrating relational, dense-vector, and graph storage (Section 3.2.1); (ii) layered guardrails enforcing read-only, bounded, and audited access at the application, session, and database layers (Section 3.2.1 onward); (iii) a custom PostgreSQL-backed implementation of the DeepAgents `BackendProtocol`, scoped per session, with universal tool-result offloading (Section 3.2.1); and (iv) a three-tier evaluation pipeline with calibration and meta-judge arbitration (Section 3.3).

The contributed extensions are workflow-agnostic with respect to the orchestration strategies supported by DeepAgents. The tool registry, guardrails, virtual filesystem, and reliability configuration are held constant. ReAct [29] as a single-agent reasoning–acting loop, and Plan-and-Execute over an orchestrator-worker topology [34, 37], both adopted unchanged, serve as the two experimental conditions under RQ2. These workflows were selected because they represent the two control patterns available in the adopted DeepAgents foundation and correspond to the study's main workflow contrast: reactive single-agent control versus explicit task decomposition. Because every harness-side component is identical across the two workflows, observed performance differences are attributable to the workflow's reasoning strategy under the tested runtime constraints.

Table 3.1: Harness-enforced invariants and the workflow behaviors they exclude.

Harness invariant	**Workflow cannot**
Fixed tool registry composition	call tools outside the registry
SQL / Cypher guardrails	issue unvalidated queries
Tool output bounds and timeouts	let a single tool call run unbounded
Universal offload to virtual filesystem	push raw tool output into context
Per-call checkpoint logging	evade audit
Read-only data access	modify the corpus

Table 3.2: Workflow-level policies that vary across the two conditions under RQ2.

Policy	**ReAct**	**Plan-and-Execute**
Tool-call ordering	serial, interleaved	dispatch to sub-agents
When reasoning occurs	interleaved with action	plan first, then execute
Task decomposition	implicit	explicit plan
Failure recovery	self-loop	replan
Context topology	single shared context	isolated per sub-agent
Termination	model-decided	model-decided

Table 3.3: Contribution boundary: harness layers contributed by this thesis versus components adopted from the DeepAgents foundation and prior work.

Component	**This thesis**	**Adopted from**
DeepAgents harness foundation (agent loop, middleware, tool registry, `BackendProtocol`)	no	LangChain [37, 46]
Polyglot retrieval substrate (Postgres + pgvector + Neo4j)	design	pattern [31, 32]
Three-layer SQL guardrails (AST + role + DB)	yes	original
Cypher guardrails (allowlist + hop bound + probe)	yes	original
AgentFS schema (3 tables, dual-key scoping) implementing DeepAgents `BackendProtocol`	yes (impl.)	interface [46]
Universal tool-result offloading	yes (extension)	default [49]
Three knowledge-augmentation modes	yes	original
Tool registry composition (14+ tools, schemas)	yes	original
Three-tier evaluation pipeline (judge + calibration + meta)	yes	original
52-scenario stratified industrial benchmark	yes	original
ReAct workflow	no	[29]
Plan-and-Execute / orchestrator-worker workflow	no	[34, 37]

Tables 3.1 and 3.2 summarize the separation of concerns; Figure 3.1 shows the runtime behavior of the two workflows side-by-side. Table 3.3 delineates which harness components are this thesis's contribution and which are adopted from prior work or from the DeepAgents foundation.

The remainder of this chapter details each harness layer.

3.2.1 Polyglot retrieval substrate

RQ1 and RQ2 require retrieval from heterogeneous knowledge types (structured records, embedded text, explicit artifact relationships) under controlled, auditable conditions. The underlying corpus comprised 11,366 test cases, each carrying structured metadata (name, description, quality area, execution context, automated suite assignment) and linked to 1,365 backlog entities (requirements, design documents, specification items) through 15,222 reference

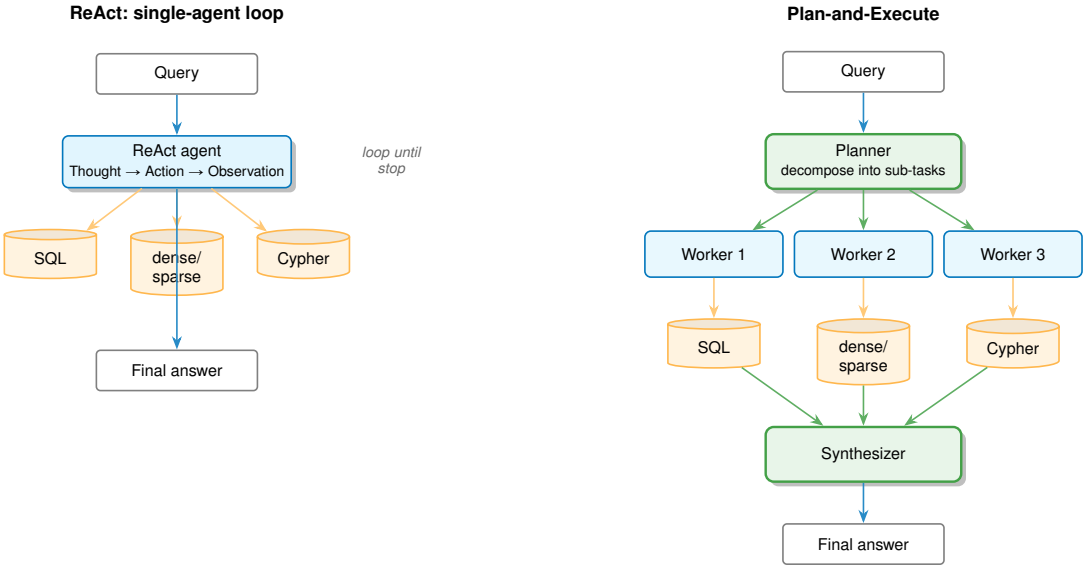


Figure 3.1: Runtime flow of the two agentic workflows compared under RQ2. **Left:** ReAct executes a single agent in an interleaved Thought–Action–Observation loop over a shared context window. **Right:** Plan-and-Execute decomposes the query in a planner, dispatches sub-tasks to isolated workers (each with its own sub-context), and synthesizes results; replanning is triggered when synthesis detects gaps. Both workflows operate over an identical harness contract (tool registry, SQL/Cypher guardrails, AgentFS, reliability configuration), so any performance difference is attributable to the workflow's reasoning strategy rather than to the surrounding infrastructure.

links. Concretely: typed test-case records and backlog items, free-text requirement descriptions and test instructions, and trace links with resolution outcomes. No single backend supports relational queries, approximate nearest-neighbor search, and bounded graph traversal at the required granularity [31]. Two complementary backends were adopted, providing three retrieval modalities. PostgreSQL (extended with pgvector [78]) served as the canonical data store and supported two of these modalities: relational query access over structured records, and approximate nearest-neighbor search via dense retrieval over embedded requirement and test-instruction text. Neo4j provided the third modality, relationship-centric traversal over a synchronized graph projection derived from the canonical layer. Relational canonical views remained the authoritative data layer; graph data was synchronized as a derived query layer for structural navigation. This separation kept the provenance chain unambiguous: every graph node or edge could be traced back to a canonical record at a specific ingest run.

Run persistence. Beyond retrieval data, the system persisted runtime state to support post-hoc auditability, a prerequisite for the reproducibility controls in Section 3.4 and the tool-trace analysis in Section 3.3. Two persistence layers served this purpose. Conversation checkpoints were written to a dedicated checkpoint schema (`langgraph_cp`) using the LangGraph PostgresSaver [79], enabling replay and inspection of reasoning trajectories across sessions; tool-call sequences, arguments, return values, and token counts were recoverable from the checkpoint message history. In addition, an append-only virtual filesystem (`agentfs` schema; Section 3.2.1), scoped by assistant and thread identifiers, recorded file artifacts produced during agent runs. This separation was necessary because the evaluation pipeline's grounding checks (Section 3.3) relied on structured tool-call records extracted from checkpoints rather than on free-text output parsing.

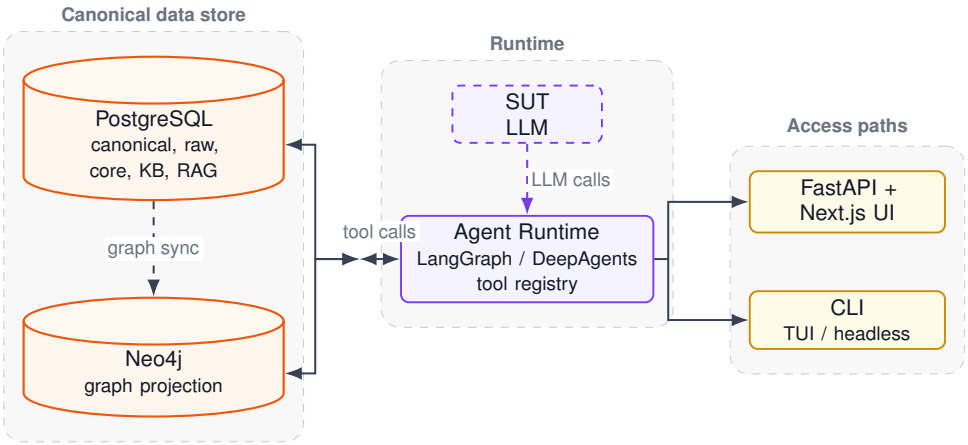


Figure 3.2: System architecture overview. PostgreSQL, extended with pgvector, serves as the canonical data store; Neo4j provides a derived graph projection. The agent runtime exposes guarded tool interfaces to both the web application and the CLI.

Agent virtual filesystem. The agent operated over a PostgreSQL-backed virtual filesystem (`agentfs` schema) implementing the DeepAgents `BackendProtocol` [46]. This protocol is the framework's pluggable storage contract that any custom file backend must satisfy to be exposed through the agent's file-oriented tools. The schema (Appendix B) comprises three tables: file metadata, append-only content revisions, and binary blobs. These tables are jointly scoped by (`assistant_id`, `thread_id`) so that each conversation session held an isolated, persistent workspace.

Within this workspace, the agent could write, read, list, and search file artifacts through the backend's file-oriented tool interface. Artifact and export tools (CSV, JSON, XLSX tabular export; markdown response writer; source-link collector for citation assembly) operated over this virtual filesystem. This enabled the agent to accumulate structured outputs across multi-step reasoning rather than emitting results only as transient tool return values. The same workspace was accessible to the end-user through both the web application and the CLI, providing a shared surface for collaborative artifact inspection. A non-sandbox filesystem backend was selected because the artifacts written to it (markdown summaries, tabular exports, source-link lists) are pure data and are never executed, making the isolation cost of a sandbox-backed volume unnecessary [47].

The virtual filesystem additionally served as the target for systematic result offloading. The DeepAgents framework supports offloading tool results that exceed a token threshold to the filesystem, substituting a compact reference and preview in the context window [49]. This implementation extended that pattern to offload *every* retrieval tool result from PostgreSQL and Neo4j, regardless of size. Each tool call wrote its full result set to a named file in the virtual filesystem and returned only a truncated preview together with the file path. The agent then used the filesystem's read and search operations to access the offloaded results selectively, retrieving only the portions relevant to the current reasoning step. This design prevented large query results (vector search candidate lists, multi-row SQL outputs, graph traversal neighborhoods) from consuming the context window, and converted the agent's interaction with retrieval evidence from a push model (entire result injected into context) to a pull model (agent reads what it needs, when it needs it).

The virtual filesystem also served as a fallback extraction path: when checkpoint data was unavailable, the evaluation pipeline recovered agent artifacts from the `agentfs` schema (Section 3.3).

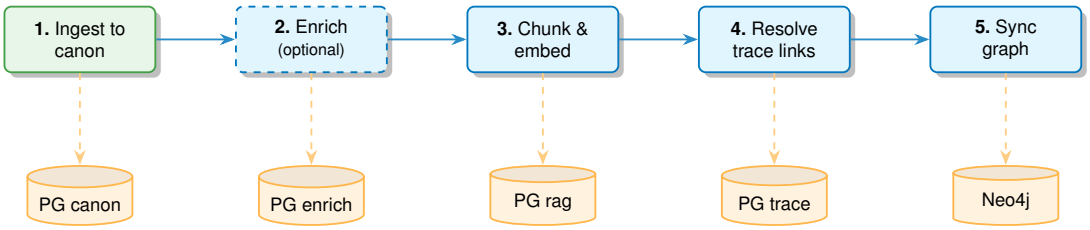


Figure 3.3: Data pipeline stages, executed sequentially. Solid arrows denote execution order: stage $n + 1$ consumes the output of stage n. Dashed arrows denote persistence side effects (each stage writes its results to the indicated store). A post-ingest orchestrator chains stages 3–5 automatically. The enrichment stage (dashed border) is optional; the pipeline operates correctly without LLM-based metadata.

Data pipeline. The data pipeline was orchestrated through a unified CLI that executed five stages sequentially.

1. **Ingest to canonical schema:** Ingest source records, detect changes against the previous ingest run, and transform raw records into canonical tables with stable identifiers, typed fields, and deterministic deduplication. Test instructions were split into typed sections by a deterministic SQL function.

2. **Enrich:** Extract structured metadata through two independent paths. A deterministic path used regular expressions to extract domain-specific attributes directly from canonical text fields. An LLM-based path produced summaries, tags, and component classifications. Both results were stored in the enrichment schema and consolidated into a unified metadata view where deterministic extractions took precedence on overlapping fields.

3. **Build chunks and embeddings:** Generate chunk representations from canonical and enrichment records and compute 2000-dimensional dense vectors using the project embedding model (Table 3.6). Chunk deduplication was enforced by a SHA-256 content hash computed at the database level, ensuring idempotent re-ingestion, meaning that re-running ingestion on unchanged content produced the same stored chunk identity rather than duplicates.

4. **Resolve trace links:** Extract requirement references embedded in test-case records, normalize identifiers, infer reference kinds, and populate a trace-link table for downstream graph construction.

5. **Sync graph projection:** Populate graph nodes and edges from canonical and enrichment tables.

Procedure Catalog. A second data stream, the Procedure Catalog, was ingested independently of the primary source. This stream contained protocol specifications (interface and protocol names), message definitions with alias variants, and coverage status records. The Procedure Catalog was stored in a dedicated schema (`proc`) and exposed through combined views joining protocols, messages, and aliases. It enabled protocol-oriented coverage checks (e.g., matching test instructions to standardized message names) alongside the primary artifact retrieval.

Trace-link resolution. Trace-link resolution extracted requirement references from test-case payloads, normalized identifiers (stripping trailing qualifiers and bracketed annotations), and

inferred the reference kind from the identifier structure. The resulting trace-link table recorded (test-case identifier, reference identifier, inferred kind) triples and served as the input for the graph edge derivation step.

Graph projection rebuild. The graph synchronization step operated in *full wipe-and-reload mode*: all existing graph nodes and edges were deleted before repopulating from canonical and enrichment tables. Ten node types were materialized: test cases, six backlog-item subtypes sharing a common base label, and two procedure types (protocols and messages). Five relationship types connected them: hierarchical containment, test-case–to–backlog references, test-case–to–message coverage, derived test-case–to–protocol coverage, and message–to–protocol membership. Uniqueness constraints on each node type's business key ensured idempotent loading. Full wipe-and-reload was chosen to eliminate stale edges across independently updated source streams. Incremental graph synchronization was not implemented because freshness and reproducibility were more important for the benchmark than update latency, and the graph size was small enough for full rebuilds to remain practical.

Retrieval interfaces and guardrails. The agent accessed guarded retrieval interfaces spanning complementary access modes. *Identifier and trace lookup* provided direct retrieval by primary key or trace-link query. Text-oriented retrieval had three variants: *sparse keyword retrieval* (BM25 over indexed text fields [39, 80]), *dense vector retrieval* (cosine-similarity over 2000-dimensional HNSW-indexed chunk embeddings [81]), and *hybrid retrieval* combining both via RRF [41]. Structural navigation included *bounded graph access* (node lookup, neighborhood expansion, impact analysis, read-only Cypher) and *constrained SQL access* (read-only queries restricted to current-snapshot views).

Schema introspection tools provided schema overview and relation descriptions. Four *knowledge-base grounding* tools validated domain terminology against a curated KB: direct entry lookup, keyword and vector search over KB fragments, single-term resolution with fuzzy matching, and query-term expansion. These tools reduced hallucination of acronyms, abbreviations, and testing-strategy references. *Artifact and export tools* operated over a per-session virtual filesystem: artifact overview and JSON slicing, glob and grep across accumulated artifacts, a markdown response writer, tabular export (CSV, JSON, XLSX), and a source-link collector for citation assembly.

Read-only SQL enforcement. SQL queries were constrained at three complementary layers, forming a defense-in-depth arrangement.

At the *application layer*, every query was parsed by an abstract-syntax-tree validator that rejected any statement other than a pure `SELECT` (with `WITH` clauses permitted) and any query containing an embedded semicolon. A deny-list additionally blocked calls to filesystem and network primitives (`pg_read_file`, `pg_ls_dir`, `pg_read_binary_file`, `lo_export`, `dblink`, `dblink_connect`).

At the *session layer*, the connection issued `SET ROLE agent_readonly` before every guarded query, downgrading privileges to those of a role holding only `SELECT` on the six allowlisted schemas (`canon`, `enrichment`, `proc`, `rag`, `core`, `kb`). The search path was restricted to those schemas so that unqualified names could not resolve to other objects.

At the *database layer*, the `agent_readonly` role was defined as `NOLOGIN` with no default privileges beyond those explicitly granted.

No single misconfiguration can permit a write or out-of-scope read, since all three layers must be bypassed simultaneously.

Numeric bounds applied uniformly to all SQL retrievals are summarized in Table 3.4.

Read-only Cypher enforcement. Cypher queries were validated by a multi-stage guard. First, queries were required to begin with one of the read-only clause initiators (`MATCH`, `OPTIONAL`

Table 3.4: SQL retrieval bounds (from `config.toml`).

Parameter	Value	Effect
Statement timeout	4,000 ms	Hard wall-clock limit per query
Row cap	2,000	Post-fetch truncation
Default result limit	50	Applied when agent omits `LIMIT`
Maximum result limit	2,000	Ceiling for agent-specified `LIMIT`
Text truncation (default)	8,000 chars	Default output truncation
Text truncation (detail)	20,000 chars	Extended mode truncation

`MATCH`, `WITH`, `UNWIND`, `RETURN`) and to include an explicit `RETURN` clause. A deny-list of 16 mutation and administrative keywords (`CREATE`, `MERGE`, `SET`, `DELETE`, `DETACH`, `REMOVE`, `CALL`, `DROP`, `LOAD`, `FOREACH`, `DBMS`, `TRANSACTION`, `INDEX`, `CONSTRAINT`, `GRANT`, `REVOKE`) blocked write and schema-altering operations. Procedure namespaces with write or administrative capability (`apoc.*`, `db.*`, `gds.*`) were denied by wildcard pattern, where `*` means any procedure name below that namespace. A self-check probe executed at startup verified that the guard correctly blocked a representative `CALL` invocation. Queries were additionally constrained to a node-label allowlist (10 labels) and a relationship-type allowlist (5 types), both loaded from the configuration file. Unbounded variable-length traversal (`*` without an upper bound) was rejected at parse time. Bounded patterns of the form `*n..m` were accepted only if $m \leqslant 4$; otherwise the query was rejected before execution.

Numeric bounds for Cypher retrievals are summarized in Table 3.5.

Table 3.5: Cypher retrieval bounds (from `config.toml`).

Parameter	Value	Effect
Query timeout	4.0 s	Hard wall-clock limit per query
Row cap	200	Post-fetch truncation
Default result limit	50	Applied when agent omits `LIMIT`
Maximum result limit	200	Ceiling for agent-specified `LIMIT`
Maximum hop count	4	Upper bound on variable-length traversal
Default traversal depth	1	Default for structured traversal tool

Typed input validation. All tool inputs were validated against typed schemas before query construction. Response shapes were checked post-execution; tool errors were surfaced to the agent as structured error objects and recorded in the conversation checkpoint. Every scenario therefore yields a verdict: there are no failure paths in which an erroneous tool call is silently discarded.

Agent runtime and reliability controls. The agent was built on the LangChain DeepAgents framework [37], a Python library that implements agents as middleware-extensible ReAct-style reasoning loops [29] with built-in support for guardrails, tool interception, and checkpoint persistence. Two workflow families were supported, distinguished by execution topology and cognitive control pattern (Section 2.4). A *ReAct* workflow operated as a single DeepAgent with the full retrieval tool registry, alternating between reasoning and tool calls within one iterative cycle. A *Plan-and-Execute* workflow implemented a plan-and-execute cognitive pattern [45] atop an orchestrator-worker topology [34]. An orchestrator DeepAgent, equipped only with a delegation tool and no retrieval tools, decomposed the query into sub-tasks. Each sub-task was delegated to an executor sub-agent (itself a ReAct-based DeepAgent with the full retrieval tool registry) running in its own isolated context window, with optional replanning between steps. Both workflows shared the same retrieval tool registry and guardrail configuration at the executor level; the choice of workflow was an experimental variable for RQ2 (Section 3.3).

The primary access path was a web service that exposed thread, run-stream, and file endpoints consumed by a browser-based application, designed as the end-user interface for interactive test scope analysis; no formal user study of this interface was conducted (Section 5.4). A secondary CLI provided a headless invocation mode, meaning it ran the same agent without the browser UI. This mode was used exclusively by the evaluation harness (Section 3.3) to execute benchmark scenarios under deterministic, scriptable conditions. Both paths shared the same runtime layer for tool construction, prompt assembly, and persistence: conversation checkpoints and virtual-filesystem state were stored in PostgreSQL (schemas `langgraph_cp` and `agentfs`, respectively).

Context engineering. Three context-management strategies (Section 2.4) were applied to sustain multi-step reasoning within context-window limits [49]. *Offloading* was applied universally to all PostgreSQL and Neo4j retrieval tool results as described in Section 3.2.1: every result was written to the virtual filesystem with only a truncated preview and file reference retained in context. *Summarization* was handled by a compaction middleware that monitored context usage and, when the accumulated token count approached the model's window limit, replaced older conversation turns with an LLM-generated summary preserving task intent, produced artifacts, and next steps; the original messages were preserved in the virtual filesystem for on-demand recovery. *Context isolation* was inherent in the Plan-and-Execute workflow: each executor sub-agent ran in its own context window, and the orchestrator received only the sub-agent's final result, preventing intermediate retrieval outputs from accumulating in the orchestrator's working memory.

Execution controls. Execution controls constrained the agent's behavior at runtime. A recursion limit of 1,000 tool steps prevented non-terminating reasoning loops while remaining far above the expected path length of successful runs, so it functioned as a safety ceiling rather than as an ordinary stopping rule. The plan-execute planner was permitted at most 6 iterations per scenario, enough to cover retrieval, traceability, verification, and synthesis steps while bounding repeated replanning. Strict input-schema validation rejected malformed tool calls before execution. Retrieval result sizes were bounded per tool (Tables 3.4, 3.5, and 3.6). When the agent's response did not satisfy evidence contracts (for example, if claims lacked grounding in retrieved artifacts), a validation step triggered retry or fallback behavior rather than emitting an unsupported answer.

Retrieval configuration. To support RQ2's comparison of workflow configurations under controlled conditions, all retrieval and agent parameters were held constant across reported runs. Table 3.6 reports these values, drawn from a single version-controlled configuration file (`config.toml`). Fixing these parameters ensures that observed differences between runs are attributable to the independent variables (workflow family, knowledge configuration, and model) rather than to incidental parameter variation.

Table 3.6: Retrieval and embedding parameters used in reported runs.

Parameter	Value
Embedding	
Embedding model	Qwen/Qwen3-Embedding-8B
Embedding dimension	2 000
Chunk size (% of model context)	90%
Chunk overlap (% of chunk size)	10%
Vector index type	HNSW (cosine)
Sparse / keyword retrieval	
Default result limit	50
Maximum result cap	400
Dense / vector retrieval	
Default result limit	50
Maximum result cap	300
Hybrid retrieval (RRF)	
RRF fusion constant K	60
Maximum result cap	300
Graph traversal	
Default neighborhood depth	1 hop
Maximum traversal depth	4 hops
Agent runtime	
Recursion limit (tool steps)	1,000
Plan-execute planner iterations	6
LLM temperature	0.0
LLM provider request timeout	2 000 s
Benchmark per-turn cutoff	1 200 s

Chunks were encoded with `Qwen/Qwen3-Embedding-8B` [82] (2000-dimensional vectors). At query time, the provided query string was embedded with the same model and compared against stored embeddings using cosine distance via an HNSW index. The dense-retrieval default result limit was top-50, with a maximum of 300. This default gave the agent enough candidates for recall-oriented scope analysis without flooding the context or artifact store. Keyword search used BM25 ranking via the `pg_search` extension [80, 39], preserving operator semantics (`AND`, `OR`, phrase queries). Its default result limit was also top-50, with a maximum of 400 to support broader exact-term searches over identifiers and names. Hybrid retrieval gathered dense candidates (pool size 100) and sparse candidates (pool size 100) independently, then fused the ranked lists using Reciprocal Rank Fusion [41] with constant $K = 60$:

$$\text{score}(d) = \frac{1}{K + r_{\text{dense}}(d) + 1} + \frac{1}{K + r_{\text{sparse}}(d) + 1} \tag{3.1}$$

where $r_{\text{dense}}(d)$ and $r_{\text{sparse}}(d)$ are the zero-based ranks of document d in the dense and sparse lists, respectively (absent documents receive no contribution). The fused list was sorted descending by score and truncated to the requested limit (default 50; maximum 300). The value $K = 60$ is the standard setting from the original RRF paper and provides a smooth rank-based discount without the need for score calibration between the two retrieval lanes [41].

3.3 Evaluation Protocol

Evaluation objectives and RQ mapping. Evaluation targeted three dimensions mapped to the research questions. RQ1: whether LLM-as-a-Judge correlates with deterministic IR metrics [57, 10]. RQ2: how knowledge-augmentation mode and agentic workflow affect task correctness, retrieval effectiveness, and evidence grounding, in line with agentic tool-use benchmarking practice [71, 72]. RQ3: how language-model characteristics influence reliable tool use and abstention behavior across configurations.

Benchmark pipeline overview. The evaluation was implemented as a modular eight-stage CLI pipeline:

1. **Generate:** create stratified scenario fixtures with ground truth.
2. **Execute:** run the agent (SUT) against all scenarios.
3. **Extract:** parse raw output into structured records with tool traces.
4. **Judge:** score each execution via an LLM-as-a-Judge evaluator [57].
5. **Calibrate:** re-judge a stratified sample under identical settings to measure intra-judge consistency.
6. **Meta-judge:** resolve disagreements between primary and calibration verdicts to assess judge reliability.
7. **Score:** compute deterministic IR and functional metrics.
8. **Plots:** produce publication-quality figures and LaTeX tables organized by research question.

Each stage reads from the previous stage's output and writes to its own directory, enforcing a directed acyclic graph (DAG) structure. In this context, DAG means that stages flow forward without circular dependencies: later stages depend on earlier artifacts, but not the reverse. Individual stages can be re-run independently (e.g., re-scoring after a metric change without re-executing the agent). A single configuration file parameterizes benchmark parameters, agent settings, judge and meta-judge models, retry defaults, and plotting options.

Knowledge configurations. Knowledge configuration was varied across three levels that shared identical domain knowledge and tool registries but differed in delivery mechanism (Section 2.4). A *Prompt* configuration embedded domain policies (retrieval hierarchy, domain model, and response format rules) directly in the system prompt, serving as the baseline without agent skills. A *Skill* configuration delivered the same policies through three curated agent skill modules (domain knowledge, tool routing, and response format) loaded into the agent's context via metadata-driven activation rather than baked into the prompt [50]. A *Skill+Meta* configuration extended the Skill setup with a preferences module containing explicit load-order guidance and usage heuristics for applying the domain policies. This tested whether guidance about *when* to activate each skill provided benefit beyond the skill content itself. All three configurations shared the same tool registry and guardrail settings (Section 3.2.1); only the knowledge-delivery mechanism differed.

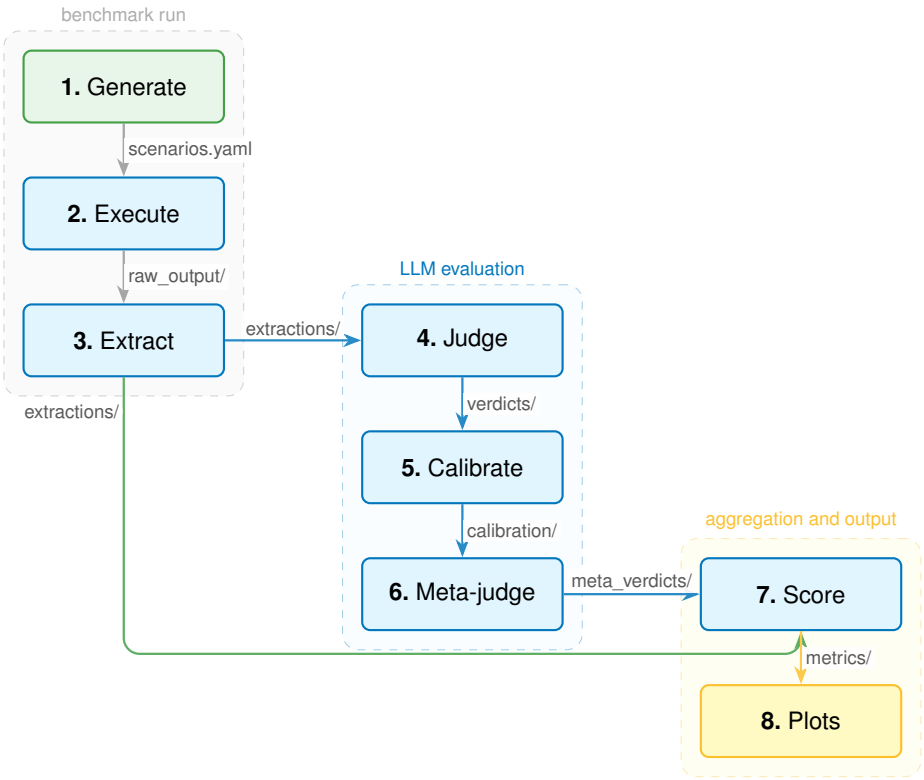


Figure 3.4: Benchmark evaluation pipeline. Eight stages process scenarios through agent execution, LLM-based judging with calibration and meta-judge validation, deterministic scoring, and publication-quality export. Arrows are annotated with intermediate artifact types.

Workflow families. Workflow family was varied across two levels. A ReAct loop [29] operated as a single agent with the full retrieval tool registry, alternating between reasoning and tool calls within one iterative cycle. A Plan-and-Execute workflow [45] implemented plan-then-execute control atop an orchestrator-worker topology [34]. An orchestrator agent equipped only with a delegation tool decomposed the query into sub-tasks. ReAct-based executor sub-agents, each with an isolated context window and the full retrieval tool registry, handled those sub-tasks; replanning between steps was optional. These workflows were selected because they were supported by the adopted harness and expose the study's intended control-flow contrast.

Model selection and execution matrix. Three frontier-class SUT models were selected to provide an all-MoE, roughly matched active-parameter sample with distinct tool-calling and post-training regimes: DeepSeek-V3.2, Qwen3-Coder, and GLM-5.1. The selection reflects both research design and practical availability through the industrial evaluation environment. Holding architecture family approximately constant while varying post-training and tool-use behavior supports RQ3, although it does not allow causal isolation of any single model property. Table 3.7 summarizes the selected models. The full execution matrix comprised 3 models $\times$ 2 workflows $\times$ 3 knowledge configurations, yielding 18 configurations. Each configuration was applied to all 52 scenarios, for a total of 936 agent executions.

Scenario generation. Scenarios were generated through a multi-agent orchestration pipeline with human validation. Generation agents queried the canonical database to derive ground-truth identifiers, entity references, and detailed reference answers directly from production

Table 3.7: Evaluated LLM back-ends. All are mixture-of-experts architectures; active parameter counts reflect per-token routing.

Model	Developer	Params (total / active)	Experts (total / active)	Context (served)	Release	License
DeepSeek-V3.2	DeepSeek-AI	671 B / 37 B	257 / 9	128 K	Jan 2026	MIT
GLM-5.1	Z.ai (Zhipu)	744 B / 40 B	256 / 8	197 K	Apr 2026	MIT
Qwen3-Coder	Alibaba	480 B / 35 B	160 / 8	205 K	Jul 2025	Apache-2.0

Model specifications per the developers' technical reports and model cards [73, 74, 75]. *Context (served)* is the context-window limit as deployed on the internal AI provider through which all three models were accessed in this study (Section 5.3); native vendor-reported windows differ for some models (e.g., Qwen3-Coder supports 256 K natively). DeepSeek-V3.2 expert counts include the shared expert (256 routed + 1 shared total; 8 routed + 1 shared active); GLM-5.1 and Qwen3-Coder counts are routed-only.

data. Independent reviewer agents then re-queried the database to validate every expected identifier and entity reference, flagging factual errors and hallucinated ground truth. Approximately one-third of the scenarios were additionally validated manually through the agent CLI, verifying that the system produced responses consistent with the stated ground truth. This subset was chosen as a coverage-oriented manual audit under project timeline and domain-access constraints: it sampled scenario families and complexity levels without turning the study into a full human-annotation project.

Scenarios were stratified along three dimensions to ensure coverage: *retrieval family*, *complexity* (single-hop, multi-hop, and reasoning), and *answerability* (answerable or unanswerable) [83]. Five retrieval families were defined:

- **Lookup** (10 scenarios): resolve a single artifact by identifier and return its metadata.
- **Search** (14 scenarios): find artifacts matching a natural-language description across multiple fields.
- **Traceability** (16 scenarios): follow trace links between artifacts (e.g., from a requirement to the test cases that verify it).
- **Impact** (8 scenarios): given a component or requirement, identify all test cases that would be affected if it changed (change-impact analysis).
- **Comparison** (4 scenarios): contrast two or more artifacts on shared and distinguishing attributes.

Both single-turn and multi-turn scenarios were supported. The final scenario set comprised 52 evaluation scenarios. Each scenario is a distinct retrieval task whose ground truth references specific test cases and entities within the full 11,366-case corpus; individual scenarios span up to 97 test cases, and scenarios are not synonymous with test cases. For example, an impact-analysis scenario asks which test cases would be affected if a specific architectural component changed; the ground truth for this single scenario references 83 test cases and requires the agent to traverse knowledge-graph edges to identify the full blast radius. A 52-scenario topic set is consistent with established IR evaluation practice: TREC ad hoc tracks have long used topic sets of approximately 50 queries, as standardized in the TREC-8 ad hoc evaluation (50 topics) [33], supporting this as a defensible operating point rather than an arbitrary small number. Each scenario in this benchmark also carries more signal than a typical TREC topic, since it embeds multi-hop reasoning, abstention checks, and database-derived ground truth with confusion traps; the operative unit count for the underlying retrieval target is the 11,366-case corpus, not the scenario count itself.

Each scenario included: (i) expected artifact identifiers and expected entity references for deterministic ground-truth verification; (ii) a detailed reference answer against which the LLM judge could assess completeness and factual accuracy; and (iii) an answerability classification indicating whether the query was designed to be answerable given the current knowledge base. Listing 3.1 shows a representative scenario.

Listing 3.1: Redacted impact-analysis scenario (YAML). Ground truth references 12 representative test cases out of 83 total; the agent must traverse graph edges from the target component to identify the full set.

```yaml
- id: impact_component_X
  type: single_turn
  query: >
    If component X is updated, which test cases
    would be affected?
  stratification:
    retrieval_family: impact
    complexity: single_hop
    answerable: true
  ground_truth:
    expected_ids:       # 12 of 83 shown
    - TC-001
    - TC-002
    - TC-003
    # ... (71 additional IDs omitted)
    expected_entities:
    - COMPONENT-X
    reference_answer: >
      Component X is directly referenced by 83
      test cases. A change would require
      re-validating all of them. [...]
```

The full scenario set was persisted as a versioned YAML file at the start of each benchmark run.

Agent execution. The execution stage ran the SUT against every cell in the Cartesian product of scenarios, knowledge configurations, workflow modes, and models. An async bounded-concurrency executor limited parallelism. Here, *async* means executions were scheduled without blocking on one run at a time, while *bounded concurrency* means that the number of simultaneous executions was capped to avoid overloading providers or infrastructure. Each execution invoked the agent programmatically, supplying a single query and collecting the final response without intermediate feedback. Two clocks were configured: the LLM provider client used a 2 000 s request timeout, while the benchmark harness enforced a 1 200 s per-turn cutoff. The generous benchmark cutoff targeted correctness rather than latency: a tight deadline would conflate genuine failures (tool-call errors, reasoning loops, malformed outputs) with timeout artifacts. Executions exceeding the benchmark cutoff were classified as timeouts, indicating non-terminating loops given the generous limit. No retries were performed, preserving the natural failure distribution. Execution-time distributions are reported (Section 4.3) via cumulative distribution functions.

Each execution was classified into one of four outcome categories: success, timeout, agent error, and cascaded failure (propagated from an upstream dependency). Per-execution metadata was written to a structured manifest record containing the scenario identifier, thread and assistant identifiers for post-hoc checkpoint recovery, the knowledge configuration and workflow mode used, the SUT model identifier, and per-turn timing and outcome data.

Extraction and trace recovery. The extraction stage parsed raw execution output into structured records suitable for downstream scoring. The primary recovery path used the LangGraph checkpointer to extract token counts (input and output), the complete tool-call sequence with arguments and return values, and the final agent response text. A fallback path recovered traces from the virtual file-system backend when checkpointer data was unavailable. Each extraction record was written to a per-execution JSON file in the `extractions/` directory.

LLM-as-Judge scoring. Each extraction was evaluated by an LLM-based judge following the LLM-as-a-Judge paradigm [57]. The judge was implemented using a common harness interface with multiple backend implementations. A resilient wrapper chained backends with exponential backoff, falling back first to a secondary provider and then to a tertiary provider, meaning the third provider in the fallback order, on rate-limit or availability failures. Judge outputs were constrained to a JSON schema enforcing the verdict format [59], eliminating the need for post-hoc output parsing.

Model separation was enforced between the SUT and the evaluator: the judge used a frontier-class model from a different provider family than the SUT to reduce evaluator–SUT bias. Context isolation was maintained by instantiating one judge invocation per scenario, preventing cross-scenario leakage of identifiers or prior judgments.

The verdict schema recorded four components. First, it stored a binary judge status (pass/-fail) and an overall score on a continuous $[0, 1]$ scale. Second, it stored a ground-truth sub-verdict checking expected identifiers, expected entities, and reference-answer alignment. Third, it stored a quality assessment across four dimensions (Completeness, Relevance, Coherence, and Evidence Grounding), each scored on a 1–5 Likert scale with a free-text rationale. Fourth, it stored a short natural-language summary. If all retry attempts failed for a given scenario, the pipeline emitted a *blocked* verdict rather than silently dropping the scenario, ensuring that the failure was visible in downstream aggregation. Additional pipeline-level statuses (cascaded_ blocked, cascaded_failure) tracked scenarios affected by upstream failures.

Judge calibration. LLM-based judging introduces evaluator variance [57]. Zheng et al. documented systematic biases when an LLM acts as an external evaluator: verbosity bias, position bias, and self-enhancement bias. Meta-judging surveys [58] add prompt sensitivity, Likert score saturation, bias toward confident-sounding but logically flawed responses, and hallucinated rationales. Because this benchmark uses an external, stronger model as judge (not the SUT judging itself), these biases must be estimated empirically.

LLM judges exhibit non-negligible intra-rater variability, meaning that the same evaluator can assign inconsistent scores to identical inputs across repeated runs [84, 85]. To quantify this, a calibration stage re-judged 156 executions under identical prompt, model, and parameter settings. The sample was stratified across the reported model–workflow–knowledge cells while preserving coverage of both pass and fail verdicts. Primary and calibration verdicts were compared per quality dimension using Cohen's κ [86] at two binarization thresholds ($\geqslant 3$ and $\geqslant 4$ on the 1–5 Likert scale) and Krippendorff's α [87] for ordinal reliability across the full range.

Meta-judge arbitration. Where primary and calibration verdicts disagreed, a meta-judge stage escalated the conflict to a higher-tier model, following cascaded selective evaluation [88]. For each disagreeing (scenario, configuration) pair, the meta-judge received both verdict rationales alongside the original agent output and ground truth, and produced an arbitrated verdict. The meta-judge used Claude Opus 4.6 rather than the primary judge model, Claude Sonnet 4.6, according to the provider's model hierarchy at evaluation time. Bootstrap confidence intervals [89] were computed for the judge-agreement metrics, and Holm–Bonferroni correction [90] was applied within that family of agreement hypotheses. These statistics bound evaluator variance and surface systematic scoring drift (Section 3.4); their scope is the judge-reliability analysis, not the configuration-level comparisons in Chapter 4, which are reported descriptively.

Deterministic scoring and metrics. Five metric families were computed from extraction and verdict data, each targeting a specific evaluation dimension.

IR metrics (RQ1) captured rank-aware retrieval quality using standard formulations [9, 10]. NDCG@5 and NDCG@10 measured relevance in the top ranks using a gain function $G(r) = 2^r - 1$ where $r \in \{0, 1\}$ is a binary relevance indicator derived from the scenario's

expected artifact identifiers (present = relevant, absent = not relevant). MRR@10 captured how quickly the first relevant artifact appeared. Recall@10 measured coverage of known relevant artifacts, and Precision@5 measured the fraction of top-ranked results that were relevant. Label coverage, the proportion of retrieved documents for which relevance judgments were available, was computed during scoring and inspected as a sanity check; it is preserved in the run artifacts but is not reported per configuration in Chapter 4. All IR metrics were computed from binary relevance labels derived from the expected identifiers in each scenario, independently of the LLM judge; retrieved artifacts outside the expected-identifier list are scored as non-relevant, a known limitation of evaluation with incomplete relevance judgments [91].

Functional verification (RQ2) applied deterministic checks to each extraction: whether the exit code indicated success, and whether expected artifact identifiers and entity references appeared in the agent's response. Each criterion was binary (pass/fail) with an explicit explanation.

Grounding analysis (RQ2) assessed hallucination risk by comparing entity identifiers in the agent's response against the evidence pool constructed from tool-call results [26, 64]. Entity extraction used five regular-expression patterns targeting domain-specific identifier formats present in the evaluation data: Source A test-case identifiers, Source A backlog-item identifiers, cross-system requirement references, Procedure Catalog protocol names, and Procedure Catalog message identifiers. An entity was classified as *grounded* if it appeared in any successful tool-call result, and *ungrounded* (hallucinated) otherwise. The primary metric was grounding coverage: the fraction of response entities with tool-backed evidence.

Abstention scoring (RQ3) evaluated the agent's behavior on unanswerable scenarios, a well-established problem in selective prediction and unanswerable question answering [54, 55, 92], recently extended to RAG and LLM evaluation settings [56, 93]. Each scenario carried a binary answerability flag in its stratification metadata. A scenario was scored as correct if the agent abstained on unanswerable queries and provided a substantive response on answerable ones.

Efficiency metrics recorded resource consumption per execution: total tool-call count, token usage (input and output), and wall-clock latency derived from per-tool-call timing data in the extraction records. These metrics supported cross-model and cross-configuration efficiency comparisons.

3.4 Reproducibility and Validity

Run directory and artifact layout. Each benchmark run produced a self-contained, timestamped directory preserving all pipeline artifacts:

- `scenarios.yaml`: the generated scenario set;
- `run_config.json`: benchmark configuration snapshot;
- `raw_output/`: per-execution manifests;
- `extractions/`: structured trace records;
- `verdicts/`: per-scenario primary judge verdicts;
- `verdicts_calibration/`: calibration judge verdicts;
- `verdicts_meta/`: meta-judge arbitrated verdicts;
- `metrics/`: aggregated scores as JSON, CSV, and LaTeX tables;
- `plots/`: PDFs, PNGs for plots.

This layout ensured traceability from reported figures to the underlying verdicts and executions.

Reproducibility controls. Reproducibility depended on associating each run with a fully specified configuration. RAG chunks carried a SHA-256 content hash (stored generated column in PostgreSQL), uniquely identifying each indexed chunk's textual content. Generation and embedding models were fixed in `config.toml` and validated against an allowlist at startup. Scenario generation was deterministic given a fixed database snapshot and configuration; the scenario set was persisted at run start and reused across all stages.

Each verdict recorded the evaluator model identifier alongside the SUT model identifier, enabling post-hoc analysis of evaluator-specific effects. The benchmark configuration file was identified by its git commit hash at evaluation time, and retrieval parameters (Tables 3.4, 3.5, and 3.6) were held constant across all reported runs.

Validity considerations. Four validity dimensions shaped the evaluation design.

Construct validity was addressed by assessing recommendations through both IR metrics and grounded evidence checks rather than answer text alone. This dual assessment prevents a system from scoring well by producing plausible but unsupported responses.

Internal validity was supported by deterministic post-processing for identifier extraction and IR scoring. The LLM judge's evaluator variance was bounded by model separation (Section 3.3), context isolation, structured output constraints [59], and calibration/meta-judge stages (Sections 3.3, 3.3) reporting Cohen's κ [86] and Krippendorff's α [87].

Conclusion validity was bounded by scenario-level reporting and metric aggregates with explicit coverage notes. Each scenario–configuration cell was executed once, so the evaluation spans 936 executions but yields a single observation per cell rather than repeated samples. Bootstrap confidence intervals and Holm–Bonferroni correction were applied to the judge-agreement statistics (Section 3.3); the configuration-level percentage-point differences in Chapter 4 are reported descriptively, without per-contrast hypothesis tests. At $n = 52$ scenarios per cell, a binomial 95% confidence interval on a pass rate spans up to roughly ± 13 percentage points, so small cross-configuration differences should be read as directional observations rather than established effects.

External validity is limited to the proprietary industrial artifacts, prototype system, and runtime constraints used here. Generalization to other artifact types, domains, or organizations requires further evaluation; key variables include artifact vocabulary overlap, graph schema completeness, and the availability of pre-existing trace links. The architecture, tool-interface constraints, and evaluation protocol are domain-agnostic in design and could be instantiated over different schemas, but this transferability has not been empirically validated.

4 Results

This chapter presents the outcomes of 936 benchmark executions spanning 18 configurations. Section 4.1 briefly restates the setup needed to read the tables and figures. Sections 4.2–4.4 address each research question in turn. Section 4.5 reports cross-cutting analyses. Interpretation is deferred to Chapter 5.

4.1 Evaluation Setup Recap

The benchmark comprised 52 evaluation scenarios stratified by retrieval family (comparison, impact, lookup, search, traceability), complexity (single-hop, multi-hop, reasoning), and answerability. Each scenario was executed across three LLM back-ends (Table 3.7), two workflow families (ReAct and Plan-and-Execute), and three knowledge-augmentation modes (Prompt, Skill, Skill+Meta), yielding 18 configurations per scenario and 936 individual executions. The full experimental design is described in Section 3.3.

The result sections map directly to the research questions. RQ1 concerns whether the scoring protocol is valid and how LLM-as-a-judge scores relate to IR metrics. RQ2 concerns task correctness, evidence quality, knowledge augmentation, and workflow effects. RQ3 concerns model characteristics and operational reliability.

Four related evaluation terms are used throughout the results. *Functional pass rate* denotes the deterministic task-correctness verdict used in the main RQ2 tables. *Judge status pass rate* denotes the LLM judge's pass/fail status. *Judge overall score* denotes the judge's continuous $[0, 1]$ score. *Mean dimension score* denotes the average 1–5 Likert quality score across Completeness, Relevance, Coherence, and Evidence Grounding. Unqualified "pass rate" in this chapter refers to functional pass rate unless explicitly labelled as judge status.

A per-turn timeout of 1 200 s was enforced. Table 4.1 reports the fraction of passing scenarios retained at progressively tighter wall-clock thresholds. At 180 s, 65.0% of successful executions had completed; this fraction rose to 80.1% at 300 s. Qwen3-Coder was the fastest model, with 61.8% of its passing runs finishing within 60 s, while DeepSeek-V3.2 was the slowest at 4.7%.

Table 4.1: Fraction of passing executions completed within progressively tighter wall-clock thresholds, by model.

Threshold (s)	DeepSeek-V3.2	GLM-5.1	Qwen3-Coder	Overall
60	4.7%	24.2%	61.8%	28.5%
120	26.8%	46.4%	85.3%	51.1%
180	40.9%	65.7%	92.6%	65.0%
240	53.7%	77.4%	96.3%	74.7%
300	60.7%	83.5%	99.1%	80.1%

4.2 Scoring-Protocol Validity (RQ1)

RQ1 asks how the benchmark and scoring protocol are validated, and how LLM-as-a-judge scores relate to traditional IR metrics.

Construct validity of the scoring protocol was assessed through judge–IR correlation, distributional analysis, agreement statistics, and a Bland–Altman complementarity analysis. Before examining these relationships, Tables 4.2 and 4.3 present the two metric families that the protocol must relate.

Retrieval metrics. Table 4.2 reports rank-aware retrieval quality metrics at multiple cutoff depths for all 18 configurations. Recall@1 ranged from 0.285 (DeepSeek-V3.2 ReAct Prompt) to 0.471 (Qwen3-Coder Plan-and-Execute Prompt), indicating that the top-ranked result contained a relevant artifact in 28–47% of scenarios. Recall improved substantially at deeper cutoffs: Recall@5 reached 0.655 (GLM-5.1 ReAct Skill+Meta) and Recall@10 peaked at 0.842 (DeepSeek-V3.2 Plan-and-Execute Skill). The gap between Recall@1 and Recall@10 was largest for DeepSeek-V3.2 Plan-and-Execute Skill ($\Delta = 0.383$), suggesting that this configuration retrieved many relevant artifacts but ranked them below the first position.

Plan-and-Execute configurations consistently outperformed their ReAct counterparts on NDCG: mean NDCG@10 under Plan-and-Execute ranged from 0.544 to 0.733, versus 0.432–0.626 under ReAct. MRR@10 showed a similar pattern, with Plan-and-Execute achieving 0.561–0.742 compared to 0.488–0.682 for ReAct, indicating that Plan-and-Execute placed the first relevant artifact higher in the ranking.

Judge quality assessment. Table 4.3 reports LLM-as-Judge status pass rates, judge overall scores on $[0, 1]$, and per-dimension quality scores on 1–5 Likert scales. GLM-5.1 under Plan-and-Execute with Prompt augmentation achieved the highest judge status pass rate (80.8%), while the highest judge overall score (0.880) was recorded under Skill+Meta. Coherence was the highest-scoring dimension across all configurations (3.50–4.77), while Completeness exhibited the widest variation (3.12–4.35). Judge status pass rates differ from functional pass rates because the judge status requires all expected identifiers, at least 80% of expected entities, and reference-answer alignment of at least 0.7, whereas the functional pass rate is computed by the deterministic scoring stage.

Judge–IR correlation. Table 4.4 reports Spearman rank correlations between the four LLM judge quality dimensions and four deterministic IR metrics, computed over all executions that produced scored retrieval evidence. All correlations reached statistical significance at $p < 0.001$. Relevance achieved the highest coefficients, peaking at $\rho = 0.526$ against NDCG@10. Completeness followed with coefficients in the range $\rho = 0.271$–0.394. Coherence and Evidence Grounding occupied a middle band ($\rho = 0.289$–0.412). Cross-family correlations fell in the range $\rho = 0.27$–0.53, while within-family correlations exceeded $\rho = 0.7$, indicating shared but not redundant quality signals between the two metric families.

Table 4.2: Retrieval metrics per model, agentic workflow, and knowledge-augmentation mode. Mean values over $n = 52$ evaluation scenarios (each a distinct retrieval task over the 11,366-case corpus). Recall@k reports the fraction of known relevant artifacts appearing in the top-k retrieved results.

Model	Wkfl.	Aug.	R@1	R@5	R@10	NDCG@5	NDCG@10	MRR@10
DS	P&E	Prompt	0.420	0.610	0.666	0.592	0.615	0.598
	P&E	Skill	0.459	0.628	0.842	0.660	0.733	0.699
	P&E	Skill+Meta	0.456	0.633	0.714	0.670	0.692	0.742
	ReAct	Prompt	0.285	0.423	0.449	0.443	0.437	0.488
	ReAct	Skill	0.293	0.397	0.424	0.441	0.432	0.496
	ReAct	Skill+Meta	0.301	0.458	0.482	0.478	0.464	0.516
GLM	P&E	Prompt	0.453	0.521	0.595	0.576	0.597	0.632
	P&E	Skill	0.355	0.557	0.709	0.646	0.695	0.724
	P&E	Skill+Meta	0.447	0.607	0.700	0.633	0.660	0.672
	ReAct	Prompt	0.462	0.615	0.620	0.627	0.613	0.682
	ReAct	Skill	0.340	0.512	0.512	0.497	0.487	0.547
	ReAct	Skill+Meta	0.466	0.655	0.661	0.632	0.626	0.669
Qwen	P&E	Prompt	0.471	0.615	0.688	0.674	0.689	0.707
	P&E	Skill	0.362	0.524	0.557	0.532	0.544	0.561
	P&E	Skill+Meta	0.413	0.547	0.610	0.608	0.615	0.657
	ReAct	Prompt	0.350	0.439	0.480	0.460	0.472	0.537
	ReAct	Skill	0.334	0.445	0.445	0.474	0.459	0.543
	ReAct	Skill+Meta	0.410	0.524	0.524	0.545	0.538	0.609

DS = DeepSeek-V3.2, GLM = GLM-5.1, Qwen = Qwen3-Coder. Wkfl. = Workflow, Aug. = Augmentation.

Table 4.3: LLM-as-Judge results per model, agentic workflow, and knowledge-augmentation mode. Judge status pass rate, judge overall score on $[0, 1]$, and mean quality dimension scores (1–5 Likert) over $n = 52$ evaluation scenarios (each a distinct retrieval task over the 11,366-case corpus).

Model	Wkfl.	Aug.	Judge pass	Overall	Compl.	Relev.	Coher.	Evid. Gr.
DS	P&E	Prompt	65.4%	0.834	4.10	4.04	4.62	4.38
	P&E	Skill	76.9%	0.872	4.27	3.90	4.77	4.33
	P&E	Skill+Meta	76.9%	0.852	4.33	4.10	4.75	4.46
	ReAct	Prompt	69.2%	0.829	4.06	4.19	4.67	4.38
	ReAct	Skill	67.3%	0.794	3.98	3.98	4.56	4.13
	ReAct	Skill+Meta	67.3%	0.807	3.90	3.96	4.56	4.12
GLM	P&E	Prompt	80.8%	0.860	4.31	4.29	4.65	4.46
	P&E	Skill	71.2%	0.818	4.19	4.23	4.46	4.25
	P&E	Skill+Meta	76.9%	0.880	4.35	4.42	4.71	4.56
	ReAct	Prompt	67.3%	0.778	3.79	4.08	4.35	4.21
	ReAct	Skill	55.8%	0.628	3.12	3.25	3.50	3.19
	ReAct	Skill+Meta	63.5%	0.730	3.62	3.75	4.10	3.85
Qwen	P&E	Prompt	59.6%	0.763	3.50	3.90	4.42	3.90
	P&E	Skill	55.8%	0.713	3.19	3.60	4.38	3.63
	P&E	Skill+Meta	53.8%	0.732	3.25	3.94	4.46	3.73
	ReAct	Prompt	48.1%	0.673	3.15	3.83	4.40	3.58
	ReAct	Skill	48.1%	0.678	3.15	3.77	4.25	3.48
	ReAct	Skill+Meta	48.1%	0.693	3.13	4.04	4.38	3.65

DS = DeepSeek-V3.2, GLM = GLM-5.1, Qwen = Qwen3-Coder. Wkfl. = Workflow, Aug. = Augmentation.

Table 4.4: Spearman rank correlations (ρ) between LLM judge quality dimensions and deterministic IR metrics. All correlations significant at $p < 0.001$ ($\star\star\star$).

Dimension	NDCG@5	NDCG@10	MRR@10	Recall@10
Completeness	0.348^{***}	0.383^{***}	0.271^{***}	0.394^{***}
Relevance	0.509^{***}	0.526^{***}	0.414^{***}	0.468^{***}
Coherence	0.354^{***}	0.383^{***}	0.289^{***}	0.393^{***}
Evidence Grounding	0.385^{***}	0.412^{***}	0.332^{***}	0.390^{***}

Per-scenario judge scores showed an upward trend against all four IR metrics across complexity levels (Figure 4.1), consistent with the moderate positive correlations in Table 4.4. Substantial scatter around the regression line persists in every subplot, indicating that the judge captures quality aspects not fully explained by retrieval rank alone.

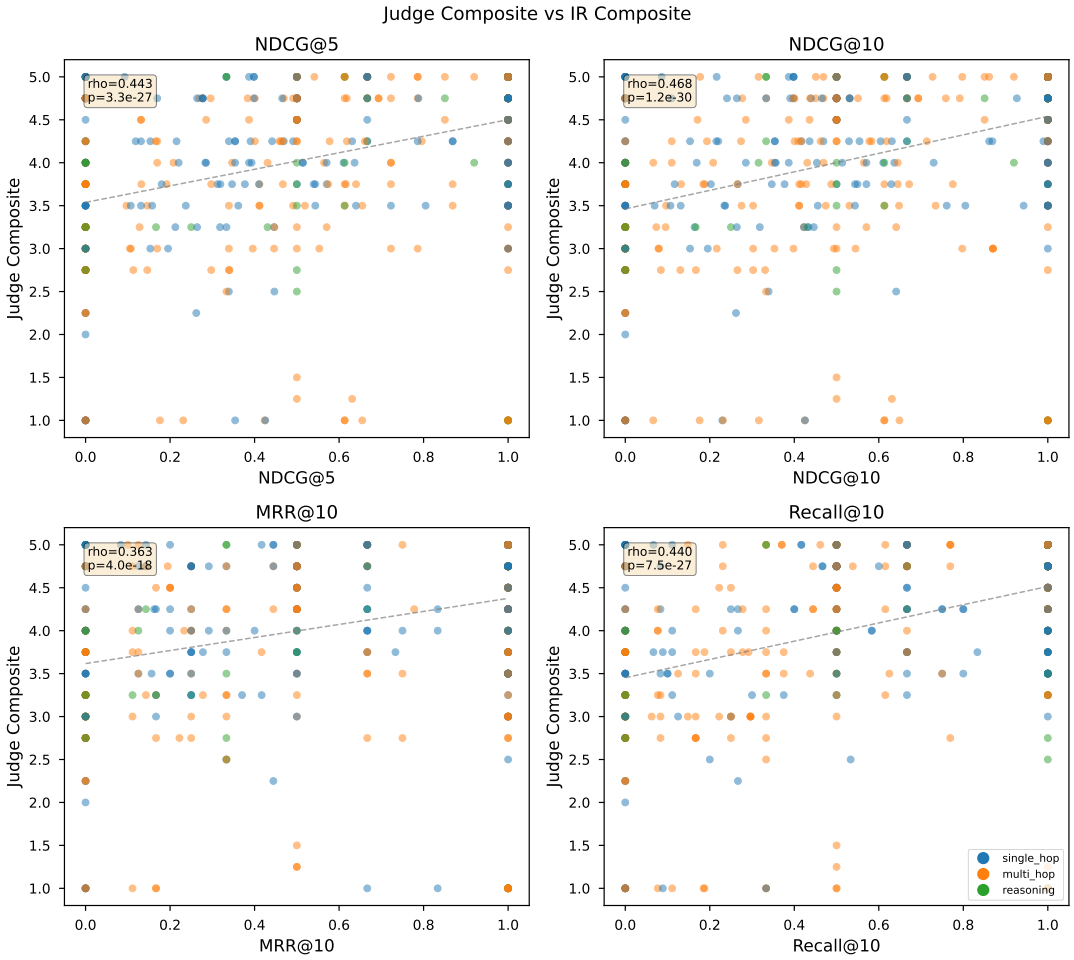


Figure 4.1: Scatter plot of per-scenario judge composite scores versus IR metrics, split by metric (NDCG@5, NDCG@10, MRR@10, Recall@10). Spearman ρ and p-values are annotated per metric panel; points are colored by scenario complexity level.

Score distributions. Mean dimension scores exhibited a bimodal distribution when grouped by functional verdict outcome (Figure 4.2). Functionally passing executions ($n = 722$) clustered at the upper end of the 1–5 score range, with a dominant mode near 5.0. Functionally failing

executions ($n = 214$) concentrated around 1.0, with a secondary spread through the 3.0–4.0 range. This separation indicates that the judge quality dimensions discriminate well between passing and failing executions, although overlap in the mid-range suggests that borderline cases receive intermediate scores regardless of verdict.

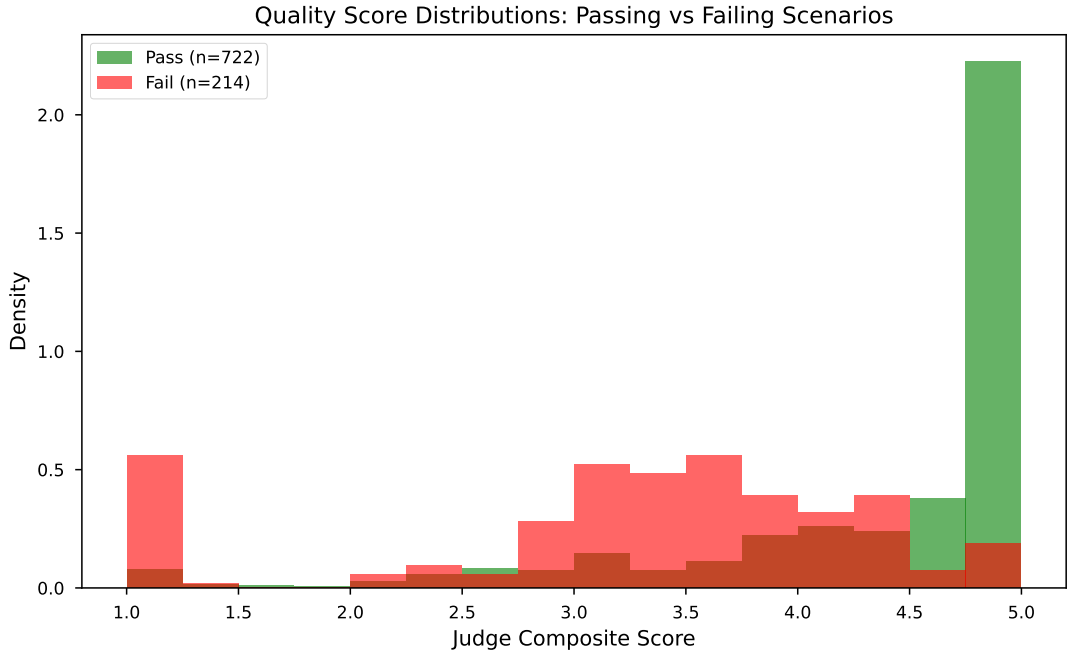
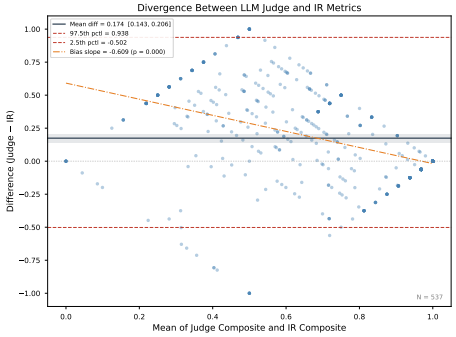


Figure 4.2: Distribution of mean judge dimension scores across all benchmark executions, grouped by functional verdict outcome (pass vs. fail).

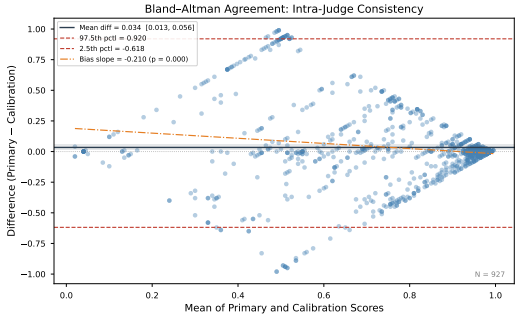
Bland–Altman complementarity analysis. Two Bland–Altman analyses quantified systematic and random divergence between measurement approaches (Figure 4.3). The inter-method comparison (judge composite vs. normalized IR score) revealed a positive mean bias and a significant negative proportional-bias slope, indicating that the judge is more lenient than IR metrics at low scores and converges at high scores. Nonparametric limits of agreement (2.5th/97.5th percentile) and bootstrap confidence intervals on the mean difference are annotated in the figure.

The intra-judge comparison (primary vs. calibration scoring round) showed a near-zero mean difference. This panel uses $N = 624$ per-dimension score pairs derived from the 156 paired executions (each execution yields four scored dimensions). The contrast between the two panels, substantial inter-method divergence alongside near-zero intra-judge bias, is examined further in Chapter 5.

Intra-judge agreement. Table 4.5 reports agreement between the primary and calibration scoring rounds over $n = 156$ paired executions. Cohen's $\kappa = 0.844$ on binary verdict indicates near-perfect classification agreement. Pearson $r = 0.986$ and Spearman $\rho = 0.903$ on the overall composite score confirm strong linear and rank-order consistency. Per-dimension Krippendorff α exceeded 0.95 for all four quality dimensions, indicating excellent intra-rater reliability on each quality dimension.



(a) Inter-method: judge composite vs. normalized IR score.



(b) Intra-judge: primary vs. calibration scoring round.

Figure 4.3: Bland–Altman divergence analysis with nonparametric limits of agreement, proportional-bias regression, and bootstrap confidence interval on the mean difference.

Table 4.5: Intra-judge agreement between primary and calibration scoring rounds ($n = 156$ paired executions). Interpretation scales: κ Landis & Koch (1977); r/ρ Evans (1996); α Krippendorff (2011).

Metric	Value
Cohen's κ (verdict)	0.844
Pearson r (overall score)	0.986
Spearman ρ (overall score)	0.903
Krippendorff α (Completeness)	0.958
Krippendorff α (Relevance)	0.971
Krippendorff α (Coherence)	0.985
Krippendorff α (Evidence Grounding)	0.977

4.3 Task Correctness and Evidence Quality (RQ2)

RQ2 asks how knowledge augmentation affects correctness, retrieval quality, and grounding, and how workflow choice moderates those effects.

With the scoring protocol's reliability established, this section applies it alongside the metrics from Section 4.2 to evaluate task correctness and evidence quality across configurations.

Pass rate by configuration. Plan-and-Execute configurations achieved higher aggregate functional pass rates than ReAct for GLM-5.1 and near-parity for DeepSeek-V3.2, but not for Qwen3-Coder (Figure 4.4). GLM-5.1 exhibited the highest rates across both workflows: 82.7–90.4% under Plan-and-Execute and 65.4–76.9% under ReAct. DeepSeek-V3.2 Plan-and-Execute rates ranged from 76.9% to 88.5%, while its ReAct rates were more uniform at 80.8–84.6%. Qwen3-Coder occupied the narrowest band overall (65.4–73.1%), with ReAct exceeding Plan-and-Execute in every matched augmentation condition.

Knowledge-augmentation effects. The effect of knowledge-augmentation mode on pass rate was model-dependent. Under Plan-and-Execute, DeepSeek-V3.2 improved from 76.9% (Prompt) to 88.5% (Skill), a gain of 11.6 percentage points. GLM-5.1 showed the opposite pattern: Prompt achieved 90.4% while Skill dropped to 82.7%, a loss of 7.7 percentage points. Qwen3-Coder exhibited a similar but smaller decline (71.2% Prompt vs. 67.3% Skill). Under ReAct, all three models showed stable or declining pass rates when moving from Prompt to Skill or Skill+Meta, with GLM-5.1 recording the largest drop (76.9% Prompt vs. 65.4% Skill,

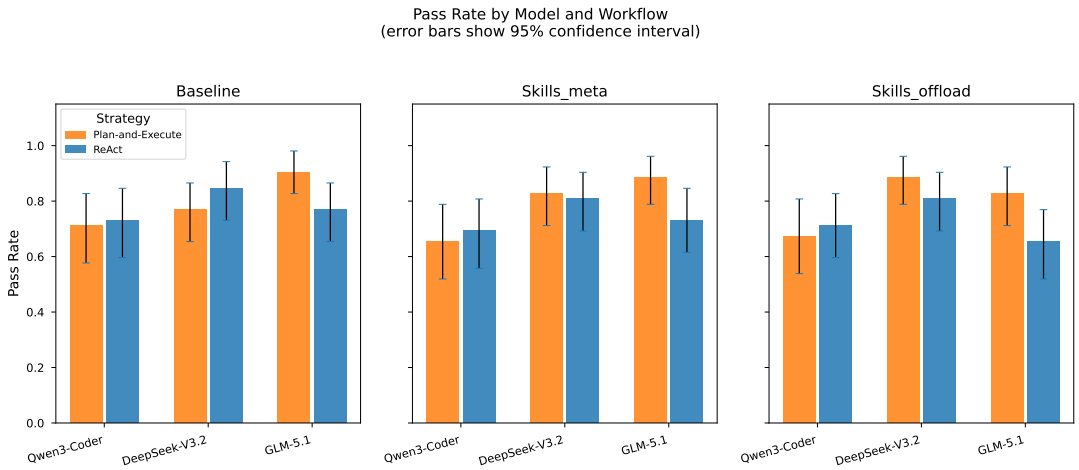


Figure 4.4: Pass rate by model, workflow, and knowledge-augmentation mode. Facet labels correspond to augmentation modes: Baseline = Prompt, Skills_meta = Skill+Meta, Skills_offload = Skill.

−11.5 percentage points). Because each scenario–configuration cell was executed once, these percentage-point contrasts are directional observations rather than tested effects (Section 3.4).

Success heatmap. An aggregated functional pass-rate heatmap (Figure 4.5) summarized performance across models and workflow families. GLM-5.1 under Plan-and-Execute achieved the highest pass count (136/156, 87%), while Qwen3-Coder under Plan-and-Execute recorded the lowest (106/156, 68%). DeepSeek-V3.2 showed the smallest gap between workflows (83% vs. 82%). GLM-5.1 exhibited the largest positive Plan-and-Execute gap (87% vs. 72%), and Qwen3-Coder was lower under Plan-and-Execute than under ReAct (68% vs. 71%). To disentangle scenario difficulty from retrieval family, the following paragraph disaggregates results by complexity level.

Complexity breakdown. Table 4.6 disaggregates pass rate and NDCG@10 by scenario complexity. Reasoning scenarios achieved the highest pass rates under both workflows (94.4% Plan-and-Execute, 92.4% ReAct), despite their apparent conceptual difficulty; this pattern reflects the scenario design and is examined in Section 5. Multi-hop scenarios were the most difficult at 67.8% (Plan-and-Execute) and 63.3% (ReAct). Single-hop scenarios fell between the two at 78.5% and 72.2% respectively. NDCG@10 values were consistently higher under Plan-and-Execute than ReAct across all complexity tiers.

Table 4.6: Pass rate and mean NDCG@10 disaggregated by scenario complexity and workflow family.

		Plan-and-Execute		ReAct	
Complexity	n	Pass rate	NDCG@10	Pass rate	NDCG@10
Multi-hop	180	67.8%	0.646	63.3%	0.498
Reasoning	144	94.4%	0.684	92.4%	0.534
Single-hop	144	78.5%	0.640	72.2%	0.498

Quality dimensions. Per-dimension judge scores revealed that Coherence was the highest-scoring dimension across configurations, while Evidence Grounding was the lowest (Fig-

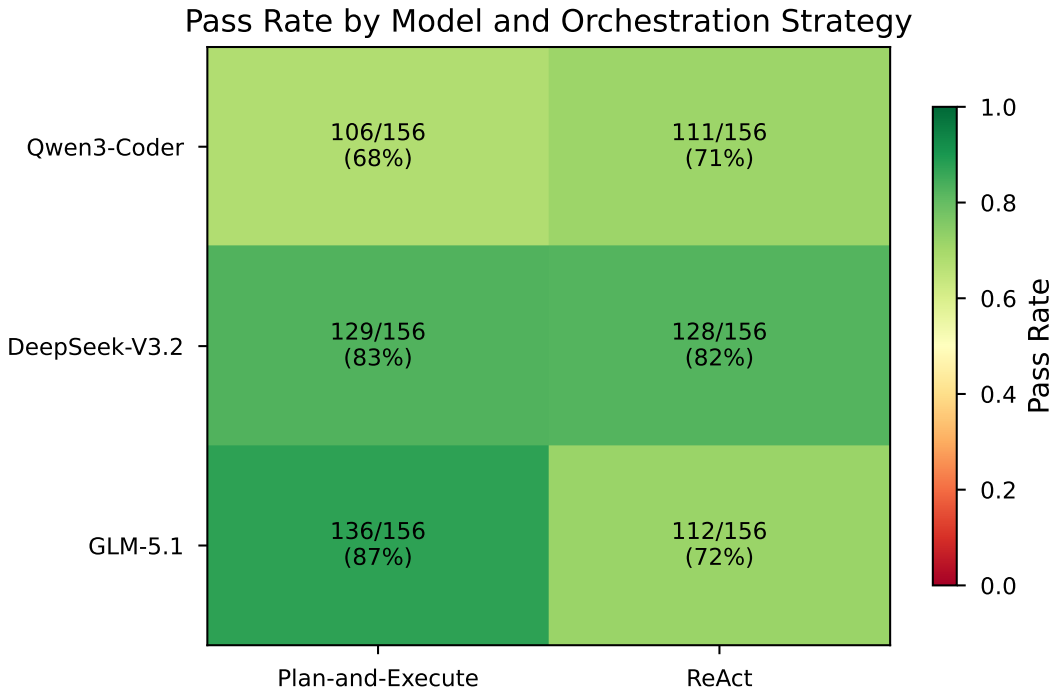


Figure 4.5: Aggregated pass-rate heatmap by model and workflow family. Cell annotations show pass count and percentage out of 156 executions per cell.

ures 4.6–4.8). Plan-and-Execute configurations scored higher than their ReAct counterparts in 30 of 36 matched dimension comparisons (9 model–augmentation pairs $\times$ 4 dimensions; Table 4.3), with the largest margins on Completeness and Evidence Grounding. The exceptions concentrate in the Relevance dimension (four of six), most notably DeepSeek-V3.2 under Prompt, where ReAct scored higher on Relevance and Coherence. The radar charts visualize per-model dimension profiles grouped by workflow family.

Evidence grounding and answer quality. Figure 4.9 plots grounding coverage against hallucinated identifier count for Plan-and-Execute configurations; grounding is computed for Plan-and-Execute only because ReAct produces no recoverable ranked artifact list (Section 3.3). Higher grounding coverage co-occurred with fewer hallucinated identifiers across all three models, following a decaying trend. Executions with near-zero grounding coverage exhibited the widest spread in hallucination count (up to $\sim$60 hallucinated identifiers), while executions above 0.5 grounding coverage rarely exceeded 15.

Execution time. Qwen3-Coder under ReAct exhibited the shortest mean execution times (30.3–31.5 s), while DeepSeek-V3.2 under Plan-and-Execute was the slowest (415.6–520.3 s) (Figure 4.10). GLM-5.1 under Plan-and-Execute recorded intermediate times (166.7–201.8 s). Within each model, ReAct configurations completed 5–17$\times$ faster than Plan-and-Execute.

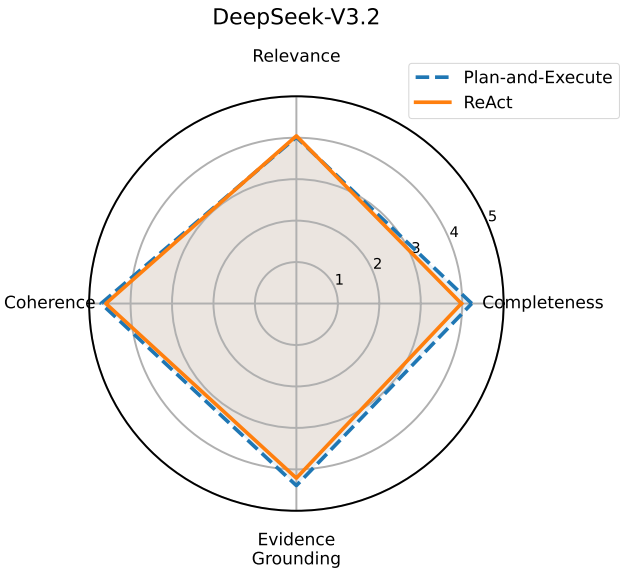


Figure 4.6: Judge quality dimensions for DeepSeek-V3.2 by workflow family.

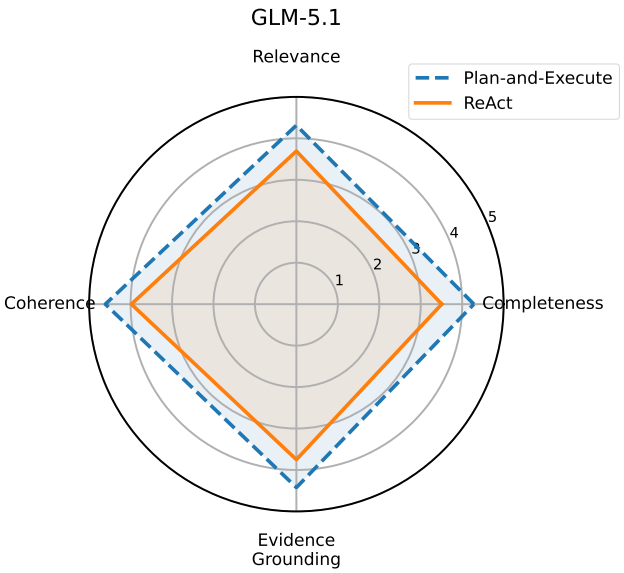


Figure 4.7: Judge quality dimensions for GLM-5.1 by workflow family.

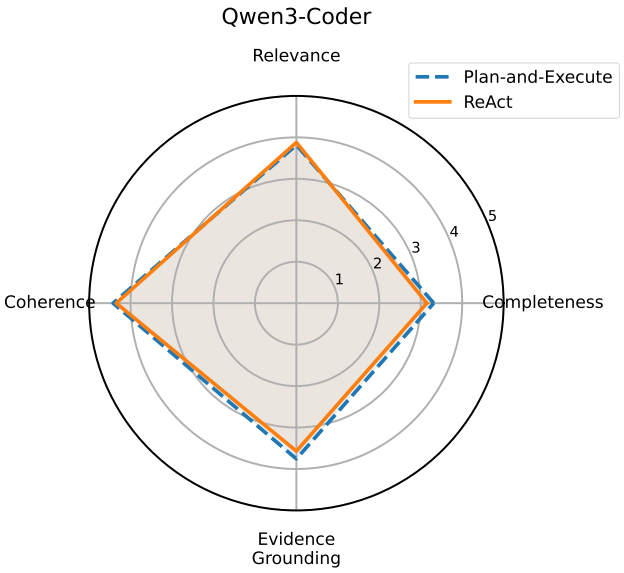


Figure 4.8: Judge quality dimensions for Qwen3-Coder by workflow family.

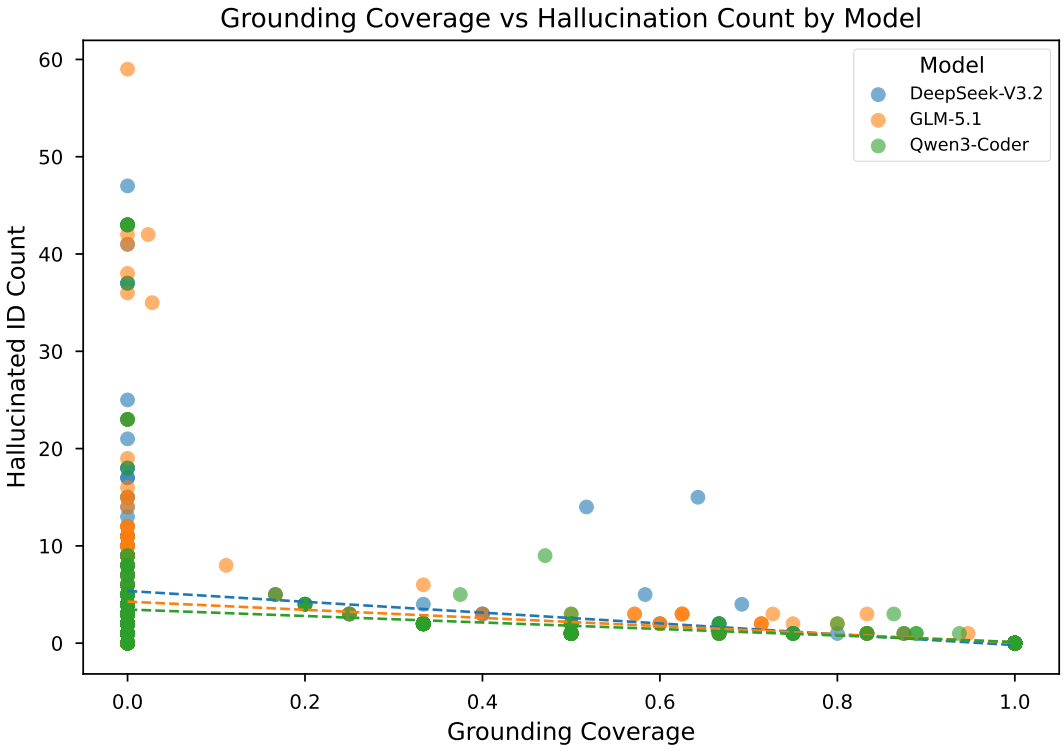


Figure 4.9: Grounding coverage versus hallucinated identifier count for Plan-and-Execute configurations. Each point is one execution; dashed lines show per-model regression trends.

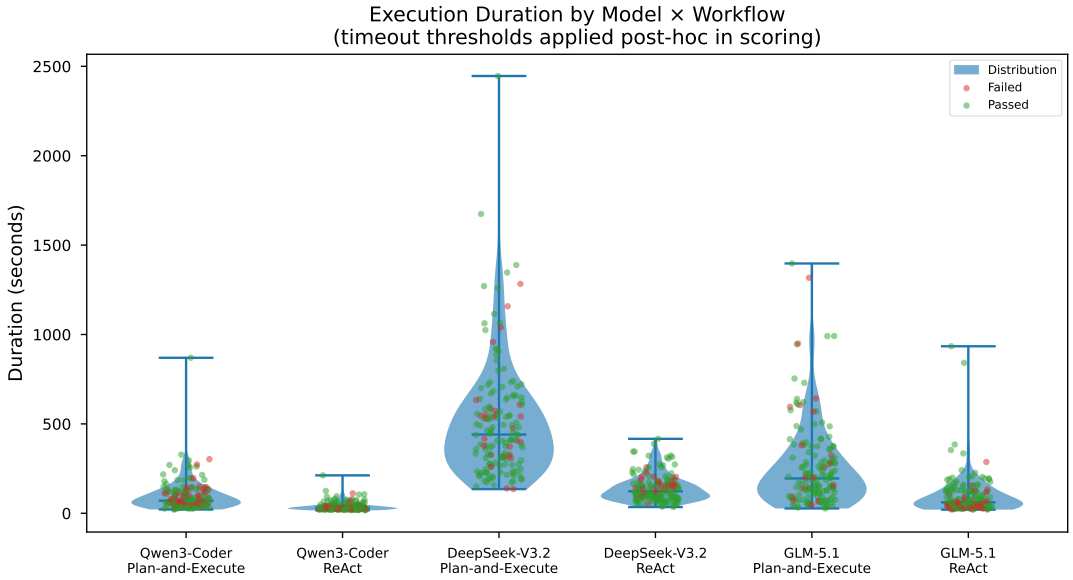


Figure 4.10: Violin plots of per-scenario execution time by model and workflow. Distribution shapes reveal multi-modal patterns for Plan-and-Execute configurations.

4.4 Operational Reliability (RQ3)

RQ3 asks how model characteristics affect reliable tool use and evidence-grounded retrieval performance.

Tool reliability. Table 4.7 reports tool-call success rates and mean evidence grounding by model and workflow. Under Plan-and-Execute, DeepSeek-V3.2 achieved a perfect 100.0% success rate across 7 760 calls, while Qwen3-Coder recorded the lowest Plan-and-Execute rate at 92.6% across 3 292 calls. Under ReAct, success rates were lower across all models, with Qwen3-Coder recording 86.0% across 1 076 calls. Mean grounding scores are reported only for Plan-and-Execute configurations (0.384–0.504). This difference arises from the extraction method described in Section 3.3: Plan-and-Execute sub-agents produce structured artifact files from which ranked retrieval lists can be recovered, whereas ReAct accumulates retrieved content within a single context window without producing an explicit ranked list.

Table 4.7: Tool-call success rates and mean evidence grounding by model and workflow family. Grounding is not measured for ReAct because the extraction pipeline lacks explicit ranked artifact lists for that workflow.

Model	Workflow	Total calls	Success	Error	Grounding
DeepSeek-V3.2	P&E	7 760	100.0%	0.0%	0.504
DeepSeek-V3.2	ReAct	2 324	99.9%	0.1%	N/A
GLM-5.1	P&E	7 290	97.6%	2.4%	0.384
GLM-5.1	ReAct	1 962	96.4%	3.6%	N/A
Qwen3-Coder	P&E	3 292	92.6%	7.4%	0.495
Qwen3-Coder	ReAct	1 076	86.0%	14.0%	N/A

Retrieval family pass rate. Pass rates varied substantially across retrieval families (Figure 4.11). Lookup scenarios achieved the highest overall rate (95.0%), followed by traceability (94.1%). Search scenarios were the most challenging at 54.0%, with comparison and impact scenarios at

66.7% each. Model-level variation was most pronounced for search and comparison scenarios, where the spread between the best and worst model exceeded 20 percentage points.

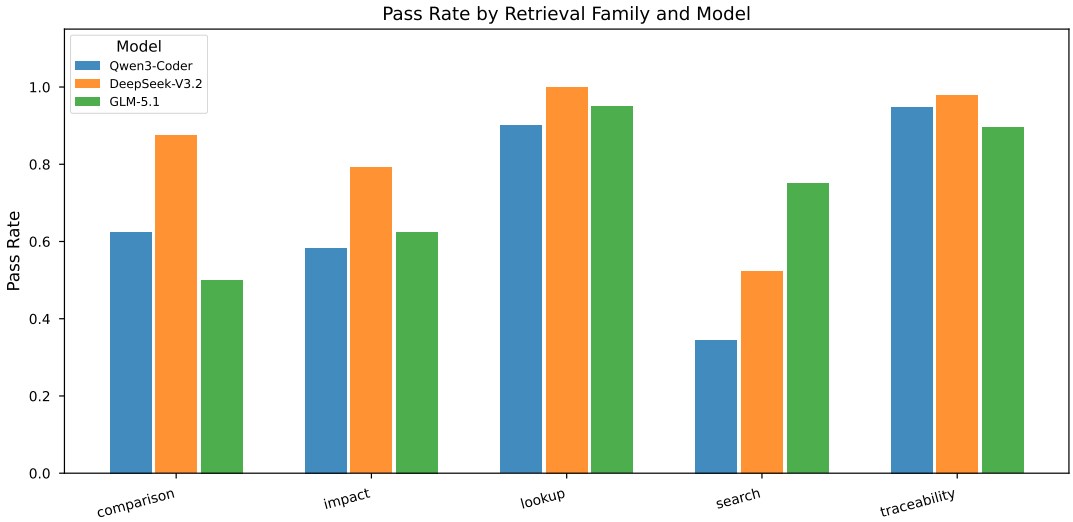


Figure 4.11: Pass rate by retrieval family and model.

Token consumption. DeepSeek-V3.2 consumed substantially more input tokens than the other models within each workflow (Figure 4.12; filtered to executions $\leqslant 600$ s). Under Plan-and-Execute, DeepSeek-V3.2 median input tokens were 1.30–1.62 M unfiltered (Table 4.9), or $\sim$1.14 M within the $\leqslant 600$ s-filtered distribution shown in the figure, compared with 325–432 K for Qwen3-Coder and 652–753 K for GLM-5.1 (unfiltered medians). Under ReAct, token counts were lower overall, with Qwen3-Coder at 142–188 K and DeepSeek-V3.2 at 415–508 K. Plan-and-Execute token figures include both orchestrator and sub-agent consumption.

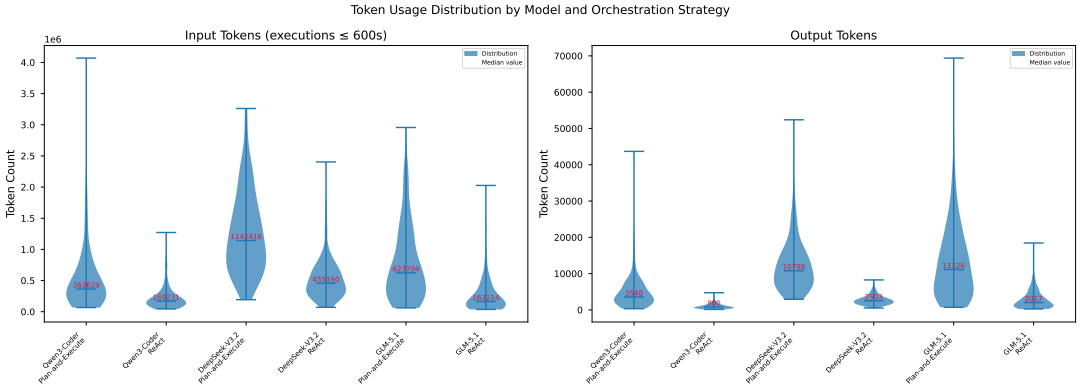


Figure 4.12: Distribution of input and output token counts by model and workflow.

Abstention accuracy. Beyond correctness on answerable queries, the system's ability to abstain on unanswerable scenarios is a separate reliability concern. Table 4.8 reports abstention behavior on 14 unanswerable and 38 answerable scenarios per configuration. Correct and missed abstention rates use the 14 unanswerable scenarios as their denominator; false-abstention rates use the 38 answerable scenarios as their denominator. Missed abstention (the failure to withhold an answer when the knowledge base lacked sufficient evidence) was the dominant

error mode, ranging from 42.9% to 71.4% across configurations. DeepSeek-V3.2 under ReAct with Skill+Meta augmentation achieved the highest correct abstention rate (57.1%, 8/14), while DeepSeek-V3.2 under Plan-and-Execute with Prompt recorded the lowest (28.6%, 4/14). False abstention rates (incorrectly withholding an answer to an answerable query) remained below 26.3% for all configurations, with most falling below 11%. With only 14 unanswerable scenarios per configuration, a single scenario shifts an abstention rate by 7.1 percentage points; per-configuration differences of one or two scenarios should therefore not be over-interpreted.

Table 4.8: Abstention behaviour per configuration. Correct and Missed use the 14 unanswerable scenarios as denominator; False uses the 38 answerable scenarios as denominator ($n = 52$ total, over the 11,366-case corpus). Correct = correctly withheld answer; False = incorrectly abstained on answerable query; Missed = failed to abstain when answer was unavailable.

Model	Wkfl.	Aug.	n	Correct /14	False /38	Missed /14
DS	P&E	Prompt	52	28.6%	13.2%	71.4%
	P&E	Skill	52	35.7%	2.6%	64.3%
	P&E	Skill+Meta	52	35.7%	0.0%	64.3%
	ReAct	Prompt	52	50.0%	26.3%	50.0%
	ReAct	Skill	52	50.0%	5.3%	50.0%
	ReAct	Skill+Meta	52	57.1%	5.3%	42.9%
GLM	P&E	Prompt	52	35.7%	2.6%	64.3%
	P&E	Skill	52	42.9%	5.3%	57.1%
	P&E	Skill+Meta	52	42.9%	2.6%	57.1%
	ReAct	Prompt	52	50.0%	21.1%	50.0%
	ReAct	Skill	52	42.9%	26.3%	57.1%
	ReAct	Skill+Meta	52	35.7%	23.7%	64.3%
Qwen	P&E	Prompt	52	42.9%	10.5%	57.1%
	P&E	Skill	52	42.9%	0.0%	57.1%
	P&E	Skill+Meta	52	28.6%	5.3%	71.4%
	ReAct	Prompt	52	42.9%	13.2%	57.1%
	ReAct	Skill	52	42.9%	5.3%	57.1%
	ReAct	Skill+Meta	52	42.9%	10.5%	57.1%

DS = DeepSeek-V3.2, GLM = GLM-5.1, Qwen = Qwen3-Coder. Wkfl. = Workflow, Aug. = Augmentation.

4.5 Cross-Cutting Analyses

Efficiency. Table 4.9 reports median tool calls and token consumption for all 18 configurations. Tool-call counts ranged from 4 (Qwen3-Coder ReAct Prompt) to 48 (DeepSeek-V3.2 Plan-and-Execute Skill). Input token consumption spanned more than an order of magnitude, from 142 K (Qwen3-Coder ReAct Prompt) to 1.62 M (DeepSeek-V3.2 Plan-and-Execute Skill). The largest values are consistent with DeepSeek-V3.2's exhaustive exploration behavior, discussed in Section 5.1, and are amplified under Plan-and-Execute because token accounting includes both orchestrator and sub-agent activity. Output tokens were uniformly low under ReAct (777–2 669) and higher under Plan-and-Execute (3 108–12 321).

The cumulative distribution of execution time (Figure 4.13) shows that 80.1% of passing executions completed within 300 s. Qwen3-Coder ReAct configurations contributed the steepest initial rise, with over 60% of runs completing within 60 s. DeepSeek-V3.2 Plan-and-Execute configurations formed a long tail extending beyond 500 s.

Stratification coverage. Table 4.10 reports pass rate and mean NDCG@10 disaggregated by each stratification variable across all configurations. Among complexity levels, reasoning scenarios achieved the highest pass rate (93.4%), followed by single-hop (75.3%) and multi-hop (65.6%). Single-hop scenarios recorded the highest mean NDCG@10 (0.525), while multi-hop

Table 4.9: Median tool calls and token consumption per configuration.

Model	Workflow	Augmentation	Tools	In tokens	Out tokens
DeepSeek-V3.2	P&E	Prompt	46	1 504 719	11 580
	P&E	Skill	48	1 619 462	10 922
	P&E	Skill+Meta	40	1 297 872	10 548
	ReAct	Prompt	12	414 850	2 306
	ReAct	Skill	14	456 508	2 669
	ReAct	Skill+Meta	14	507 540	2 608
GLM-5.1	P&E	Prompt	36	651 610	12 321
	P&E	Skill	47	752 589	11 491
	P&E	Skill+Meta	30	665 642	9 938
	ReAct	Prompt	6	117 973	1 701
	ReAct	Skill	12	198 277	2 313
	ReAct	Skill+Meta	10	183 415	2 077
Qwen3-Coder	P&E	Prompt	14	328 116	3 497
	P&E	Skill	19	431 743	4 076
	P&E	Skill+Meta	14	325 862	3 108
	ReAct	Prompt	4	142 422	838
	ReAct	Skill	6	175 176	798
	ReAct	Skill+Meta	6	188 288	777

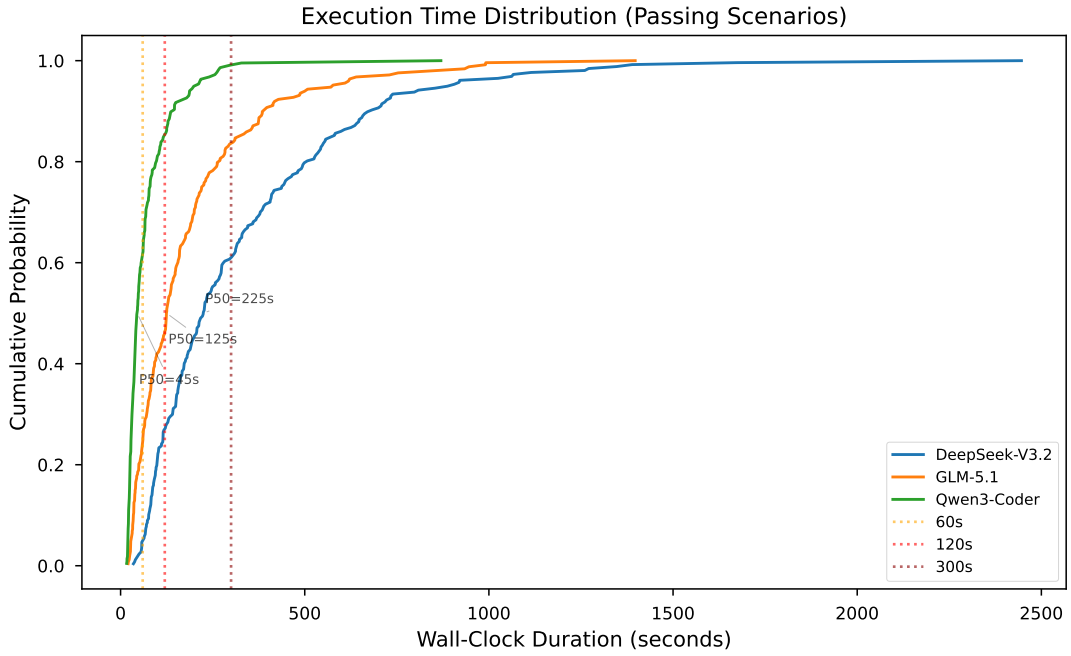


Figure 4.13: Cumulative distribution function of execution time across all configurations.

yielded the lowest (0.449). Among retrieval families, lookup scenarios achieved the highest pass rate (95.0%) with an NDCG@10 of 0.762, while search scenarios achieved the lowest pass rate (54.0%) with an NDCG@10 of 0.325. Comparison scenarios recorded the highest NDCG@10 (0.838) despite a moderate pass rate (66.7%).

Table 4.10: Pass rate and mean NDCG@10 disaggregated by stratification variable across all configurations.

Variable	Value	n	Pass rate	NDCG@10
Complexity	Multi-hop	360	65.6%	0.449
	Reasoning	288	93.4%	0.458
	Single-hop	288	75.3%	0.525
Retrieval family	Comparison	72	66.7%	0.838
	Impact	144	66.7%	0.246
	Lookup	180	95.0%	0.762
	Search	252	54.0%	0.325
	Traceability	288	94.1%	0.391

Critical difference diagram. The three models were ranked by mean rank across scenarios (Figure 4.14), where a lower rank indicates a higher pass rate. DeepSeek-V3.2 achieved the lowest (best) mean rank, followed closely by GLM-5.1. Qwen3-Coder received the highest (worst) mean rank, consistent with its lower overall pass rates.

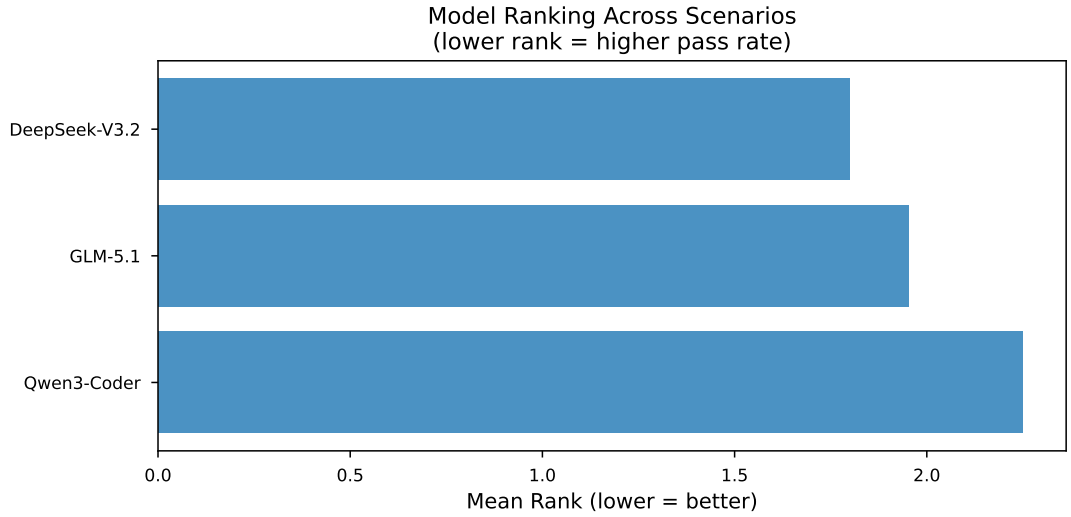


Figure 4.14: Mean rank of each model across scenarios (lower rank = higher pass rate).

Entity-coverage sensitivity. Because the entity-coverage threshold used in functional verification affects verdict classification, its sensitivity is examined here rather than within a single RQ. Table 4.11 and Figure 4.15 report sensitivity over 594 executions with extractable expected-entity lists. Pass rates declined gradually from 58.1% at the 50% threshold to 52.4% at the 80% operating point and 51.0% at the 100% threshold. The small change from 95% to 100% indicates that the strictest threshold did not create a floor effect in this subset. Between 55% and 70% the pass count was stable (318–324), indicating that few executions had entity-coverage scores concentrated in this band. The largest jump occurred between 50% and 55% ($\Delta = 21$ executions), suggesting a cluster of executions with approximately half their required entities present.

Table 4.11: Verdict distribution under varying entity-coverage thresholds for executions with extractable expected-entity lists ($n = 594$). Δ reports change in pass count relative to the 80% operating point.

Threshold	Pass	Fail	Pass rate	Δ vs 80%
50%	345	249	58.1%	+34
55%	324	270	54.5%	+13
60%	324	270	54.5%	+13
65%	324	270	54.5%	+13
70%	318	276	53.5%	+7
75%	318	276	53.5%	+7
80% ←	311	283	52.4%	—
85%	304	290	51.2%	−7
90%	304	290	51.2%	−7
95%	304	290	51.2%	−7
100%	303	291	51.0%	−8

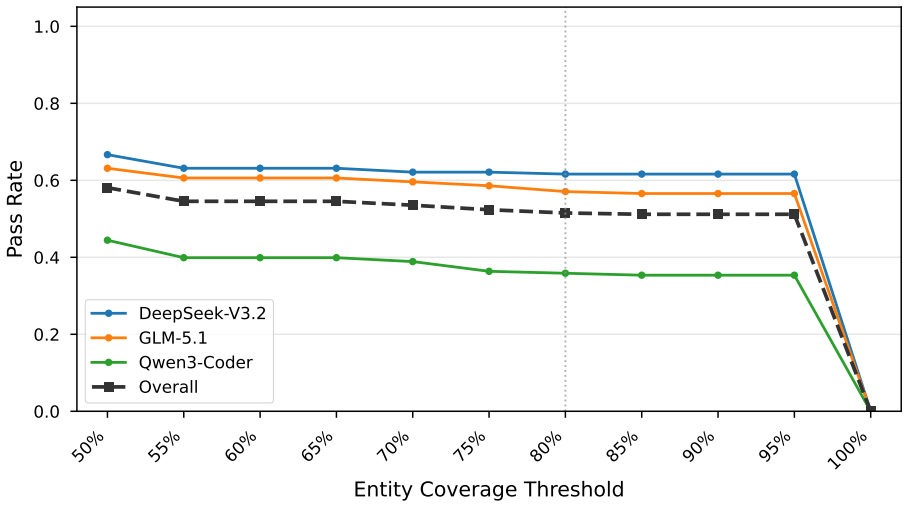


Figure 4.15: Pass rate as a function of entity-coverage threshold. Vertical marker at 80% indicates the operating point used throughout this study.

Summary. GLM-5.1 under Plan-and-Execute with Prompt augmentation achieved the highest pass rate (90.4%), while DeepSeek-V3.2 under Plan-and-Execute with Skill augmentation recorded the highest NDCG@10 (0.733). Knowledge-augmentation effects were model-dependent: DeepSeek-V3.2 gained 11.6 percentage points under Skill, whereas GLM-5.1 and Qwen3-Coder declined. Reasoning scenarios were the easiest (93.4%), search the hardest (54.0%). Intra-judge agreement was near-perfect ($\kappa = 0.844$, all $\alpha > 0.95$), and Bland–Altman analysis showed substantial inter-method divergence alongside near-zero intra-judge bias. Chapter 5 interprets these findings, with particular attention to the model-dependent knowledge-augmentation effects and the divergence between judge and IR metrics.

5 Discussion

This chapter interprets the empirical results from Chapter 4, situates them within the theoretical framework established in Chapter 2, and examines threats to validity, methodological limitations, and ethical considerations.

5.1 Results Discussion

RQ1: Scoring-protocol validity and judge–IR complementarity. *RQ1 asks how the benchmark and scoring protocol are validated, and how LLM-as-a-judge scores relate to traditional IR metrics.*

Before the judge can be compared against external metrics, its internal consistency must be established. The calibration stage (Section 3.3) produced Cohen's $\kappa = 0.844$ on binary verdict, classified as *almost perfect* agreement under the Landis and Koch scale [94]. Pearson $r = 0.986$ on the composite score falls into the *very strong* band of Evans' correlation guidelines [95]. Per-dimension Krippendorff α exceeded 0.95 for all four quality facets, well above the 0.800 threshold that Krippendorff identifies as sufficient for drawing reliable conclusions [87]. The Bland–Altman intra-judge panel (Figure 4.3(b)) corroborates these statistics: mean difference between primary and calibration rounds is near zero, with no proportional bias. Taken together, these results establish that the judge is a *reliable* measurement instrument in the test–retest sense: its outputs are stable enough across repeated scoring of identical inputs for comparisons against external baselines to be meaningful. Reliability of this kind is a precondition for validity, not a demonstration of it [87]; whether judge scores track expert assessment remains unmeasured (Section 5.4). The binary-verdict $\kappa = 0.844$ also implies that a small fraction of verdicts—on the order of 5–7%, depending on the marginal pass rate—flip between identical re-judgings, which sets a noise floor when interpreting small cross-configuration pass-rate differences.

With internal consistency established, the cross-family Spearman correlations ($\rho = 0.27$–0.53, all $p < 0.001$; Table 4.4) warrant interpretation. The moderate magnitude reflects a structural difference in what each metric family captures rather than a deficiency in either. IR metrics assess rank-aware retrieval quality: whether relevant artifacts are retrieved and placed highly. The judge assesses response-level quality: whether the agent's final answer is complete, coherent, and grounded in retrieved evidence. These are related but distinct constructs. The gradient across judge dimensions supports this reading: Relevance, the dimension most conceptually aligned with retrieval, achieves the highest correlation ($\rho = 0.526$ against

NDCG@10), while Coherence and Evidence Grounding, which measure synthesis rather than retrieval, correlate less strongly. If the judge merely recapitulated IR scoring, all dimensions would correlate equally; if it were orthogonal, none would. The observed gradient indicates shared but not redundant quality signals.

The inter-method Bland–Altman panel (Figure 4.3(a)) adds a distributional perspective: the judge is systematically more lenient than IR metrics at low scores and converges at high scores, producing a significant negative proportional-bias slope. This pattern is consistent with the judge assigning partial credit for response coherence and presentation quality even when retrieval ranking is poor; this is credit that rank-based IR metrics cannot assign.

These results position the dual-metric protocol as a complementary evaluation design: each family surfaces quality aspects invisible to the other. A retrieval-focused judge that replicated IR scoring was a feasible alternative but was deprioritized in favor of dimensions that capture response quality beyond retrieval rank. The correlation coefficients reported here provide one reference point for future evaluations that adopt LLM-based judges in retrieval-augmented settings. They sit alongside an established LLM-as-relevance-judge literature in IR evaluation [60, 61, 62] and automated RAG-judge frameworks [63, 64]; those lines of work validate LLM judges as relevance assessors or RAG scorers, whereas the comparison here correlates a response-level judge with deterministic IR metrics over agentic executions. No directly comparable *agentic* retrieval benchmark reporting the same judge-versus-IR comparison was identified in the surveyed literature, but the claim should not be read as an exhaustive bibliographic assertion.

RQ2: Knowledge-augmentation effects are model-dependent. *RQ2 asks how knowledge augmentation affects correctness, retrieval quality, and grounding, and how workflow choice moderates those effects.*

The central finding for RQ2 is that the mode of domain-knowledge augmentation is not universally beneficial: its effect on task correctness is moderated by model characteristics, reversing direction across models. Under Plan-and-Execute, DeepSeek-V3.2 improved from 76.9% (Prompt) to 88.5% (Skill), a gain of 11.6 percentage points, while GLM-5.1 showed the opposite pattern, declining from 90.4% (Prompt) to 82.7% (Skill), a loss of 7.7 percentage points. Qwen3-Coder exhibited a smaller decline (71.2% to 67.3%). These opposing effects appeared despite all three knowledge configurations delivering equivalent domain knowledge content through different delivery mechanisms.

Two complementary mechanisms explain the divergent responses. First, a *context-pressure mechanism*: the Prompt configuration embeds approximately 2,700 tokens of domain knowledge into the system prompt, re-transmitted on every LLM invocation. For DeepSeek-V3.2 (128 K served context window), this overhead represents a proportionally larger fraction of the available context budget than for GLM-5.1 (197 K) or Qwen3-Coder (205 K served; Table 3.7). Under Plan-and-Execute, where the orchestrator spawns multiple sub-agents each receiving the system prompt, this cost compounds. Moving domain knowledge to on-demand skill loading under the Skill configuration relieves this pressure, freeing context capacity for retrieval results and reasoning traces. This mechanism predicts that models with smaller context windows benefit more from on-demand loading, consistent with the observed +11.6 percentage point gain for DeepSeek-V3.2.

Second, an *autonomous skill-selection mechanism*: on-demand skill loading requires the model to decide *when* to load *which* skill, a meta-cognitive step absent under Prompt, where all knowledge is immediately available. The efficiency data (Table 4.9) reveals that GLM-5.1 under Skill issued 47 median tool calls, compared with 36 under Prompt, a 30% increase that coincided with a 7.7 percentage point *decline* in pass rate. This pattern is consistent with GLM-5.1 thrashing: issuing more retrieval and skill-loading calls without effectively leveraging the loaded knowledge. The Skill+Meta configuration, which adds explicit guidance on skill-selection order and usage heuristics, partially recovered GLM-5.1's performance to 88.5%

while reducing tool calls to 30, fewer than under Prompt. The meta-prompting layer appears to compensate for GLM-5.1's weaker autonomous skill-selection capability by providing an external decision scaffold.

DeepSeek-V3.2 exhibited the inverse Skill+Meta pattern: performance dropped from 88.5% (Skill) to 82.7% (Skill+Meta). Where GLM-5.1 benefited from the meta-prompting scaffold, DeepSeek-V3.2 was constrained by it. The additional guidance may have restricted reasoning strategies that DeepSeek-V3.2 would otherwise have selected autonomously, reducing the flexibility that made the Skill configuration effective.

Qwen3-Coder's uniformly low performance (65.4–73.1%) across all augmentation modes reflects a different bottleneck. With a median of 14 tool calls under Plan-and-Execute Prompt (compared with 46 for DeepSeek-V3.2 and 36 for GLM-5.1) and output token counts roughly one-third of the other models (Table 4.9), Qwen3-Coder did not engage deeply enough with the retrieval task for the knowledge-delivery mechanism to make a material difference. Its 7.4% tool-call error rate under Plan-and-Execute (Table 4.7), the highest among all models, further limited effective retrieval depth. Knowledge augmentation, whether inline, on-demand, or guided, cannot compensate for insufficient retrieval engagement.

RQ2: Orchestration strategy as moderator. Plan-and-Execute configurations achieved higher aggregate functional pass rates than ReAct for GLM-5.1, reached near-parity for DeepSeek-V3.2, and did not improve Qwen3-Coder (Figure 4.4). GLM-5.1 exhibited the largest gap (87% Plan-and-Execute vs. 72% ReAct, aggregated across augmentation modes; Figure 4.5), while DeepSeek-V3.2 showed near-parity (83% vs. 82%) and Qwen3-Coder was lower under Plan-and-Execute than under ReAct (68% vs. 71%).

The orchestrator-worker pattern provides three structural advantages: explicit task decomposition by the orchestrator, isolated context windows per sub-task preventing cross-contamination, and deeper exploration through more tool calls. These advantages are most valuable when the model cannot autonomously decompose multi-step queries within a single-agent loop. DeepSeek-V3.2's near-parity between workflows suggests that it already performs effective multi-step reasoning within a single context window, making the orchestrator's decomposition redundant. The added latency and token cost of Plan-and-Execute (Table 4.9) then represents overhead without proportional benefit.

However, the GLM-5.1 comparison requires a caveat. GLM-5.1 under ReAct exhibited a systematic failure mode in which specific model–query–workflow combinations produced empty final responses despite successful intermediate tool calls. Post-hoc analysis of all 156 GLM-5.1 ReAct extractions found that 22 executions (14.1%) terminated on an invalid tool call with no final response text; all 22 ended on a graph-traversal tool (11 on *graph_traverse*, 9 on *graph_blast_radius*, 2 on *graph_trace_coverage*), and none of the 156 GLM-5.1 Plan-and-Execute executions or any DeepSeek-V3.2 or Qwen3-Coder ReAct execution exhibited this pattern. The failure rate was consistent across knowledge configurations (baseline 11.5%, Skill 17.3%, Skill+Meta 13.5%), affecting 15 of 52 unique scenarios. Because the judge evaluates the final response text, these empty responses were correctly scored as failures, inflating the observed ReAct failure rate for GLM-5.1 by 14.1 percentage points in the worst case. The true Plan-and-Execute advantage for GLM-5.1 is therefore bounded above by the observed 15 percentage points; the actual advantage attributable to orchestration-strategy differences, rather than this runtime interaction, cannot be isolated from the present data. A post-hoc sensitivity analysis excluding the 22 affected executions from both workflows reduced the gap from 15.4 to 8.2 percentage points (134 paired executions), suggesting that a residual orchestration-strategy advantage persists independent of this failure mode.

The absence of this failure mode in Plan-and-Execute results does not necessarily mean it does not occur at the sub-agent level. Under Plan-and-Execute, if an executor sub-agent produces an empty response, the orchestrator can treat this as an incomplete sub-task. It can then replan or delegate to a fresh executor, masking the failure through retry. This self-healing

property of the orchestrator-worker topology may contribute to Plan-and-Execute's higher observed pass rate for GLM-5.1, independent of the task-decomposition advantages discussed above.

The efficiency data offers indirect support. Moving from Prompt to Skill increased GLM-5.1's median tool-call count by 11, from 36 to 47. The corresponding increases were 2 for DeepSeek-V3.2 and 5 for Qwen3-Coder (Table 4.9). The disproportionate increase for GLM-5.1 is consistent with additional orchestrator delegation calls to replacement sub-agents following empty responses from failed executors. These short interactions add to the call count without proportional token overhead.

For Qwen3-Coder, the orchestrator-worker pattern did not yield meaningful improvement. Although the orchestrator decomposed queries into sub-tasks, Qwen3-Coder's sub-agents executed each sub-task with the same shallow engagement observed under the single-agent loop (14 median tool calls under Plan-and-Execute vs. 4 under ReAct; Table 4.9). Structural scaffolding helps task decomposition but cannot increase retrieval depth when the underlying model does not pursue iterative evidence gathering.

RQ3: Model characteristics and operational reliability. *RQ3 asks how model characteristics affect reliable tool use and evidence-grounded retrieval performance.*

All three evaluated models share a Mixture-of-Experts architecture with comparable active-parameter counts (35–40 B per token; Table 3.7), yet their tool-call success rates diverged sharply. DeepSeek-V3.2 achieved 100.0% under Plan-and-Execute and 99.9% under ReAct, GLM-5.1 recorded 97.6% and 96.4%, and Qwen3-Coder fell to 92.6% and 86.0% (Table 4.7). Because architecture family and active scale are not varied independently, the present design cannot estimate their causal effect. The observed gradient is instead consistent with the post-training dimensions identified in Section 2.4: reinforcement-learning pipeline structure, tool-call serialization format, and runtime reasoning mode.

The published technical reports for each model offer plausible, though post-hoc, explanations for these behavioral profiles. The study design cannot isolate training-pipeline effects from other confounds such as serving infrastructure and tokenizer differences. DeepSeek-V3.2's near-perfect reliability coincides with the highest token consumption across all configurations (Table 4.9). Its single-stage reinforcement-learning recipe jointly optimizes reasoning and tool use with a relaxed length penalty. Its preservation of chain-of-thought traces across consecutive tool calls also produces a policy that explores exhaustively before committing to an answer [73]. Case Study 3 (Section 5.2) shows how this exhaustive exploration becomes counterproductive on unanswerable queries.

GLM-5.1's sequential three-stage pipeline (reasoning, agentic, general) with on-policy cross-stage distillation yields strong schema compliance when the task matches its agentic-RL training distribution, such as verifiable software-engineering and search environments. However, the distribution seams between stages produce thrashing when the model must autonomously select among skills outside that distribution [74]. Qwen3-Coder operates as a non-thinking instruct variant without chain-of-thought before tool calls, and its agent reinforcement learning was concentrated on coding execution environments. This combination is consistent with the observed pattern of shallow engagement and high error rates. The custom XML tool-call grammar may also introduce additional parsing fragility in third-party frameworks [75].

Token consumption and task accuracy correlate positively across models, but aggregate data cannot disentangle whether thoroughness drives accuracy or whether both reflect training quality. The grounding analysis partially supports a causal link: higher grounding coverage co-occurred with fewer hallucinated identifiers across all three models (Figure 4.9), and the decaying trend held consistently regardless of model. Configurations that consumed more tokens on retrieval produced responses with stronger evidence grounding and fewer hallucinated identifiers. A common-cause explanation rooted in training quality remains

possible, but the grounding-hallucination gradient is at minimum consistent with evidence-first answering as a reliability mechanism.

Abstention accuracy presents a second reliability concern. Across all 18 configurations, missed-abstention rates ranged from 42.9% to 71.4% on the 14 unanswerable scenarios (Table 4.8). No model or configuration achieved acceptable abstention accuracy, consistent with recent findings that reasoning-capable LLMs systematically fail on unanswerable questions [93]. One might expect runtime reasoning mode to matter here, since models that suppress chain-of-thought lack a deliberative step in which evidence sufficiency could be assessed. However, DeepSeek-V3.2 and GLM-5.1, which retain active chain-of-thought, exhibited similarly high missed-abstention rates. The more likely explanation is the training signal: all three models were optimized for task completion rather than selective prediction, and none received explicit reward for withholding answers on unanswerable inputs. The deterministic runtime failure observed for GLM-5.1 under ReAct, discussed in the orchestration-strategy analysis above, further illustrates that operational reliability depends on the interaction between model and execution environment, not on either alone.

Cross-Cutting Observations. Beyond model-level differences, scenario characteristics also modulate observed performance. Pass rates stratified by scenario complexity (Table 4.10) show an unexpected ordering: reasoning scenarios achieved the highest pass rate (93.4%), followed by single-hop (75.3%) and multi-hop (65.6%). This ordering likely reflects the structure of ground-truth annotations rather than intrinsic task difficulty. Reasoning scenarios require the agent to synthesize or infer from retrieved evidence, but they typically involve fewer expected entities in the ground truth. With fewer entities required, the 80% entity-coverage threshold is easier to satisfy. Multi-hop scenarios, which require cross-artifact evidence assembly over larger sets of expected entities, impose a stricter effective bar despite the same nominal threshold. Whether this fully accounts for the ordering or whether reasoning queries are also more self-contained (less dependent on cross-artifact retrieval) cannot be determined from aggregate pass rates alone.

Retrieval-family pass rates (Figure 4.11) followed a gradient that matches task structure. Lookup scenarios, which require resolution of a single entity or identifier, achieved the highest rate (95.0%), closely followed by traceability (94.1%). Search scenarios were the most challenging (54.0%), with comparison and impact scenarios at 66.7% each. Models diverged most on search and comparison families, where the spread between the best- and worst-performing models exceeded 20 percentage points. Search scenarios are difficult because they are open-ended and multi-step: unlike lookup, which terminates after a single retrieval, search requires iterative query refinement and evidence aggregation across multiple tool calls.

This per-family profile is inverted relative to the motivating use case. The families closest to real change-scoping work—search (54.0%) and impact analysis (66.7%)—are precisely where performance is weakest, while the strongest families (lookup 95.0%, traceability 94.1%) are those where deterministic queries over recorded links are already viable. Deployment guidance therefore cannot be read off aggregate pass rates: under supervision, lookup- and traceability-style scoping is near-deployable, whereas search and impact analysis—the core of the semantic-gap motivation—remain research-grade.

The entity-coverage sensitivity analysis (Table 4.11; Figure 4.15) supports the choice of an 80% operating point. Pass rates remained stable between the 55% and 70% thresholds, indicating that few executions had entity-coverage scores falling within this band. The pass rate changed only slightly from approximately 51.2% at the 95% threshold to 51.0% at 100%, so this subset did not exhibit a floor effect at exact match. This result validates the decision to report threshold sensitivity rather than treating the 80% operating point as self-evident.

5.2 Qualitative Case Studies

This section presents three executions selected to expose behavioral patterns that aggregate metrics cannot reveal: error recovery through tool diversity, task-decomposition failure despite competent retrieval, and partial abstention driven by training incentives. Each case study traces a single execution from query through tool-call sequence to judge verdict. The three cases progress from a successful lookup (composite score 0.95) through a retrieval that failed at the traceability step (0.56) to a correct abstention undermined by excessive elaboration (0.88).

Case 1: Successful lookup with tool-error recovery. A lookup scenario queried details and traceability links for a single test-case identifier (hereafter TC-A). The ground truth specified one expected identifier and no expected entities; the correct answer was that TC-A exists but has no linked backlog requirements. This scenario tests whether the agent can report absence of evidence without fabricating links.

Under Plan-and-Execute with the Prompt configuration, Qwen3-Coder's orchestrator delegated to an executor sub-agent that issued six tool calls. The first two, a dedicated identifier lookup and a trace-link lookup, both failed due to malformed parameters. This matches the 7.4% tool-call error rate observed for this model (Table 4.7), the highest among the three evaluated models. Rather than terminating, the agent fell back to general-purpose retrieval: a Cypher graph query matching TC-A to its backlog references returned no rows, correctly indicating no backlog references exist in the knowledge graph. A constrained SQL query confirmed that TC-A exists with its name and suite membership. A graph traversal for coverage analysis returned six protocol interfaces covered by TC-A but listed a single backlog entry with all null fields: no node identifier, no name, no label. This null-field entry is a data artifact representing an edge in the graph with no target node, not a genuine traceability link. The agent correctly interpreted it as absence rather than presence and did not report a nonexistent link.

The fallback from specialized to general-purpose tools shows why a diverse tool registry matters: redundant retrieval paths allow recovery when individual tools fail. The null-field backlog entry was a potential fabrication trap; a naive agent might have reported it as a linked requirement. The judge assigned identifier coverage 1.0, entity coverage 1.0 (vacuously satisfied), reference alignment 0.85, and quality scores of completeness 4, relevance 4, coherence 5, and evidence grounding 5, yielding a composite score of **0.95** with a pass verdict. The two non-maximum quality scores reflected omitted metadata (quality area and execution context) rather than factual error. The judge noted: *"correctly stated no backlog requirements are linked. Did not fabricate links."* This case connects the tool-reliability gradient from RQ3 (Table 4.7) to a concrete recovery mechanism: the model with the highest error rate still arrived at the correct answer because the tool registry offered alternative retrieval paths.

Case 2: Retrieval success, traceability failure. A search scenario asked the agent to find test cases covering a specific protocol interaction during a particular handover phase and to identify their associated backlog reference entities. The ground truth specified two expected test-case identifiers and three expected backlog entities (two requirements and one feature reference). This scenario tests both retrieval (finding the right test cases) and traceability (following their requirement linkages).

Under ReAct with the Prompt configuration, GLM-5.1 issued seven tool calls. It began with two knowledge-base terminology resolution calls for the relevant protocol and handover terms; both returned no matches. This is expected behavior: the knowledge base indexes domain-specific acronyms and abbreviations, not standard protocol procedure names. A no-match result here is not a retrieval failure but a correct signal that the terms do not require disambiguation. The agent then executed two dense vector searches and two sparse keyword

searches, returning semantically and lexically relevant test-case chunks, followed by one additional terminology resolution call.

The agent produced a well-structured response: a table categorizing five relevant test cases by interaction type and scope, correctly identifying both expected test-case identifiers. The response earned coherence 5/5, relevance 4/5, and evidence grounding 4/5. All seven tool calls targeted relevant data sources, and the retrieval results were accurate.

The failure occurred at the traceability step. No graph traversal, trace-link lookup, or any traceability-oriented tool call appeared in the sequence. The agent treated the task as "find test cases" rather than "find test cases and their associated requirements." The three expected backlog entities were never queried for, yielding 0% entity coverage and a fail verdict at overall score **0.56**. The calibration re-judgment produced a near-identical result (overall 0.54, delta 0.02). The meta-judge confirmed both judgments as equally accurate: *"Both judgments correctly identify the IDs as found, all three expected entities as missing, and correctly assign fail status. The overall score difference of 0.02 is negligible."*

This execution was one of GLM-5.1's ReAct successes: competent retrieval, well-grounded evidence, and a coherent response. The failure is not in retrieval capability but in task decomposition. Under ReAct, GLM-5.1 must self-direct its tool-call chain, and it did not reason about what additional retrieval steps were needed to resolve the backlog entities; had it taken one additional step to trace backlog linkages, the composite score could have reached 0.8–1.0 rather than 0.56.

The same scenario executed with the same model and knowledge configuration under Plan-and-Execute confirms this interpretation. GLM-5.1 under Plan-and-Execute scored **0.98** on this query, with all three expected backlog entities found (3/3 entity coverage), reference alignment 0.95, and quality scores of completeness 5, relevance 4, coherence 5, and evidence grounding 5. The orchestrator decomposed the query into separate retrieval and traceability sub-tasks, and the executor sub-agents traced the backlog references that the ReAct agent never queried for. The contrast between 0.56 (ReAct) and 0.98 (Plan-and-Execute) on identical input isolates the effect of agentic workflow: the model's retrieval and traceability capabilities were sufficient in both cases, but only the orchestrator-worker topology ensured that both were exercised. This provides concrete evidence for the structural advantage of Plan-and-Execute discussed in Section 5.1: explicit task decomposition compensates for the model's inability to autonomously recognize that its retrieval was incomplete.

Case 3: Correct abstention with excessive elaboration. An impact scenario asked the agent to identify test cases affected by a specific requirement identifier. The identifier was fabricated and did not exist in the knowledge base; the ground truth specified empty expected-identifier and expected-entity lists. The correct behavior was to recognize the nonexistence and clearly state that no impact assessment is possible.

Under Plan-and-Execute with the Skill configuration, DeepSeek-V3.2's orchestrator decomposed the task into four sub-tasks and delegated to executor sub-agents that collectively issued 60 tool calls. The first phase confirmed nonexistence through four independent retrieval methods: sparse keyword search, dense vector search, Cypher graph queries, and direct SQL lookups. All returned no matches for the fabricated identifier. At this point, the agent had sufficient evidence to report the finding.

Rather than stopping, the orchestrator's third sub-task instructed an executor to search for similar identifiers as a fallback. This triggered a second phase: pattern-based queries for requirements sharing a similar identifier prefix, which returned 45 main requirements and 95 solution packages with 1,755 associated test cases. The agent then produced two response documents totaling over 1,500 words. These comprised a coverage audit and an impact analysis examining the broader requirement pattern's test coverage distribution (89% traffic functionality, 10% system operation), automation gaps (0% automated test cases), and

system-level breakdown. This analysis was factually grounded in tool results but entirely tangential to the original query, which asked about a specific nonexistent requirement.

The agent correctly concluded that the requirement does not exist and suggested verifying the identifier, earning a pass verdict at **0.88**. However, the tangential analysis reduced relevance to 3/5; the judge noted: *"much of the response analyzes the broader pattern which is tangential to the specific nonexistent requirement."* Reference alignment dropped to 0.6 because the reference answer recommended suggesting specific nearby valid identifiers, which the agent replaced with an unsolicited statistical survey. The calibration re-judgment agreed on status and alignment (overall 0.87, delta 0.01) but scored coherence at 4 rather than 5. The meta-judge ruled the difference not meaningful: *"The only difference is coherence scoring (A=5, B=4), which is within 1 point and both defensible. No bias detected."*

This case is a positive counterexample to the high missed-abstention rates (42.9–71.4%) reported in the model-characteristics analysis: the binary verdict was correct. It reveals, however, a subtler failure mode: partial abstention. The agent recognized the unanswerable condition within the first few tool calls yet continued generating content, sacrificing precision for volume. This behavior aligns with the training characteristics discussed in Section 5.1: the model's reinforcement-learning recipe with a relaxed length penalty produces a policy that explores exhaustively before committing to an answer. This improves thoroughness on answerable queries; on unanswerable ones, it wastes computation. The 60 tool calls and 1,500 words of tangential analysis reduced the relevance and precision scores without improving the pass verdict.

Taken together, the three cases expose a common theme: the gap between what the agent retrieves and what it reports. In Case 1, the agent correctly interprets ambiguous evidence (null-field graph entries) as absence. In Case 2, it retrieves the right artifacts but omits a traceability step that would have surfaced the expected entities. In Case 3, it identifies the correct answer early but buries it under tangential analysis. Each case also validates a different stage of the evaluation pipeline: Case 1 shows the composite scoring protocol distinguishing factual correctness from presentation quality; Case 2 confirms calibration stability through near-identical primary and calibration verdicts; Case 3 shows the meta-judge resolving a coherence disagreement without overriding a correct pass verdict.

5.3 Threats to Validity

Scope of the comparison. This thesis does not empirically compare hybrid retrieval against single-modality alternatives (dense-only, sparse-only, graph-only), nor agentic pipelines against non-agentic retrieve-then-generate baselines. Both are scope premises (Section 1.3). Prior work motivates hybrid retrieval and agentic orchestration for heterogeneous, multi-step retrieval settings [25, 24, 29, 32, 35, 36]. The scenario taxonomy used here adopts that premise by stratifying queries across structural traversal, reverse-edge enumeration, set difference, and identifier-grounded lookup. An ablation against single-modality or non-agentic baselines would be useful for quantifying the premise on this exact benchmark, but it was outside the contribution boundary. The empirical study instead isolates effects *within* this scope: workflow $\times$ knowledge-mode $\times$ model interactions. The boundary between the adopted DeepAgents harness foundation and the retrieval-domain extensions contributed by this thesis is summarized in Table 3.3.

Construct Validity. The dual-metric evaluation protocol mitigates single-metric bias by comparing LLM-as-a-judge composite scores against traditional IR metrics on the same benchmark executions. However, three researcher-defined parameters are especially consequential. The entity-coverage threshold (80%) determines how many expected entities must appear in the response for a pass verdict. The overall score weighting (30/30/20/20 across identifier coverage, entity coverage, reference alignment, and quality average) prioritizes deterministic

verification over subjective quality assessment. The low-alignment threshold (0.4) categorizes failure modes in post-hoc analysis. Of these, only the entity-coverage threshold was examined through sensitivity analysis (Section 4.5; Table 4.11), which reports pass rates across the full threshold range. The composite weighting and the alignment threshold are design choices informed by domain conventions; their effect on configuration rankings was not tested, and alternative choices would shift absolute pass rates.

A second construct threat concerns entity coverage itself and is unmitigated by the sensitivity analysis. The judge evaluates entities as they appear in the agent's final response, but an agent may retrieve the correct artifacts via tool calls and then fail to surface specific entities in its written answer. The judge-facing scoring protocol does not itself distinguish retrieval failure (the entity was never found) from communication failure (the entity was found but omitted from the response); both are penalized equally under entity coverage. A post-hoc cross-reference of tool-call results against verdict entity lists quantifies the split: of 902 missing expected entities across all executions, 640 (71%) never appeared in any tool result, while 262 (29%) were present in retrieved evidence but omitted from the final response, with the communication-failure share ranging from 24% (GLM-5.1) to 32% (Qwen3-Coder). Most entity-coverage failures are therefore genuine retrieval failures, but roughly a third reflect response synthesis rather than retrieval capability.

A third construct threat concerns the deterministic grounding metric. First, its evidence pool is built from tool-call records recovered from conversation checkpoints (Section 3.3); because every retrieval result was offloaded to the virtual filesystem with only a truncated preview returned in context (Section 3.2.1), identifiers present only in the full offloaded artifact may be absent from this pool. Under that construction, the measured grounding coverage is a conservative lower bound rather than a point estimate. Second, the absolute level warrants confrontation: mean grounding coverage for Plan-and-Execute configurations was 0.384–0.504 (Table 4.7), meaning that roughly half of the response identifiers could not be matched to tool-backed evidence under this construction. Third, this level diverges from the judge's Evidence Grounding dimension (3.6–4.6 of 5 on the same Plan-and-Execute runs; Table 4.3), which scores the impression of evidence use rather than verifiable identifier provenance; the divergence is consistent with the two constructs measuring different things, with the deterministic metric being the stricter of the two. Finally, the metric operates on identifier-substring co-occurrence: it bounds the risk of fabricated identifiers but is blind to relation-level fabrication, in which genuine identifiers are connected by invented claims.

Internal Validity. Each configuration was executed once per scenario. The execution harness caught and retried infrastructure-level failures, but run-to-run variance in model output was not measured. LLM sampling introduces non-determinism: the same configuration and scenario could produce different tool-call sequences on a second execution. Observed percentage-point differences between configurations (e.g., the 11.6 percentage point gain for DeepSeek-V3.2 from Prompt to Skill) may therefore fall within the range of stochastic variation. No per-contrast confidence intervals or hypothesis tests were computed for these comparisons (the bootstrap procedure was scoped to the judge-agreement statistics), so per-configuration effect estimates should be interpreted as directional observations.

GLM-5.1 under ReAct exhibited a systematic failure mode: 22 of 156 executions (14.1%) terminated on an invalid graph-tool call with no final response text (Section 5.1). This behavior was reproducible, model-specific, and workflow-specific; it was absent from all GLM-5.1 Plan-and-Execute executions and from all DeepSeek-V3.2 and Qwen3-Coder ReAct executions. Because the root cause could not be isolated to the model, the serving infrastructure, or their interaction, all GLM-5.1 ReAct metrics should be interpreted as lower bounds on true capability. The observed Plan-and-Execute advantage for GLM-5.1 is bounded above by the reported 15 percentage points.

Extraction fidelity varies by model and workflow. Plan-and-Execute sub-agents produce structured artifact files from which ranked retrieval lists can be recovered, enabling grounding-coverage computation. ReAct accumulates retrieved content within a single context window without producing an explicit ranked list; consequently, grounding metrics for ReAct configurations reflect a structural absence of data rather than a measured value of zero. Cross-workflow grounding comparisons are therefore not possible.

External Validity. The benchmark was developed and executed within a single industrial partner organization, using proprietary test artifacts from one product domain (telecommunications). The 52 evaluation scenarios, while stratified across five retrieval families and three complexity levels, represent one domain's artifact structure, terminology, and traceability conventions. Generalization to other industrial domains, artifact types, or organizational contexts requires independent replication.

All three evaluated models were served through the same internal AI provider. Provider-level implementation choices (context handling, tool-call parsing, response-format enforcement) are confounded with model characteristics. The GLM-5.1 ReAct runtime failure may reflect a provider-specific serving interaction rather than an inherent model limitation. Whether these results generalize to other serving infrastructure for the same model weights is unknown.

Conclusion Validity. Bootstrap confidence intervals and Holm–Bonferroni correction were applied to the judge-agreement statistics (Section 3.3); the configuration-level comparisons in Chapter 4 are reported descriptively, without per-contrast hypothesis tests. At $n = 52$ scenarios per configuration, a binomial 95% confidence interval on a pass rate spans up to roughly ± 13 percentage points, so single-cell percentage-point differences should be read as directional. The topic-set size itself is consistent with TREC ad hoc practice [33] (Section 3.3). However, disaggregated analyses (per retrieval family, per complexity level) operate on substantially smaller cell sizes; the comparison family, for example, contains only 4 scenarios. Effect estimates at this granularity should be treated as directional rather than definitive.

5.4 Limitations and Method Discussion

No human judge baseline was collected. The reason was practical rather than conceptual: the project timeline and access to qualified domain experts did not allow a representative human-rating study. The LLM-as-a-judge was validated internally through calibration (intra-judge consistency) and externally against IR metrics (cross-family correlation), but it was never compared against ratings from human domain experts. How well judge scores approximate expert assessment remains unknown. Collecting human ratings on a representative subset of executions would establish judge-human agreement and is a priority for future work.

The entity-coverage metric conflates retrieval and communication failures at scoring time; the post-hoc cross-reference reported under construct validity (Section 5.3) attributes roughly 71% of missing entities to retrieval failure and 29% to response-synthesis omission. Integrating this decomposition into the scoring pipeline itself, rather than as a post-hoc analysis, would enable targeted improvements to either retrieval or response generation.

Relevance labels for IR scoring derive solely from each scenario's expected identifiers, so the evaluation operates with incomplete relevance judgments: artifacts retrieved outside the expected list count as non-relevant even when genuinely useful [91]. Per-configuration label coverage was computed and inspected during scoring but is not reported in the results tables; absolute IR-metric levels should therefore be compared within this benchmark rather than against externally pooled collections.

All three models were served through a single internal AI provider, so some behavioral differences ascribed to model training may instead stem from serving-infrastructure variation

that the present design cannot disentangle. Replicating the benchmark with an alternative serving stack for the same model weights would isolate this confound.

Grounding metrics are available only for Plan-and-Execute configurations because ReAct does not produce ranked artifact lists (Section 4.4). The grounding analysis is therefore restricted to one workflow family, and cross-workflow grounding comparisons remain infeasible. Logging explicit artifact lists within the ReAct loop would address this gap in future iterations.

The benchmark's ground truth is bounded by recorded knowledge. Expected identifiers and entities derive from trace links and structured fields recorded in the canonical database—the same links whose incompleteness motivates the thesis. The benchmark therefore measures recovery of *recorded* knowledge; discovery of relevant-but-unlinked test cases, the core semantic-gap use case, is structurally untested, since an agent surfacing a genuinely relevant but unrecorded test would be penalized rather than rewarded by identifier-based scoring. Reference answers carry a second, weaker layer of assurance: two-thirds were validated only by reviewer agents, and the one-third manual audit verified the system under test's responses against the stated ground truth, a procedure that involves the SUT itself and therefore carries a circularity risk.

5.5 Ethical and Societal Considerations

The benchmark scenarios reference real industrial test artifacts, including test-case identifiers, requirement descriptions, and procedural specifications. All scenario content was anonymized prior to evaluation: the authors replaced internal entity type names with generic terms and excluded personally identifiable information from the benchmark data. The proprietary nature of the underlying artifacts prevents public release of the benchmark dataset.

The evaluation protocol relies on an LLM-as-a-judge to assess response quality, replacing human expert judgment with automated scoring. Calibration and meta-judge verification reduce systematic scoring errors, but adopting automated evaluation in practice would reduce direct human oversight. The high missed-abstention rates observed across all configurations (Section 5.1) illustrate that the agent-judge system can fail to recognize its own limitations. The judge is intended as a research evaluation instrument; its deployment as an unsupervised quality gate would require additional validation, including comparison against human expert ratings.

6 Conclusion

This thesis designed, implemented, and evaluated a retrieval-domain extension of the DeepAgents agent harness [37, 35, 36] for hybrid retrieval over heterogeneous industrial test-case artifacts in a telecommunications setting. The evaluation comprised 936 benchmark executions across 18 configurations (three large language models, two orchestration strategies, and three knowledge-augmentation modes) on 52 evaluation scenarios, each targeting a distinct retrieval task over a corpus of 11,366 industrial test cases, stratified by retrieval family and complexity level. A three-tier evaluation pipeline (primary judge, calibration, meta-judge arbitration) with a composite scoring protocol assessed task correctness alongside traditional IR metrics. The following sections answer the three research questions, discuss implications, and outline future work.

6.1 Answering the Research Questions

RQ1: Scoring-protocol validity and judge–IR complementarity. The three-tier evaluation pipeline produced a stable and internally consistent measurement instrument. Intra-judge agreement reached Cohen's $\kappa = 0.844$ on binary verdict, classified as almost perfect under the Landis and Koch scale [94]. Pearson $r = 0.986$ on composite score and per-dimension Krippendorff α above 0.95 for all four quality facets [87] further confirmed internal consistency. Cross-family Spearman correlations between judge dimensions and IR metrics fell in the range $\rho = 0.27$–0.53 (all $p < 0.001$), indicating shared but not redundant quality signals. The judge assesses response-level synthesis quality, such as coherence and evidence grounding, which rank-based IR metrics miss. Conversely, IR metrics reveal retrieval ranking quality that the judge does not directly measure. The dual-metric protocol is therefore complementary, and single-metric evaluation would miss quality aspects visible only to the other family. The principal limitation is the absence of a human-expert baseline; how well judge scores approximate domain-expert assessment remains unknown.

RQ2: Agent-skill configuration, agentic workflow, and model capability. The effect of knowledge-augmentation mode on task correctness was model-dependent. Under Plan-and-Execute, DeepSeek-V3.2 gained 11.6 percentage points moving from Prompt to Skill augmentation, while GLM-5.1 lost 7.7 percentage points on the same transition. These single-run

contrasts at $n = 52$ are directional observations, consistent with two non-exclusive candidate mechanisms—context-budget relief and the added burden of autonomous skill selection—that cannot be separated with one model per behavioral profile. The Skill+Meta configuration, which adds explicit skill-selection guidance, partially recovered GLM-5.1's performance to 88.5% while reducing tool calls below the Prompt baseline. DeepSeek-V3.2 showed the inverse pattern: Skill+Meta dropped performance from 88.5% to 82.7%, consistent with the meta-prompting scaffold constraining reasoning strategies the model would have selected on its own. In this three-model sample, meta-prompting thus compensated the model that struggled with autonomous skill selection and constrained the model that did not.

Holding the agentic architecture fixed per Section 1.3, the workflow comparison contrasted ReAct against orchestrator-worker Plan-and-Execute. The orchestrator-worker pattern improved aggregate functional pass rate for GLM-5.1, reached near-parity for DeepSeek-V3.2, and did not improve Qwen3-Coder. GLM-5.1 exhibited the largest observed gap (87% vs. 72%), though this difference must be interpreted cautiously.

A systematic empty-response failure mode affected 22 of 156 GLM-5.1 ReAct executions (14.1%), all terminating on an invalid tool call with no final response text. Because these empty responses were correctly scored as failures, they inflated GLM-5.1's observed ReAct failure rate. A sensitivity analysis excluding the 22 affected executions from both workflows reduced the gap from 15.4 to 8.2 percentage points.

DeepSeek-V3.2 achieved near-parity between workflows (83% vs. 82%), suggesting that it already performs effective multi-step reasoning within a single context window. Qwen3-Coder was lower under Plan-and-Execute than under ReAct (68% vs. 71%) and exhibited uniformly low performance (65–73%) across augmentation modes and workflows. Combined with its shallow tool engagement (14 median tool calls vs. 36–46 for the other models), this pattern is consistent with runtime reasoning capability being necessary for effective agentic retrieval in this domain. Neither structural scaffolding nor knowledge-delivery mechanisms substituted for a model that did not pursue iterative evidence gathering.

RQ3: Model characteristics and operational reliability. Within the all-MoE, roughly matched active-parameter sample, tool-use reliability varied substantially. Tool-call success rates ranged from 86.0% (Qwen3-Coder, ReAct) to 100.0% (DeepSeek-V3.2, Plan-and-Execute). This variation aligns with differences in post-training: reinforcement-learning pipeline structure, tool-call serialization format, and runtime reasoning mode. Attributing the differences to specific post-training choices is post-hoc; the study design cannot isolate training-pipeline effects from serving infrastructure, tokenizer differences, architecture details, or active parameter count.

No configuration solved abstention accuracy. Missed-abstention rates ranged from 42.9% to 71.4% across all 18 configurations on 14 unanswerable scenarios. No model, agentic workflow, or agent-skill configuration achieved acceptable abstention accuracy, consistent with models trained for task completion rather than selective prediction. The limitation was not resolved by any of the workflow or knowledge-delivery variations tested here; we conjecture it stems from training objectives that reward task completion over selective prediction and would require training-level intervention, consistent with recent abstention benchmarks [93].

6.2 Implications

For practitioners. When choosing a large language model for agentic retrieval, post-training methodology, runtime reasoning capability, and serving behavior should be evaluated directly rather than inferred from architecture family or active parameter count. The orchestrator-worker pattern provides a structural advantage for models that struggle with autonomous task decomposition, as the GLM-5.1 workflow contrast suggested. However, it introduces latency and token overhead that yields no proportional benefit when the model already reasons effectively within a single context window, and it did not improve Qwen3-Coder under the

functional pass-rate criterion. Practitioners should match knowledge-augmentation strategy to model capability rather than apply it uniformly: in this three-model sample, on-demand skill loading benefited the model with strong autonomous reasoning and burdened the model that required explicit guidance. Deployment readiness is also family-dependent. Lookup and traceability scoping (95.0% and 94.1% pass rates) are near-deployable under supervision, while search (54.0%) and impact analysis (66.7%)—the families closest to the motivating change-scoping use case—remain research-grade; aggregate pass rates alone should not drive deployment decisions. Finally, the high missed-abstention rates across all configurations mean that agentic retrieval systems cannot yet be deployed without supervision. Because the system emits no usable uncertainty signal of its own, human review must be routed by structural signals such as query family or grounding coverage rather than by model self-assessment.

For researchers. The dual-metric evaluation protocol exposes quality differences that neither IR metrics nor LLM-as-a-judge scores show on their own. Future evaluations of agentic retrieval systems should consider pairing both metric families rather than relying on either alone. The persistent missed-abstention rates across all tested configurations suggest that current model training objectives do not address abstention. Benchmarks that report abstention accuracy as a first-class metric alongside task correctness would help track progress toward reliable abstention. Establishing judge-human agreement through expert annotation remains a prerequisite before LLM-based judges can replace traditional evaluation in high-stakes settings.

6.3 Future Work

Four directions remain open for future investigation. First, because thesis time and access to qualified domain experts prevented a representative human-rating study, future work should collect human-expert ratings on a subset of benchmark executions. This would establish judge-human agreement (Cohen's κ) and quantify how well the automated scoring protocol approximates domain-expert assessment. Second, replicating the benchmark on a different industrial codebase would test whether the observed model-dependent knowledge-augmentation effects and orchestration-strategy advantages generalize beyond telecommunications test artifacts. Third, constructing a dedicated abstention benchmark would provide a structured baseline for evaluating future models that address abstention in their training objectives. Such a benchmark should include a larger set of unanswerable scenarios with varied nonexistence types: fabricated identifiers, valid identifiers with no linked artifacts, and partially answerable queries. Fourth, executing the same model weights on an alternative serving infrastructure would isolate whether behavioral differences attributed to model characteristics, in particular the GLM-5.1 ReAct empty-response failure affecting 14.1% of executions, stem from the model or from provider-specific serving interactions.

A E2E Harness Prompts and Configurations

This appendix provides the redacted prompt templates, verdict schemas, and configuration parameters for the end-to-end LLM-as-a-Judge evaluation harness described in Chapter 3. Content is presented verbatim (aside from redactions) to support auditability and replicability.

A.1 Judge system prompt

The following prompt is provided to the judge LLM for each extraction it evaluates.

```
# LLM-as-Judge System Prompt

You are an LLM judge evaluating a retrieval agent's response against ground
truth for a domain knowledge system.

## Input Sections

You will receive:

- **Scenario ID**: Unique identifier for the test scenario
- **Knowledge Config**: Prompt, Skill, or Skill+Meta
- **Workflow Mode**: react or plan_execute
- **SUT Model**: Model identifier for the evaluated agent
- **Query**: The user question the agent was asked
- **Ground Truth**: Contains expected_ids (artifact IDs the agent must find),
  expected_entities (requirement reference IDs the agent must surface), and
  reference_answer (gold-standard answer for comparison)
- **Tool Calls**: Ordered list of tools the agent called. Each includes name,
  status, parameters, and result (truncated to 2000 chars). Entries with
  "source": "subagent" are from sub-agents in Plan-and-Execute mode.
- **Agent Response**: The agent's final textual answer
- **Agent Response Documents** (optional): Markdown documents written by the
  agent or its sub-agents during execution. When present, these are the
  primary output.

## Evaluation
```

```
### Ground Truth Verification

Check ALL agent output (response + response documents) for:

1. **expected_ids**: Which artifact IDs from the list appear in the output?
   List found and missing separately. Use substring matching.

2. **expected_entities**: Which requirement reference IDs appear?
   Use substring matching.

3. **reference_answer_alignment**: Compare the agent's answer against the
   reference answer. Score 0.0-1.0:
   - 1.0: Agent covers all key facts, no contradictions
   - 0.7-0.9: Most key facts present, minor omissions, no contradictions
   - 0.4-0.6: Partial coverage, some facts missing or imprecise
   - 0.1-0.3: Major facts missing or significant inaccuracies
   - 0.0: Contradicts the reference answer or completely wrong

### Quality Dimensions

Score each from 1-5:

| Score | Meaning                          |
|-------|----------------------------------|
| 1     | Absent or completely wrong       |
| 2     | Present but mostly inadequate    |
| 3     | Adequate but unremarkable        |
| 4     | Good with minor gaps             |
| 5     | Excellent, comprehensive         |

Dimensions:
- completeness: Does the response address all parts of the query?
- relevance: Is all content relevant to the query (no filler/hallucination)?
- coherence: Is the response well-structured and logically organized?
- evidence_grounding: Are claims supported by evidence from tool results?

### Status

Set status to "pass" if ALL of:
- All expected_ids found (none missing)
- At least 80% of expected_entities found
- reference_answer_alignment >= 0.7

Otherwise "fail".

### Overall Score

Compute overall_score (0.0-1.0):
- ID coverage: 30% weight (fraction of expected_ids found)
- Entity coverage: 30% weight (fraction of expected_entities found)
- Reference alignment: 20% weight (the alignment score directly)
- Quality average: 20% weight (mean of 4 dimension scores, normalized to 0-1)

### Summary

Write 2-3 sentences covering key strengths and weaknesses.
```

A.2 Verdict schema

The judge must produce a JSON object conforming to the following schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "Verdict",
  "type": "object",
  "required": [
    "scenario_id", "knowledge_config",
    "workflow_mode", "sut_model", "status",
    "overall_score", "ground_truth", "quality", "summary"
  ],
  "properties": {
    "scenario_id": { "type": "string" },
    "knowledge_config": { "type": "string" },
    "workflow_mode": { "type": "string" },
    "sut_model": { "type": "string" },
    "status": { "enum": ["pass", "fail"] },
    "overall_score": {
      "type": "number", "minimum": 0.0, "maximum": 1.0
    },
    "ground_truth": {
      "type": "object",
      "required": [
        "expected_ids_found", "expected_ids_missing",
        "expected_entities_found", "expected_entities_missing",
        "reference_answer_alignment", "details"
      ],
      "properties": {
        "expected_ids_found": {
          "type": "array", "items": { "type": "string" }
        },
        "expected_ids_missing": {
          "type": "array", "items": { "type": "string" }
        },
        "expected_entities_found": {
          "type": "array", "items": { "type": "string" }
        },
        "expected_entities_missing": {
          "type": "array", "items": { "type": "string" }
        },
        "reference_answer_alignment": {
          "type": "number", "minimum": 0.0, "maximum": 1.0
        },
        "details": { "type": "string" }
      }
    },
    "quality": {
      "type": "object",
      "required": [
        "completeness", "relevance", "coherence", "evidence_grounding"
      ],
      "properties": {
        "completeness": { "$ref": "#/$defs/QualityDimensionScore" },
        "relevance": { "$ref": "#/$defs/QualityDimensionScore" },
        "coherence": { "$ref": "#/$defs/QualityDimensionScore" },
        "evidence_grounding": { "$ref": "#/$defs/QualityDimensionScore" }
```

```
      }
    },
    "summary": { "type": "string" }
  },
  "$defs": {
    "QualityDimensionScore": {
      "type": "object",
      "required": ["score", "rationale"],
      "properties": {
        "score": { "type": "integer", "minimum": 1, "maximum": 5 },
        "rationale": { "type": "string" }
      }
    }
  }
}
```

A.3 Meta-judge system prompt

The meta-judge compares two independent judgments of the same extraction to assess inter-judge reliability.

```
# Meta-Judge System Prompt

You are a meta-judge comparing two independent judgments (A and B) of the
same agent response. Your job is to determine which judgment is more accurate
by independently verifying the ground truth claims.

## Input Sections

You will receive:

- Query: The user question the agent was asked
- Knowledge Config: Prompt, Skill, or Skill+Meta
- Workflow Mode: react or plan_execute
- SUT Model: Model identifier for the evaluated agent
- Ground Truth: Contains expected_ids, expected_entities, and reference_answer
- Agent Response: The agent's final textual answer
- Agent Response Documents (optional): Markdown documents written by the agent
- Judgment A (Primary): Full verdict JSON from the primary judge run
- Judgment B (Calibration): Full verdict JSON from the calibration judge run

## Evaluation Process

### Step 1: Independent Verification

Before looking at either judgment's claims, independently check:

1. ID verification: For each ID in expected_ids, does it appear (substring
   match) in the agent response or response documents?
2. Entity verification: For each entity in expected_entities, does it appear
   (substring match) in the agent response or response documents?
3. Status determination: Based on your verification, what should the correct
   status be?
   - pass if: ALL expected_ids found AND at least 80% of expected_entities found AND
     response aligns with reference_answer (>=0.7)
   - fail otherwise
```

```
### Step 2: Compare Judgments

For each judgment (A and B):
- Are its expected_ids_found / expected_ids_missing lists correct?
- Are its expected_entities_found / expected_entities_missing lists correct?
- Is its status correct?
- Are its quality scores (1-5) reasonable for the response quality?

### Step 3: Determine Preference

- "tie": Both judgments are equally accurate (same status, same ID/entity
  lists, scores within 1 point)
- "A": Judgment A is more accurate (correct on more ground truth claims)
- "B": Judgment B is more accurate

### Step 4: Bias Detection

- "none": No systematic bias
- "leniency": One or both judgments scored higher than warranted
- "harshness": One or both judgments scored lower than warranted

## Important Rules

- Do your OWN ID/entity verification -- do not trust either judgment's claims
- Use substring matching for IDs and entities (same as primary judge)
- quality_score_delta is absolute difference between A and B overall_score
- judgments_agree means both have same status AND overall_score differs < 0.1
- Be strict: if a judgment claims an ID is found but it's not in the response,
  that judgment is wrong
```

A.4 Meta-judge verdict schema

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "MetaJudgeVerdict",
  "type": "object",
  "required": [
    "scenario_id", "knowledge_config", "workflow_mode", "sut_model",
    "judgments_agree", "preferred_judgment",
    "a_ids_correct", "b_ids_correct",
    "a_entities_correct", "b_entities_correct",
    "a_status_correct", "b_status_correct",
    "quality_score_delta", "bias_detected", "confidence", "reasoning"
  ],
  "properties": {
    "scenario_id": { "type": "string" },
    "knowledge_config": { "type": "string" },
    "workflow_mode": { "type": "string" },
    "sut_model": { "type": "string" },
    "judgments_agree": { "type": "boolean" },
    "preferred_judgment": { "enum": ["A", "B", "tie"] },
    "a_ids_correct": { "type": "boolean" },
    "b_ids_correct": { "type": "boolean" },
    "a_entities_correct": { "type": "boolean" },
    "b_entities_correct": { "type": "boolean" },
    "a_status_correct": { "type": "boolean" },
    "b_status_correct": { "type": "boolean" },
```

```
    "quality_score_delta": {
      "type": "number", "minimum": 0.0, "maximum": 1.0
    },
    "bias_detected": { "enum": ["none", "leniency", "harshness"] },
    "confidence": { "enum": ["high", "medium", "low"] },
    "reasoning": { "type": "string" }
  }
}
```

A.5 Calibration procedure

Calibration re-judges a subset of extractions using the same judge system prompt (Section A.1) in an independent session with no access to prior verdicts. The only differences from the primary judge run are operational: the calibration run processes 156 sampled executions, stratified across the reported model–workflow–knowledge cells while preserving coverage of both pass and fail verdicts, and uses a separate output directory. The meta-judge (Section A.3) then receives both the primary verdict (Judgment A) and calibration verdict (Judgment B) for each sampled extraction and determines which is more accurate through independent ground-truth verification.

A.6 Knowledge configurations

Table A.1 summarizes the three knowledge configurations used as the independent variable in the evaluation.

Table A.1: Knowledge configuration parameters.

Parameter	Prompt	Skill	Skill+Meta
System prompt	Monolithic (all domain policies inline)	Thinned kernel	Thinned kernel
Domain skills	Disabled	Enabled	Enabled
User preferences	None	None	Injected

Prompt delivers all domain policies (knowledge model, tool routing, response format) inline in a single monolithic system prompt, with skill modules disabled (Section 3.3; Appendix C). **Skill** thins the system prompt to a kernel and delivers the same policies through three on-demand skill modules (domain knowledge, tool routing, response format). **Skill+Meta** additionally injects a user-preferences file containing meta-cognitive guidance (e.g., preferred response structure, known pitfalls). All three configurations carry equivalent domain-knowledge content; only the delivery mechanism differs.

A.7 Orchestration summary

Evaluation is executed via agentic coding tool sessions, one session per knowledge configuration and SUT model combination ($3 \times 1 = 3$ sessions for a single-model run; $3 \times 3 = 9$ for the full matrix). Each session processes approximately 104 extractions (52 scenarios $\times$ 2 workflow modes). Extractions are dispatched to parallel sub-agents, each of which reads the judge system prompt, evaluates its assigned extractions, and writes a verdict file. After all sub-agents complete, a consolidation script merges verdict files for downstream analysis.

A.8 Redaction conventions

The following placeholder conventions protect proprietary identifiers while preserving evaluation logic:

- Domain-specific artifact identifiers (e.g., test case IDs) are replaced with `<TC_ID>`.
- Requirement reference entity identifiers are replaced with `<ENTITY_ID>`.
- The system under test is referred to as "the retrieval agent" or "the SUT agent" rather than by its internal product name.
- The domain knowledge system is referred to generically rather than by its internal name.
- The upstream source system is referred to as "the source system."
- Model identifiers (e.g., `GLM-5.1-FP8`) are retained as they refer to publicly available models.

B AgentFS Schema

This appendix provides the PostgreSQL data definition for the `agentfs` virtual filesystem introduced in Section 3.2.1. The schema implements the DeepAgents `BackendProtocol` contract [46] so that the agent's file-oriented tools (read, write, list, search) operate over a durable, session-scoped store rather than over process memory or the host filesystem.

The schema in Listing B.1 reflects three design choices:

- **Append-only revisions.** The `file_revision` table records every write as a new row, and the `file.current_revision_id` pointer is updated to reference the latest revision. Older revisions are never overwritten, supporting post-hoc replay and audit of agent edits.
- **Dual-key isolation.** Each row in `file` and `file_blob` carries an (`assistant_id`, `thread_id`) pair, and a uniqueness index on (`assistant_id`, `thread_id`, `path`) prevents path collisions across concurrent conversation sessions.
- **Text/binary split.** Text artifacts (markdown summaries, CSV and JSON exports) live in `file/file_revision`, where the content is indexable and diffable; large binary artifacts (XLSX exports) live in `file_blob`, avoiding bloat of the revision history with non-text payloads.

Idempotent DDL guards (`IF NOT EXISTS`, conditional `ALTER TABLE` blocks) and secondary performance indexes are omitted from the listing for readability; the canonical definitions live in `databases/postgres/init_v2/13_agentfs.sql` and `14_agentfs_indexes.sql` in the project repository.

Listing B.1: AgentFS schema (PostgreSQL). Three tables with the cycle foreign key from `file.current_revision_id` to `file_revision`, and the uniqueness constraints that enforce per-session path isolation.

```
CREATE TABLE agentfs.file (
  file_id              bigserial PRIMARY KEY,
  assistant_id         text NOT NULL,
  thread_id            text NOT NULL,
  path                 text NOT NULL,
  current_revision_id  bigint NULL,
  created_at           timestamptz NOT NULL DEFAULT now(),
  updated_at           timestamptz NOT NULL DEFAULT now()
);

CREATE TABLE agentfs.file_revision (
  revision_id   bigserial PRIMARY KEY,
  file_id       bigint NOT NULL
                  REFERENCES agentfs.file(file_id) ON DELETE CASCADE,
  content_text  text NOT NULL,
  content_hash  bytea NULL,
  created_at    timestamptz NOT NULL DEFAULT now(),
  source        text NOT NULL DEFAULT 'write'
);

ALTER TABLE agentfs.file_revision
  ADD CONSTRAINT agentfs_file_revision_file_revision_uidx
  UNIQUE (file_id, revision_id);

CREATE TABLE agentfs.file_blob (
  blob_id        bigserial PRIMARY KEY,
  assistant_id   text NOT NULL,
  thread_id      text NOT NULL,
  path           text NOT NULL,
  content_bytes  bytea NOT NULL,
  mime_type      text NOT NULL DEFAULT 'application/octet-stream',
  created_at     timestamptz NOT NULL DEFAULT now(),
  updated_at     timestamptz NOT NULL DEFAULT now()
);

-- Cycle FK: each file's current revision must belong to that same file.
ALTER TABLE agentfs.file
  ADD CONSTRAINT agentfs_file_current_revision_id_fkey
  FOREIGN KEY (file_id, current_revision_id)
  REFERENCES agentfs.file_revision(file_id, revision_id);

-- Per-session path uniqueness (text artifacts).
CREATE UNIQUE INDEX agentfs_file_namespace_path_uidx
  ON agentfs.file (assistant_id, thread_id, path);

-- Per-session path uniqueness (binary artifacts).
CREATE UNIQUE INDEX agentfs_file_blob_namespace_path_uidx
  ON agentfs.file_blob (assistant_id, thread_id, path);
```

C System Prompt Configurations

This appendix documents the three knowledge configurations evaluated for RQ2 (Section 3.3) at the level of *where* each policy lives in each configuration, rather than reproducing the full prompt text. The Prompt configuration delivered all retrieval, evidence, and response policies inside a single monolithic system prompt. The Skill configuration delivered the same policies through a thinner kernel prompt plus three on-demand skill modules activated by metadata match. The Skill+Meta configuration extended the Skill configuration with a small coordinator module that prescribed module load order. Module names and the artifact-URL scheme are renamed below for publication; the structural organization and policy content are preserved.

Table C.1 summarizes the policy-to-host mapping across the three configurations. Figure C.1 renders the same mapping as a schematic, making the offloading and coordination shifts visible at a glance.

Table C.1: Policy delivery across the three knowledge configurations. Each row names a policy category; each cell names the host that carried that policy in the corresponding configuration.

Policy category	**Prompt**	**Skill**	**Skill+Meta**
Evidence Contract	kernel prompt	kernel prompt	kernel prompt
Tool Abstraction	kernel prompt	kernel prompt	kernel prompt
Knowledge Model	kernel prompt	`domain-knowledge`	`domain-knowledge`
Tool Routing	kernel prompt	`tool-routing`	`tool-routing`
Response / Citation Format	kernel prompt	`response-format`	`response-format`
Load-order Guidance	–	–	`preferences`

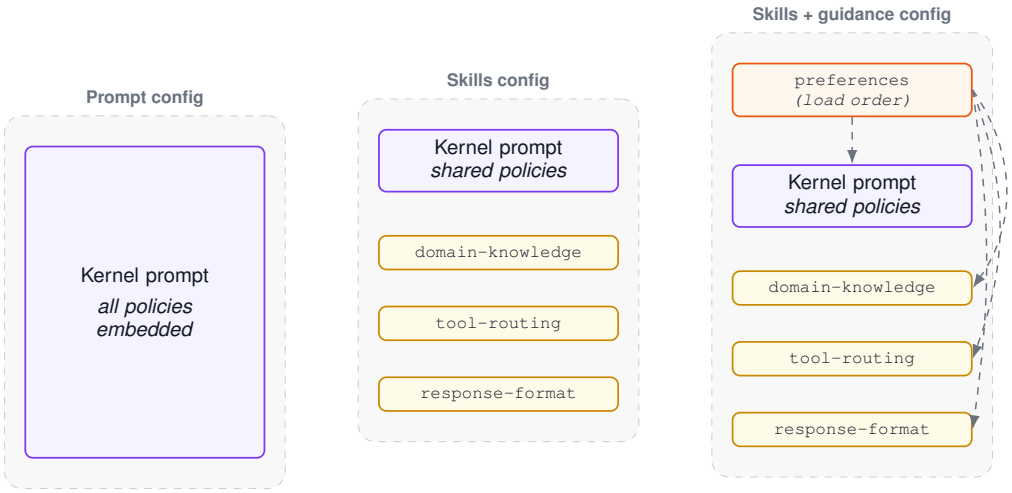


Figure C.1: Schematic of the three knowledge configurations. The Prompt configuration carries all policies inside a single kernel block. The Skill configuration thins the kernel and offloads three policy domains to standalone modules activated on metadata match. The Skill+Meta configuration adds a coordinator module (dashed arrows) that prescribes module load order at conversation start.

C.1 Citation Contract excerpt across configurations

The policy text itself was largely identical across configurations; only its host changed. Figure C.2 shows the Citation Contract policy as it appeared embedded in the kernel prompt under the Prompt configuration (left) and as a section of the `response-format` skill module under the Skill configuration (right). The wording and rule set are preserved; only the host context differs.

```
# Citation Contract

- Answers that cite specific artifact
  IDs or metadata MUST end with
  `## Sources`. Conversational or
  follow-up answers without structured
  claims do not need `## Sources`.
- Each source entry MUST use exactly
  this format:
  `- [S1](artifact://<thread_id>
     ?path=<artifact_path>)`
- Use sequential labels: `S1`, `S2`,
  `S3`, etc.
- Every source in `## Sources` MUST be
  referenced in the answer or evidence.
- Do not include unused sources.
- Do not include raw `artifact://...`
  text outside markdown links.
- Do not expose internal tool names,
  database names, SQL queries, vector
  search details, embeddings, ranking
  logic, or retrieval mechanics.
```

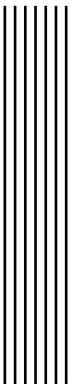
(a) Prompt configuration (excerpt from kernel)

```
## Citation Contract

- Every structured answer MUST end
  with `## Sources`.
- Format:
  `- [S1](artifact://<thread_id>
     ?path=<artifact_path>)`
- Sequential labels: S1, S2, S3, etc.
- Every source in `## Sources` MUST be
  referenced in answer or evidence.
- Do not include unused sources.
- Do not expose internal tool names,
  database names, SQL queries, vector
  search details, embeddings, ranking
  logic, or retrieval mechanics.
```

(b) Skill configuration (excerpt from `response-format`)

Figure C.2: The Citation Contract policy across two knowledge configurations. The Prompt configuration embeds the rule set in the kernel system prompt; the Skill configuration relocates it to a dedicated module loaded on demand. Wording is preserved (modulo whitespace and conversational-exception phrasing in the kernel variant); only the host changes.



Bibliography

[1] D. Graham and M. Fewster, *Experiences of test automation: case studies of software test automation*. Addison-Wesley Professional, 2012.

[2] A. Labuschagne, L. Inozemtseva, and R. Holmes, "Measuring the cost of regression testing in practice: A study of java projects using continuous integration," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017, Paderborn, Germany: Association for Computing Machinery, 2017, pp. 821–830, ISBN: 9781450351058. DOI: 10.1145/3106237.3106288. [Online]. Available: https://doi.org/10.1145/3106237.3106288.

[3] T. A. Ghaleb, D. A. Da Costa, and Y. Zou, "An empirical study of the long duration of continuous integration builds," *Empirical Software Engineering*, vol. 24, no. 4, pp. 2102–2139, 2019.

[4] G. Rothermel and M. Harrold, "Analyzing regression test selection techniques," *IEEE Transactions on Software Engineering*, vol. 22, no. 8, pp. 529–551, 1996. DOI: 10.1109/32.536955.

[5] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:34563682.

[6] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 1, pp. 1–42, 2013.

[7] A. Shi, P. Zhao, and D. Marinov, "Understanding and improving regression test selection in continuous integration," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, 2019, pp. 228–238. DOI: 10.1109/ISSRE.2019.00031.

[8] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software testing with large language models: Survey, landscape, and vision," *IEEE Transactions on Software Engineering*, vol. 50, no. 4, pp. 911–936, 2024. DOI: 10.1109/TSE.2024.3368208.

[9] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of IR techniques," *ACM Trans. Inf. Syst.*, vol. 20, pp. 422–446, 2002. [Online]. Available: https://api.semanticscholar.org/CorpusID:1981391.

[10] C. D. Manning, P. Raghavan, and H. Schutze, *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008, ISBN: 978-0521865715. [Online]. Available: https://nlp.stanford.edu/IR-book/.

[11] F. Gomes De Oliveira Neto, J. Horkoff, E. Knauss, R. Kasauli, and G. Liebel, "Challenges of aligning requirements engineering and system testing in large-scale agile: A multiple case study," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, 2017, pp. 315–322. DOI: 10.1109/REW.2017.33.

[12] B. Wang, H. Wang, R. Luo, S. Zhang, and Q. Zhu, "A systematic mapping study of information retrieval approaches applied to requirements trace recovery," in *SEKE*, 2022, pp. 1–6.

[13] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation: A retrospective," *IEEE Transactions on Software Engineering*, vol. 51, no. 3, pp. 825–832, 2025. DOI: 10.1109/TSE.2025.3534027.

[14] M. Ruiz, J. Y. Hu, and F. Dalpiaz, "Why don't we trace? a study on the barriers to software traceability in practice," *Requirements Engineering*, vol. 28, no. 4, pp. 619–637, 2023.

[15] D. Fucci, E. Alégroth, and T. Axelsson, "When traceability goes awry: An industrial experience report," *Journal of Systems and Software*, vol. 192, p. 111 389, 2022.

[16] M. Naiseh, D. Al-Thani, N. Jiang, and R. Ali, "Explainable recommendation: When design meets trust calibration," *World Wide Web*, vol. 24, no. 5, pp. 1857–1884, 2021.

[17] M. Naiseh, D. Al-Thani, N. Jiang, and R. Ali, "How the different explanation classes impact trust calibration: The case of clinical decision support systems," *International Journal of Human-Computer Studies*, vol. 169, p. 102 941, 2023.

[18] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, *Retrieval-augmented generation for knowledge-intensive NLP tasks*, 2021. arXiv: 2005.11401 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2005.11401.

[19] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems," in *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, IEEE, 2023, pp. 31–53.

[20] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, S. Yu, B. Zhang, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 8, pp. 1–79, 2024.

[21] V. Kesri, A. Nayak, and K. Ponnalagu, "Autokg-an automotive domain knowledge graph for software testing: A position paper," in *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, IEEE, 2021, pp. 234–238.

[22] S. Radhakrishnan, A. G. Cecchettini, S. Wendler, and A. Graf, "Create, explore and analyse traceability knowledge graphs," in *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, IEEE, 2023, pp. 453–454.

[23] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitansky, R. O. Ness, and J. Larson, *From local to global: A graph RAG approach to query-focused summarization*, 2025. arXiv: 2404.16130 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2404.16130.

[24] C. Sharma, "Retrieval-augmented generation: A comprehensive survey of architectures, enhancements, and robustness frontiers," *arXiv preprint arXiv:2506.00054*, 2025.

[25] M. Cheng, Y. Luo, J. Ouyang, Q. Liu, H. Liu, L. Li, S. Yu, B. Zhang, J. Cao, J. Ma, D. Wang, and E. Chen, *A survey on knowledge-oriented retrieval-augmented generation*, 2025. DOI: 10.48550/arXiv.2503.10677. arXiv: 2503.10677 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2503.10677.

[26] S. Min, K. Krishna, X. Lyu, M. Lewis, W.-t. Yih, P. W. Koh, M. Iyyer, L. Zettlemoyer, and H. Hajishirzi, *Factscore: Fine-grained atomic evaluation of factual precision in long form text generation*, 2023. arXiv: 2305.14251 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2305.14251.

[27] Z. Zhang, W. You, T. Wu, X. Wang, J. Li, and M. Zhang, "A survey of generative information extraction," in *Proceedings of the 31st International Conference on Computational Linguistics*, Abu Dhabi, UAE: Association for Computational Linguistics, Jan. 2025, pp. 4840–4870. [Online]. Available: https://aclanthology.org/2025.coling-main.324/.

[28] H. Bian, *Llm-empowered knowledge graph construction: A survey*, 2025. DOI: 10.48550/arXiv.2510.20345. arXiv: 2510.20345 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2510.20345.

[29] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, *React: Synergizing reasoning and acting in language models*, 2023. arXiv: 2210.03629 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2210.03629.

[30] D. Fuchß, T. Hey, J. Keim, H. Liu, N. Ewald, T. Thirolf, and A. Koziolek, "Lissa: Toward generic traceability link recovery through retrieval-augmented generation," in *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering. ICSE*, vol. 25, 2025.

[31] P. P. Khine and Z. Wang, "A review of polyglot persistence in the big data world," *Information*, vol. 10, no. 4, p. 141, 2019. DOI: 10.3390/info10040141. [Online]. Available: https://www.mdpi.com/2078-2489/10/4/141.

[32] J. de Curtò, I. de Zarzà, and C. T. Calafate, "Integrating polyglot persistence with large language models for scalable social network applications," *Procedia Computer Science*, vol. 270, pp. 733–743, 2025. DOI: 10.1016/j.procs.2025.09.193. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050925028637.

[33] E. M. Voorhees and D. Harman, "Overview of the eighth Text REtrieval Conference (TREC-8)," in *Proceedings of the Eighth Text REtrieval Conference (TREC-8)*, NIST Special Publication 500-246, National Institute of Standards and Technology (NIST), 1999.

[34] Anthropic. "Building effective agents." Engineering blog post defining orchestrator-worker and other agentic workflow patterns, Accessed: May 17, 2026. [Online]. Available: https://www.anthropic.com/engineering/building-effective-agents.

[35] M. Hashimoto. "My AI adoption journey: Engineer the harness." Coins the term "harness engineering" (February 2026); formulates AGENT = MODEL + HARNESS, Accessed: May 27, 2026. [Online]. Available: https://mitchellh.com/writing/my-ai-adoption-journey#step-5-engineer-the-harness.

[36] H. Seong, L. Yin, H. Zhang, and Z. Shi, *The last harness you'll ever build*, 2026. arXiv: 2604.21003 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2604.21003.

[37] LangChain, Inc. "LangGraph DeepAgents." Orchestrator-worker agent framework built on LangGraph, Accessed: May 17, 2026. [Online]. Available: https://docs.langchain.com/oss/python/deepagents/harness.

[38] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010, ISBN: 978-0321601919.

[39] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: BM25 and beyond," *Found. Trends Inf. Retr.*, vol. 3, no. 4, pp. 333–389, Apr. 2009, ISSN: 1554-0669. DOI: 10.1561/1500000019. [Online]. Available: https://doi.org/10.1561/1500000019.

[40] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Y. Wu, S. Edunov, D. Chen, and W.-t. Yih, "Dense passage retrieval for open-domain question answering," *ArXiv*, vol. abs/2004.04906, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:215737187.

[41] G. V. Cormack, C. L. A. Clarke, and S. Buettcher, "Reciprocal rank fusion outperforms condorcet and individual rank learning methods," in *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '09, Boston, MA, USA: Association for Computing Machinery, 2009, pp. 758–759, ISBN: 9781605584836. DOI: 10.1145/1571941.1572114. [Online]. Available: https://doi.org/10.1145/1571941.1572114.

[42] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, J. E. L. Gayo, S. Kirrane, S. Neumaier, A. Polleres, R. Navigli, A.-C. N. Ngomo, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, and A. Zimmermann, "Knowledge graphs," *ACM Computing Surveys (CSUR)*, vol. 54, pp. 1–37, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:212414739.

[43] W. Shen, J. Wang, and J. Han, "Entity linking with a knowledge base: Issues, techniques, and solutions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, pp. 443–460, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:16320392.

[44] J. Huang and J. T. Zhou, *A two-dimensional framework for AI agent design patterns: Cognitive function and execution topology*, 2026. arXiv: 2605.13850 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2605.13850.

[45] L. E. Erdogan, N. Lee, S. Kim, S. Moon, H. Furuta, G. Anumanchipalli, K. Keutzer, and A. Gholami, *Plan-and-act: Improving planning of agents for long-horizon tasks*, 2025. arXiv: 2503.09572 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2503.09572.

[46] LangChain, Inc. "DeepAgents backends." Pluggable filesystem backends including virtual filesystem implementation over S3 or PostgreSQL, Accessed: May 25, 2026. [Online]. Available: https://docs.langchain.com/oss/python/deepagents/backends.

[47] LangChain, Inc. "DeepAgents sandboxes." Isolated execution environments for agent filesystem and shell operations, Accessed: May 25, 2026. [Online]. Available: https://docs.langchain.com/oss/python/deepagents/sandboxes.

[48] Anthropic. "Effective context engineering for AI agents." Strategies for curating context in long-horizon agentic systems: compaction, structured note-taking, and sub-agent architectures, Accessed: May 25, 2026. [Online]. Available: https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents.

[49] LangChain, Inc. "DeepAgents context engineering." Context compression via offloading and summarization for long-running agent sessions, Accessed: May 25, 2026. [Online]. Available: https://docs.langchain.com/oss/python/deepagents/context-engineering.

[50] Anthropic. "Equipping agents for the real world with agent skills." Introduced the Agent Skills format; published as an open standard for cross-platform portability in December 2025, Accessed: May 20, 2026. [Online]. Available: https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills.

[51] R. Xu and Y. Yan, *Agent skills for large language models: Architecture, acquisition, security, and the path forward*, 2026. arXiv: 2602.12430 [cs.MA]. [Online]. Available: https://arxiv.org/abs/2602.12430.

[52] X. Li, W. Chen, Y. Liu, S. Zheng, X. Chen, Y. He, Y. Li, B. You, H. Shen, J. Sun, S. Wang, Q. Zeng, D. Wang, X. Zhao, Y. Wang, R. Ben Chaim, Z. Di, Y. Gao, J. He, Y. He, L. Jing, L. Kong, X. Lan, J. Li, S. Li, Y. Li, Y. Lin, X. Liu, X. Liu, H. Lyu, Z. Ma, B. Wang, R. Wang, T. Wang, W. Ye, Y. Zhang, H. Xing, Y. Xue, S. Dillmann, and H.-c. Lee, "Skillsbench: Benchmarking how well agent skills work across diverse tasks," *arXiv preprint arXiv:2602.12670*, 2026.

[53] E. M. Voorhees, "The TREC-8 question answering track report," in *Proceedings of the Eighth Text REtrieval Conference (TREC-8)*, National Institute of Standards and Technology (NIST), 1999, pp. 77–82. [Online]. Available: https://api.semanticscholar.org/CorpusID:16944215.

[54] Y. Geifman and R. El-Yaniv, "Selective classification for deep neural networks," *arXiv preprint arXiv:1705.08500*, 2017. DOI: 10.48550/arXiv.1705.08500.

[55] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for SQuAD," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 2018. DOI: 10.48550/arXiv.1806.03822.

[56] X. Peng, P. K. Choubey, C. Xiong, and C.-S. Wu, "Unanswerability evaluation for retrieval augmented generation," *arXiv preprint arXiv:2412.12300*, 2024. DOI: 10.48550/arXiv.2412.12300.

[57] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, *Judging LLM-as-a-judge with MT-Bench and Chatbot Arena*, 2023. arXiv: 2306.05685 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2306.05685.

[58] H. Silva, M. Mendes, and H. G. Oliveira, *Meta-judging with large language models: Concepts, methods, and challenges*, 2026. arXiv: 2601.17312 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2601.17312.

[59] B. T. Willard and R. Louf, *Efficient guided generation for large language models*, 2023. DOI: 10.48550/arXiv.2307.09702. arXiv: 2307.09702 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2307.09702.

[60] G. Faggioli, L. Dietz, C. L. A. Clarke, G. Demartini, M. Hagen, C. Hauff, N. Kando, E. Kanoulas, M. Potthast, B. Stein, and H. Wachsmuth, "Perspectives on large language models for relevance judgment," in *Proceedings of the 2023 ACM SIGIR International Conference on Theory of Information Retrieval (ICTIR)*, 2023, pp. 39–50. DOI: 10.1145/3578337.3605136.

[61] P. Thomas, S. Spielman, N. Craswell, and B. Mitra, "Large language models can accurately predict searcher preferences," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2024, pp. 1930–1940.

[62] S. Upadhyay, R. Pradeep, N. Thakur, N. Craswell, and J. Lin, *UMBRELA: UMbrela is the (open-source reproduction of the) Bing RELevance Assessor*, 2024. arXiv: 2406.06519 [cs.IR]. [Online]. Available: https://arxiv.org/abs/2406.06519.

[63] J. Saad-Falcon, O. Khattab, C. Potts, and M. Zaharia, "ARES: An automated evaluation framework for retrieval-augmented generation systems," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2024. [Online]. Available: https://aclanthology.org/2024.naacl-long.20/.

[64] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, *Ragas: Automated evaluation of retrieval augmented generation*, 2025. arXiv: 2309.15217 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2309.15217.

[65] G. Liu, Y. Qu, J. Schneider, A. Singh, and A. Kumar, *CaRT: Teaching LLM agents to know when they know enough*, 2025. DOI: 10.48550/arXiv.2510.08517. arXiv: 2510.08517 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2510.08517.

[66] H. Wang, C. Qian, W. Zhong, X. Chen, et al., *Acting less is reasoning more: Teaching model to act efficiently*, 2025. DOI: 10.48550/arXiv.2504.14870. arXiv: 2504.14870 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2504.14870.

[67] B. Chen, C. Shu, E. Shareghi, N. Collier, K. Narasimhan, and S. Yao, *FireAct: Toward language agent fine-tuning*, 2023. DOI: 10.48550/arXiv.2310.05915. arXiv: 2310.05915 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2310.05915.

[68] Z. Wang, X. Zeng, W. Liu, L. Li, Y. Wang, L. Shang, X. Jiang, Q. Liu, and K.-F. Wong, *ToolFlow: Boosting LLM tool-calling through natural and coherent dialogue synthesis*, 2024. DOI: 10.48550/arXiv.2410.18447. arXiv: 2410.18447 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2410.18447.

[69] Z. Yu, L. Yang, J. Zou, S. Yan, and M. Wang, *Demystifying reinforcement learning in agentic reasoning*, 2025. DOI: 10.48550/arXiv.2510.11701. arXiv: 2510.11701 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2510.11701.

[70] C. Qian, E. C. Acikgoz, Q. He, H. Wang, X. Chen, D. Hakkani-Tür, G. Tur, and H. Ji, *ToolRL: Reward is all tool learning needs*, 2025. DOI: 10.48550/arXiv.2504.13958. arXiv: 2504.13958 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2504.13958.

[71] F. Yan, H. Mao, C. C.-J. Ji, T. Zhang, S. G. Shi, I. Stoica, and J. E. Gonzalez, *Berkeley function calling leaderboard*, https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html, Accessed 2026-02, 2024.

[72] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan, *τ-bench: A benchmark for tool-agent-user interaction in real-world domains*, 2024. DOI: 10.48550/arXiv.2406.12045. arXiv: 2406.12045 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2406.12045.

[73] DeepSeek-AI, *Deepseek-v3.2: Pushing the frontier of open large language models*, 2025. arXiv: 2512.02556 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2512.02556.

[74] GLM-5-Team, *Glm-5: From vibe coding to agentic engineering*, 2026. arXiv: 2602.15763 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2602.15763.

[75] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng, M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, and Z. Qiu, *Qwen3 technical report*, 2025. arXiv: 2505.09388 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2505.09388.

[76] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, and D. Guo, *Deepseekmath: Pushing the limits of mathematical reasoning in open language models*, 2024. arXiv: 2402.03300 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2402.03300.

[77] Z. Liu, C. Chen, W. Li, P. Qi, T. Pang, C. Du, W. S. Lee, and M. Lin, *Understanding R1-zero-like training: A critical perspective*, Introduces Dr. GRPO; identifies the GRPO response-length bias, 2025. arXiv: 2503.20783 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2503.20783.

[78] A. Kane, *Pgvector: Open-source vector similarity search for PostgreSQL*, Accessed 2026-02-10, 2026. [Online]. Available: https://github.com/pgvector/pgvector.

[79] LangChain, Inc. "LangGraph persistence." Checkpoint-based state persistence for LangGraph agents, Accessed: May 17, 2026. [Online]. Available: https://docs.langchain.com/oss/python/langgraph/persistence.

[80] ParadeDB Developers, *pg_search: Elastic-quality full text search inside PostgreSQL*, PostgreSQL Extension, built on Tantivy, using the BM25 algorithm, First stable release (v0.6.0) on November 14, 2023; development ongoing, ParadeDB, 2023. [Online]. Available: https://github.com/paradedb/paradedb.

[81] Y. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, pp. 824–836, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:8915893.

[82] Y. Zhang, M. Li, D. Long, X. Zhang, H. Lin, B. Yang, P. Xie, A. Yang, D. Liu, J. Lin, F. Huang, and J. Zhou, *Qwen3 embedding: Advancing text embedding and reranking through foundation models*, 2025. arXiv: 2506.05176 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2506.05176.

[83] W. Cochran, *Sampling Techniques* (Wiley Series in Probability and Statistics). Wiley, 1977, ISBN: 9780471162407. [Online]. Available: https://books.google.se/books?id=8Y4QAQAAIAAJ.

[84] R. Haldar and J. Hockenmaier, "Rating roulette: Self-inconsistency in LLM-as-a-judge frameworks," in *Findings of the Association for Computational Linguistics: EMNLP 2025*, Association for Computational Linguistics, 2025, pp. 24 986–25 004. DOI: 10.18653/v1/2025.findings-emnlp.1361. [Online]. Available: http://dx.doi.org/10.18653/v1/2025.findings-emnlp.1361.

[85] F. Lau, *Same input, different scores: A multi model study on the inconsistency of LLM judge*, 2026. arXiv: 2603.04417 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2603.04417.

[86] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960. DOI: 10.1177/001316446002000104. [Online]. Available: https://doi.org/10.1177/001316446002000104.

[87] K. Krippendorff, *Computing krippendorff's alpha-reliability*, Departmental Papers (ASC), Annenberg School for Communication, University of Pennsylvania, 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:59901023.

[88] J. Jung, F. Brahman, and Y. Choi, *Trust or escalate: LLM judges with provable guarantees for human agreement*, 2024. arXiv: 2407.18370 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2407.18370.

[89] J. R. Carpenter and J. F. Bithell, "Bootstrap confidence intervals: When, which, what? a practical guide for medical statisticians," *Statistics in medicine*, vol. 19, no. 9, pp. 1141–1164, 2000. [Online]. Available: https://api.semanticscholar.org/CorpusID:15169388.

[90] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979, ISSN: 03036898, 14679469. [Online]. Available: http://www.jstor.org/stable/4615733.

[91] C. Buckley and E. M. Voorhees, "Retrieval evaluation with incomplete information," in *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, ACM, 2004, pp. 25–32.

[92] A. Kamath, R. Jia, and P. Liang, "Selective question answering under domain shift," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020. DOI: 10.48550/arXiv.2006.09462.

[93] P. Kirichenko, M. Ibrahim, K. Chaudhuri, and S. J. Bell, "Abstentionbench: Reasoning LLMs fail on unanswerable questions," *arXiv preprint arXiv:2506.09038*, 2025. DOI: 10.48550/arXiv.2506.09038.

[94] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977, ISSN: 0006341X, 15410420. Accessed: May 20, 2026.

[95] J. Evans, *Straightforward statistics for the behavioral sciences*. Brooks/Cole Publishing Company, 1996, tex.lccn: 95018736, ISBN: 978-0-534-23100-2. [Online]. Available: https://books.google.de/books?id=8Ca2AAAAIAAJ.