

A Reference Architecture for Integrating Agentic AI into Enterprise Software: Responsibilities, Interactions, and Variability

Mingqin Yu

UNSW Sydney

Sydney, Australia

mingchin.yuyu@gmail.com

Fethi Rabhi

School of Computer Science and Engineering

UNSW Sydney, Australia

f.rabhi@unsw.edu.au

Liming Zhu

CSIRO

Sydney, Australia

liming.zhu@csiro.au

Qinghua Lu

CSIRO's Data61

Sydney, Australia

qinghua.lu@data61.csiro.au

Abstract—Agentic AI moves generative AI applications toward systems that coordinate reasoning, enterprise context, model inference, and external actions. Integrating these capabilities into enterprise software introduces diverse architectural concerns spanning orchestration, knowledge and memory, model access, tools and actions, security and access control, runtime infrastructure, and evaluation. Existing research and industry architectures address many of these concerns, but differ in scope, abstraction, responsibility organization, interaction representation, and assumptions about which capabilities are required. This paper presents a technology-neutral reference architecture for enterprise agentic AI integration. We synthesize architectural capabilities from research publications, industry architecture guidance, and interoperability specifications according to architectural purpose. The architecture organizes these capabilities into ten responsibilities and defines boundary invariants, typed cross-responsibility interactions, and variation points that distinguish architectural responsibilities from condition-dependent capabilities and replaceable realization mechanisms. We instantiate the architecture as an ESG portfolio-analysis and pre-trade compliance agent built over an existing service-oriented ESG metric computation and reporting system, and complement the case with a mapping of an independently published agentic architecture. The case realizes seven of the ten responsibilities; different questions take different interaction paths, controls activate only under their conditions, and the probed boundary invariants hold. The independent architecture maps onto the proposed responsibilities without requiring an additional one. These results provide initial evidence for the architecture's organization and applicability rather than its completeness. The deployment and recorded results are available as a replication package.

Index Terms—Agentic AI, reference architecture, software architecture, enterprise AI, enterprise integration, architectural variability

I. INTRODUCTION

Agentic AI extends generative AI applications from individual model responses toward goal-directed systems that can maintain context, reason about tasks, invoke external capabilities, and coordinate actions [1]. In enterprise environments, these capabilities must operate within existing software, data, security, governance, and operational structures. An agentic system may need to access organizational knowledge, invoke one or more models, maintain execution state, interact with

enterprise services, and operate under identity, authorization, safety, and operational controls. By integration we mean embedding agentic capabilities into this existing estate, including legacy systems, identity and access infrastructure, and governance, rather than building agents in isolation. Integrating agentic AI is therefore not only a model-selection or prompting problem, but also a software-architecture problem concerning how these heterogeneous capabilities are organized, connected, controlled, and operated.

Existing research and industry architectures provide substantial guidance for such systems. Enterprise-oriented work addresses concerns such as business alignment, workflow orchestration, organizational knowledge, governance, and enterprise integration [2], while industry architectures describe runtime concerns including model access, knowledge and memory, tools, security, state, observability, and multi-agent coordination [3]–[5]. Interoperability specifications further provide mechanisms for interactions with tools and contextual resources or between independent agents [6], [7]. These approaches address many capabilities relevant to enterprise agentic AI, but differ in scope, abstraction, responsibility organization, interaction representation, and assumptions about which capabilities are required in a particular system.

These differences create three related architecture-design problems. First, heterogeneous capabilities such as orchestration, model inference, enterprise context, external action, and access control require clear responsibility boundaries, even when their implementations are colocated or provided by the same platform. Second, dependencies among these responsibilities require explicit interaction semantics without implying that every system follows the same runtime pipeline. Third, capabilities such as multi-agent collaboration, persistent memory, asynchronous execution, human approval, or execution isolation may apply only under particular workload, integration, or governance conditions. A reference architecture for enterprise agentic AI should therefore distinguish architectural responsibilities, their interactions, condition-dependent capabilities, and the replaceable mechanisms through which they are realized. We investigate the following research questions:

- **RQ1:** What architectural capabilities, interactions, and

decision conditions are identified for integrating agentic AI into enterprise software systems?

- **RQ2:** How can these capabilities be organized into technology-neutral responsibility boundaries with explicit interaction semantics and variability?
- **RQ3:** How can the reference architecture be used to build a concrete enterprise AI system?
- **RQ4:** To what extent does the reference architecture hold when evaluated against concrete enterprise agentic AI systems?

To address these questions, we synthesize architectural capabilities from research publications, industry architecture guidance, and interoperability specifications into ten responsibilities, grouped into design context (L0), interaction and access (L1–L2), agentic execution and enterprise integration (L3–L6), and runtime and operational support (L7–L9). We then define boundary invariants, typed cross-responsibility interactions, and variation points that make responsibility separation, interaction semantics, and condition-dependent choices explicit. We instantiate the architecture as an ESG portfolio-analysis and pre-trade compliance agent built over an existing service-oriented ESG metric computation and reporting system, and complement the case with a mapping of an independently published agentic architecture. The contribution is therefore not a new agent technology stack, but an architectural organization in which responsibility ownership remains stable while execution paths, optional capabilities, protocols, and technologies vary.

II. BACKGROUND AND RELATED WORK

A. Agentic AI System Architecture

Agentic AI introduces multiple architectural concerns around model inference. Prior work characterizes agentic systems through capabilities including reasoning and planning, memory and state management, tool use, environmental interaction, and coordination of actions [1], [8]. Broader reviews additionally identify areas such as networking, trust and safety, and evaluation [9]. These concerns are interdependent, and systems range from a single agent to multiple specialized agents that communicate, delegate, or collaborate [1].

Enterprise deployment broadens this problem: agentic execution must coexist with existing information, services, and controls [8], [9].

B. Reference Architectures for Enterprise Agentic AI

Existing architectures address enterprise agentic AI from different scopes and levels of abstraction. Koo and Chung take an enterprise-transformation perspective, organizing agentic AI around Business Alignment, Workflow Orchestration, Agent Collaboration, Organizational Knowledge, Governance Integration, and Enterprise System Integration [2]. Industry architectures provide more implementation-oriented perspectives. AWS describes agents together with concerns including model access, tools, knowledge, security, observability, and discoverability [3], while Microsoft’s Multi-Agent Reference Architecture focuses on systems of collaborating agents and

addresses orchestration, agent discovery, memory and state, knowledge access, external tools, security, and observability [4]. Google Cloud similarly describes components and patterns for agentic applications across different agent, workload, and tenancy configurations [5], [10].

Enterprise-oriented work treats governance, knowledge, and integration at a high level, whereas implementation-oriented architectures decompose them into runtime services or controls [2]–[4], and application-specific architectures add domain-specific structure; DevCoach, for example, adds context-refinement loops, bounded tool use, validation, and audit artifacts for a software-engineering lifecycle [11]. At the level of an individual agent, Lu et al. propose a pattern-oriented reference architecture for foundation-model-based agents covering interaction engineering, memory, planning, execution, and responsible-AI plugins [12]; it addresses agent design rather than enterprise integration. Interoperability specifications standardize specific interactions: MCP specifies how AI applications access externally exposed tools and contextual resources [6], whereas A2A addresses interoperability between independent agents [7]. Such specifications complement architectural descriptions, but do not by themselves determine how the broader set of enterprise responsibilities should be organized.

Existing approaches also assume different configurations, from workload-specific patterns [5] to multi-agent concerns that may not arise for a single agent [4]; the related work is therefore substantial but fragmented. This motivates a technology-neutral synthesis in which responsibility boundaries, cross-responsibility interactions, and architectural variability are represented explicitly while implementation technologies remain replaceable realization choices.

III. RESEARCH METHOD

We used an architecture-synthesis process comprising four activities (Fig. 1): (1) identifying architectural evidence and normalizing candidate capabilities, (2) synthesizing the normalized capabilities into technology-neutral architectural responsibilities, (3) analyzing cross-responsibility interactions and conditions of architectural variability, and (4) instantiating and evaluating the architecture. The process follows empirically grounded reference-architecture design [13] and the ProSA-RA progression from information-source investigation through architectural design to evaluation [14]. The activities correspond to the research questions: evidence (RQ1), synthesis (RQ2), instantiation (RQ3), and evaluation (RQ4). The activities were iterative during architecture development; the responsibilities, invariants, and interaction types were fixed before the evaluation reported in Section VI.

A. Evidence Identification and Capability Normalization

We conducted a targeted architecture-oriented review to identify capabilities, interactions, and decision conditions relevant to enterprise agentic AI integration. It covered three source categories: research publications describing agentic-AI and enterprise-AI architectures, industry architecture guidance

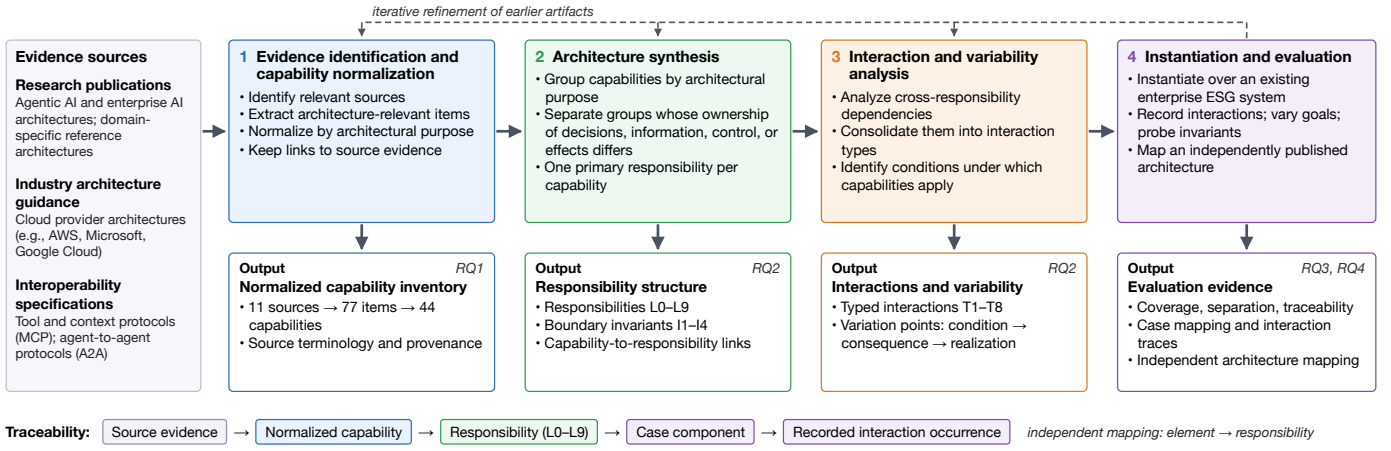


Fig. 1. Research method for developing and evaluating the reference architecture, following empirically grounded reference-architecture design [13] and ProSA-RA [14].

describing deployable enterprise patterns and components, and interoperability specifications defining interactions among agents, contextual resources, and executable capabilities; following multivocal review guidelines, the latter two were treated as grey literature [15]. Sources were considered up to September 2026. Search terms combined agentic AI, AI agents, and enterprise AI with reference architecture, orchestration, tools, memory, security, observability, and interoperability. Sources were retained when they described architectural capabilities, relationships, controls, or configuration decisions relevant to integrating agentic AI into enterprise software; sources concerned only with model-level algorithms or application outcomes were outside scope.

For each retained source, we extracted architecture-relevant statements and recorded the corresponding capability, architectural purpose, relationships with other capabilities, applicable conditions where stated, and the source terminology. Because sources describe related concerns using different terminology or levels of abstraction, extracted items were normalized by architectural purpose rather than by product or technology name. Closely related items were consolidated when they served the same purpose, while distinctions were retained when they represented different responsibilities or decision conditions. The review retained 11 sources, from which 77 architecture-relevant items were extracted and consolidated into 44 normalized capabilities, each linked to the evidence from which it was derived; this inventory served as the evidence base for synthesis and for the analysis of interactions and variability.

B. Architecture Synthesis

We synthesized the normalized capabilities by architectural purpose rather than by source-specific terminology, product, or implementation technology. Capabilities were grouped when they contributed to the same primary responsibility, while capabilities with distinct purposes or control concerns were kept separate. Responsibility boundaries were refined by examin-

ing the ownership of architectural decisions and information across capability groups: whether a group primarily determines execution, provides contextual information, performs model inference, controls access, produces external effects, supports communication, operates infrastructure, or evaluates system behavior. A separate responsibility was retained wherever merging two capability groups would conflate distinct ownership of architectural decisions, information, control, or effects; where such groups nonetheless interact closely, the distinction was expressed as a boundary invariant.

Implementation mechanisms were kept separate from the responsibility structure. Technologies, protocols, storage mechanisms, deployment platforms, and architectural patterns were treated as possible realizations of capabilities rather than as responsibilities, so that a need such as contextual retrieval does not prescribe a particular database, protocol, model provider, or deployment technology. The structure was checked against the inventory and refined until each retained capability had exactly one primary responsibility.

C. Interaction and Variability Analysis

We then analyzed dependencies that cross responsibility boundaries. For each dependency, we examined the purpose of the interaction, the information or intent transferred, the direction of exchange, and which responsibility retained ownership of the associated decision or operation. Interactions with similar semantics were consolidated into interaction types, describing contracts between responsibilities without imposing a fixed end-to-end execution sequence.

We separately analyzed the conditions under which capabilities or interaction mechanisms apply. A capability was treated as a variation point [16] when its inclusion or realization depended on characteristics such as agent topology, execution duration, integration requirements, external-effect risk, or operational constraints. For each variation point, we recorded the motivating condition, its architectural consequence, and representative realization mechanisms.

D. Instantiation and Evaluation

We evaluated the architecture through two complementary analyses, following the case-based evaluation common for reference architectures [17]. First, we implemented an instantiation as a deployed agent over an existing service-oriented ESG metric computation and reporting system and recorded its runtime interactions. We executed it against a representative portfolio question, portfolio and trade questions that differ in what they require, and probes of the boundary invariants. Second, we mapped an independently published agentic architecture that is not among the synthesis sources to the proposed responsibilities, to examine whether architectural elements defined outside the case can be represented using the same boundaries.

To answer RQ4, the evaluation considered three dimensions, one for each claim the architecture makes: that it can represent what systems require, that its boundaries are enforceable rather than nominal, and that its elements rest on evidence rather than on the case alone. *Architectural coverage* examines whether the capabilities and elements required by each evaluated configuration can be represented by the proposed responsibilities and interactions. *Responsibility separation* examines whether observed behavior is consistent with the ownership distinctions expressed by the boundary invariants. *Traceability* examines whether architectural elements can be followed from source evidence and normalized capabilities to responsibilities and, for the concrete case, to implementation components and runtime interactions.

IV. ENTERPRISE AGENTIC AI INTEGRATION REFERENCE ARCHITECTURE

Addressing RQ1 and RQ2, the synthesis produced a reference architecture of ten responsibilities, denoted L0–L9. Each groups capabilities that share a primary architectural purpose and ownership concern: Business & Solution Design (L0), Client & Experience (L1), Access, Identity & Safety Control (L2), Agent & Workflow Orchestration (L3), Model Access & Inference (L4), Enterprise Context & Knowledge (L5), Tool & Action Runtime (L6), Async & Event Infrastructure (L7), Platform & Delivery Infrastructure (L8), and Evaluation, Observability & Improvement (L9). Fig. 2 presents this structure and its principal relationships.

L0–L9 are responsibility boundaries, not a processing pipeline: L0 sets design-time requirements, L8 hosts the runtime responsibilities, L9 receives their evidence, and during execution L3 selectively coordinates L4–L6, with L7 mediating asynchronous communication.

A. Boundary Invariants

The responsibility structure alone does not prevent closely related concerns from being merged or assigned ambiguously. We therefore define four boundary invariants: ownership distinctions that should hold across implementations. They constrain architectural responsibility rather than physical deployment: components implementing different responsibilities may be colocated, provided by the same platform, or

deployed independently. The separation is logical; it becomes an enforceable security boundary only where credentials, privileges, and failure domains are also separated, which colocated components need not have.

I1—Coordination–Execution Separation. L3 owns execution coordination, including control flow and the decision to request an external capability, whereas L6 owns the authorization and execution of that capability against the target resource. L3 can therefore determine that an action should occur without assuming responsibility for producing the external side effect.

I2—Execution-State–Knowledge Separation. L3 owns information whose purpose and lifecycle are bound to coordinating an active execution, including execution state, intermediate results, and checkpoints. L5 owns information intended to be independently indexed or retrieved as reusable context across executions, including persistent enterprise knowledge and cross-execution memory.

I3—System-Access–Resource-Authority Separation. L2 establishes the principal, tenant, system-level access, delegated authority, and security context associated with a request. Responsibilities that own protected resources retain enforcement appropriate to those resources: L5 applies access and tenant constraints during retrieval, L6 enforces resource- and action-specific authorization and credential binding, and network reach is bounded by L2 egress control and L6 isolation. Requesting an action is distinct from being authorized to perform it: where execution is delegated to another agent or deferred, propagated authority should be scoped and time-limited, and L6 re-authorizes at execution time, because the originating context may have changed.

I4—Evidence-Ownership–Correlation Separation. Responsibilities L1–L8 remain authoritative producers of evidence about their local operations, while L9 correlates that evidence for end-to-end observation, evaluation, diagnosis, and improvement.

B. Design Context, Interaction, and Access

L0—Business & Solution Design (Decide). L0 translates business objectives and enterprise constraints into requirements for the agentic AI system. Its capabilities cover use cases and expected value, requirements, architectural principles and system boundaries, delivery and adoption, and governance across agents, models, prompts, knowledge, tools, and data. It also establishes ownership, accountability, risk, and regulatory requirements that constrain decisions elsewhere, rather than acting as a runtime step.

L1—Client & Experience (Interact). L1 provides the interaction surfaces through which users and software clients access agentic capabilities, such as web applications, enterprise communication channels, and programmatic clients. L1 owns interaction and presentation, not agent reasoning or coordination.

L2—Access, Identity & Safety Control (Protect). L2 protects the controlled system boundary and establishes the identity, authority, tenant, and policy context associated with a request. Its capabilities include ingress and traffic control,

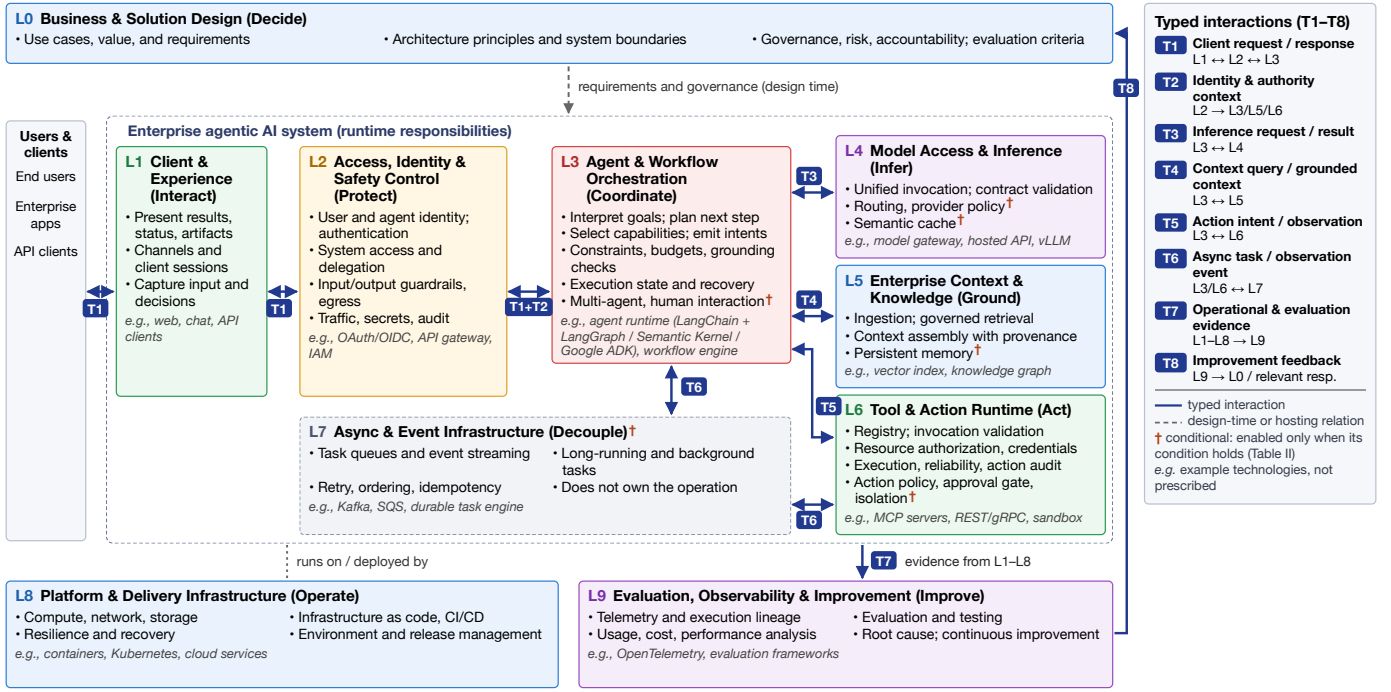


Fig. 2. Enterprise Agentic AI Integration Reference Architecture. L0–L9 define architectural responsibility boundaries rather than a sequential processing pipeline; the example technologies are illustrative, not prescribed.

user and agent identity and authentication, system-level access authorization and delegation context, input and data protection, boundary-level safety controls, egress and data control, secrets management, and access auditing. L2 does not centralize all authorization or safety decisions: authority established at L2 is propagated to responsibilities that own protected resources, which retain enforcement specific to those resources (I3).

C. Agent Execution and Enterprise Integration

L3–L6 are most directly involved in goal-directed execution. Task allocation between model and conventional software is explicit: model-driven behavior is confined to selecting the next step (L3) and performing inference (L4), while access control, context retrieval, action execution, and every control are conventional software. Their separation allows reasoning, contextual grounding, and external effects to evolve independently while retaining explicit ownership at their boundaries.

L3—Agent & Workflow Orchestration (Coordinate). L3 controls goal-directed execution. Its capabilities include agent runtime and control flow, execution constraints, in-flight execution context, prompt and runtime configuration, and execution state, recovery, and continuity; depending on requirements, it can additionally support collaboration among multiple agents and human interaction or intervention. It owns coordination decisions but not model inference, persistent enterprise knowledge, or execution of external side effects. A checkpoint can be durably stored without becoming enterprise knowledge; information meant to be reused across executions belongs to L5 (I2).

L4—Model Access & Inference (Infer). L4 provides controlled access to model inference while isolating execution logic from particular model providers. Its capabilities include unified model access, model selection and routing, provider policy and capacity controls, model invocation, and inference-contract validation; semantic caching can be introduced when reuse is beneficial. L4 determines how an inference request is routed and executed, while L3 decides why inference is required and how its result affects execution.

L5—Enterprise Context & Knowledge (Ground). L5 provides persistent enterprise information and governed context for agentic execution. Its capabilities span information sources, ingestion and preparation, enterprise context stores and persistent memory, retrieval and context assembly, and data governance. L5 owns the retrieval and assembly of persistent context, while L3 determines how that context is used; retrieval-augmented generation is therefore one possible realization of this relationship rather than the definition of the responsibility.

L6—Tool & Action Runtime (Act). L6 resolves, validates, authorizes, executes, and records invocations of capabilities that interact with enterprise systems or produce effects outside agent coordination. Its capabilities include tool and capability registration, invocation validation and adaptation, resource-specific authorization and credential binding, tool integration and execution, reliability and outcome handling, and action audit and provenance. Where an external effect requires additional control, L6 can apply an action-policy or approval gate before execution. Under I1, L3 supplies the intent and execution context, and L6 determines whether and how the

capability can be executed against the target resource and returns the observation, which L3 treats as untrusted input rather than instruction.

Guardrails and verifiers [18] are not a separate responsibility; each is placed where the checked decision is owned. Input and output guardrails sit at the L2 boundary, retrieval constraints in L5, structural checks of model output in L4, and grounding and execution-constraint checks in L3. Pre-action verification of consequential effects sits in L6, whose action-policy or approval gate permits, refuses, or escalates an action, and L9 monitors from outside the execution loop. Treating retrieved context and tool results as untrusted input, together with the L3 grounding checks, addresses prompt injection through returned content.

D. Runtime and Operational Support

L7—Async & Event Infrastructure (Decouple). L7 supports asynchronous, event-driven, and long-running communication through queues, event streaming, and delivery-reliability mechanisms such as retry, ordering, and idempotency. L7 is introduced when execution calls for temporal decoupling, background processing, durable delivery, or event-driven coordination, and changes delivery semantics without taking ownership of the operation communicated.

L8—Platform & Delivery Infrastructure (Operate). L8 provides the compute, connectivity, storage, resilience, and delivery environment (infrastructure as code, CI/CD, release management) on which the other responsibilities operate. L8 owns platform operation rather than agent behavior: infrastructure autoscaling belongs to L8, whereas terminating a task that exceeds its budget belongs to L3.

L9—Evaluation, Observability & Improvement (Improve). L9 correlates operational and evaluation evidence across the architecture to assess system behavior and support diagnosis and improvement. Its capabilities include telemetry, execution lineage, evaluation and testing, usage and cost analysis, root-cause analysis, and continuous improvement. Under I4, L1–L8 remain authoritative for events produced within their boundaries, while L9 links and analyzes this evidence across an execution.

E. Typed Interactions and Variability

Responsibility separation alone does not specify how architectural concerns cooperate during execution. We therefore characterize recurring cross-responsibility dependencies as typed interactions (Table I). An interaction type identifies the architectural purpose of an exchange and the responsibilities involved without prescribing a transport protocol, deployment topology, or fixed execution sequence. MCP can realize selected interactions with contextual resources or executable capabilities, while A2A can realize communication with agents coordinated by L3 or, for agents owned by another party, reached through L6; REST, gRPC, or messaging provide alternative realizations.

We represent a variation point through the condition that motivates an architectural choice, the resulting architectural

consequence, and representative mechanisms through which that consequence can be realized (Table II). Variability can therefore alter which capabilities are enabled and how responsibilities interact without redefining the responsibilities themselves. Synchronous and asynchronous configurations preserve the same L3 coordination and L6 action ownership while differing in whether L7 mediates their interaction. A single-agent configuration omits L3 collaboration capabilities without removing the L3 responsibility. To apply the architecture, an architect identifies the responsibilities a target system requires, enables condition-dependent capabilities according to its workload, integration, and governance requirements, selects realization mechanisms for the required interactions, and places controls according to the invariants. The responsibility boundaries and invariants remain stable while these choices vary.

V. REFERENCE ARCHITECTURE INSTANTIATION

Addressing RQ3, we instantiate the reference architecture as an agent for portfolio carbon analysis and pre-trade checking against an illustrative ESG fund mandate, built over an existing ontology-driven ESG metric computation and reporting system [19]. The case was selected because it involves concerns central to the architecture: enterprise knowledge retrieval, deterministic computation, model-driven coordination, role-based authorization, policy-controlled external action, and condition-dependent approval. The portfolio metric is the weighted average carbon intensity (WACI), the weight-averaged ratio of each holding’s Scope 1 and Scope 2 emissions to its revenue [20]. The existing system computes company-level metrics through services and represents metric definitions in a knowledge graph, queried through competency questions (CQ1–CQ5) that return, for example, a category’s metrics and how a metric is calculated. Section V-A derives the agent’s design by applying the steps of Section IV-E; Sections V-B and V-C run it.

A. Applying the Architecture

Required responsibilities. The use case, the mandate limits, and the rule that a second person approves overrides are L0 decisions that set the requirements. At runtime, the agent must accept questions and approvals (L1), authenticate users and assign entitlements (L2), coordinate the steps of a question (L3), obtain model inference (L4), retrieve metric definitions (L5), and compute metrics, analyze portfolios, and act on trades (L6), all on a deployment platform (L8). Table III lists the components. The knowledge graph (L5) and metric computation (L6) are reused from the existing system. A portfolio and compliance service added to L6 computes WACI and checks a trade deterministically against the fund’s mandate, an illustrative mandate limiting portfolio intensity, exposure to high-intensity holdings, and single-holding concentration. The service holds the mandate; the request cannot change it. L3 runs a tool-use loop in which a language model chooses each next step among the capabilities of L5 and L6, or answers. L4 validates the inference contract of every model call: a

TABLE I
TYPED CROSS-RESPONSIBILITY INTERACTIONS

Type	Interaction	Responsibilities	Architectural semantics
T1	Client Request / Response	L1 $\leftrightarrow$ L2 $\leftrightarrow$ L3	Carries client intent through the controlled boundary at L2 to L3 and returns responses, status, or artifacts through L2 egress control, separating interaction concerns from execution coordination.
T2	Identity & Authority Context	L2 $\rightarrow$ L3/L5/L6	Propagates principal, tenant, entitlement, delegated-authority, policy, and trace context; receiving responsibilities retain resource-specific enforcement.
T3	Inference Request / Result	L3 $\leftrightarrow$ L4	Requests model capabilities under specified constraints and returns model output, validation status, model metadata, and usage information.
T4	Context Query / Grounded Context	L3 $\leftrightarrow$ L5	Expresses an information need and returns governed context, evidence, metadata, and provenance appropriate to the execution.
T5	Action Intent / Observation	L3 $\leftrightarrow$ L6	Requests execution of an external capability and returns its result, status, error, state-change evidence, or provenance.
T6	Async Task / Observation Event	L3/L6 $\leftrightarrow$ L7	Mediates tasks and observations through asynchronous or event-driven delivery while preserving ownership in the originating and receiving responsibilities.
T7	Operational & Evaluation Evidence	L1–L8 $\rightarrow$ L9	Provides locally produced traces, events, outcomes, usage, cost, audit information, and provenance for correlation and evaluation.
T8	Improvement Feedback	L9 $\rightarrow$ L0 / relevant responsibilities	Returns evaluation findings, failure evidence, and quality, cost, or performance observations to inform governed architectural or configuration changes.

TABLE II
REPRESENTATIVE ARCHITECTURAL VARIATION POINTS

Condition	Architectural consequence	Representative realization
Multiple collaborating agents	Enable L3 agent collaboration, discovery, and coordination interactions	Agent registry; A2A or direct calls
Deterministic execution constraints	Constrain allowable execution transitions in L3	State machine, DAG, or workflow rules
Human participation required	Enable L3 human interaction and intervention	Clarification, review, pause/resume
Protected or consequential external effect	Enable an L6 action-policy or approval gate	Policy service or human approval
Untrusted or high-risk execution	Isolate execution of the relevant L6 capability	Sandbox or restricted runtime
Delegated or deferred execution	Scope and expire propagated authority (T2); L6 re-authorizes at execution time	Scoped, short-lived tokens; re-authorization
Multi-turn or resumable sessions	Durably store L3 execution state; cross-session reuse belongs to L5 (I2)	Checkpoint or session store
Persistent cross-execution context required	Enable L5 persistent memory and governed retrieval	Vector, graph, document, or structured store
Multiple models or provider constraints	Enable L4 model routing and provider policy	Model gateway or router
Semantic reuse beneficial	Enable L4 semantic caching	Semantic cache
Long-running or background execution	Mediate relevant interactions through L7	Queue or durable task mechanism
Event-driven coordination required	Enable L7 event infrastructure	Publish/subscribe or event stream
Recovery from multi-step state-changing operations	Coordinate recovery in L3 and compensating actions through L6	Saga or compensation pattern
Standardized tool or context interoperability desired	Use standardized realization of selected L5/L6 interactions	MCP
Direct synchronous interaction sufficient	Bypass L7 for the relevant interaction	Direct API, RPC, or equivalent call

response must call a registered tool or answer; refusals and truncated outputs count as failures. Only this step selection and inference are model-driven; all other components are conventional software.

Conditional capabilities. Submitting a trade is a consequential external effect, so we enabled the L6 action policy and approval gate (Table II). On submission, L6 re-runs the mandate check and clears a compliant trade. For a breach with a requested override, it holds the trade until a principal who holds the override entitlement and is not the submitter approves it. No order management system is connected, so a cleared trade is recorded rather than executed. The other conditional capabilities were not enabled because their conditions are absent: requests complete synchronously in under a minute (no L7), one agent suffices (no L3 collaboration), the invoked services are trusted first-party services (no L6 isolation), and each question is answered independently (no L5 persistent memory). An L9 service for cross-execution correlation lies

outside the evaluated deployment.

Realization mechanisms. L5 and L6 are exposed through MCP [6], the realization in Table II for standardized tool and context interoperability; L4 uses a hosted language model; all components run as separate containers.

Control placement. The invariants determined where each control resides. Under I1, the model selects capabilities but does not execute them: the holdings and intensities passed to the portfolio and compliance service are assembled by L3 from L6 results, never supplied by the model, and L6, not L3, decides whether a trade is cleared. Under I3, L2 assigns entitlements by role (a portfolio manager may compute metrics, analyze portfolios, and submit trades; a compliance officer may approve overrides), and L6 enforces them at each invocation. Under I2, L3 retains only execution state, and definitions are retrieved from L5 for each request. Under I4, each responsibility records its own exchanges, carrying a trace identifier assigned at L2. L3 also bounds execution: the model

TABLE III
CASE COMPONENTS AND THE RESPONSIBILITIES THEY REALIZE; *
EXISTING ESG SYSTEM.

Resp.	Case component
L1	Web service and portfolio analyst page
L2	Access gate: signed session, role-based entitlements (portfolio manager, compliance officer), request validation, rate limit, egress check
L3	Agent runtime: model-driven tool-use loop with execution constraints and number check
L4	Model access: hosted language model, inference-contract validation
L5	ESG knowledge graph* (competency questions), via MCP
L6	Metric-computation service*, via MCP
L6	Portfolio and compliance service (WACI, pre-trade mandate check), via MCP; entitlement checks; trade submission and override gate
L8	Container deployment of all components

may act only on the request’s fund, holdings, and years; the number of model calls is bounded; and every number in the final answer must occur in the request or in a result returned by L5 or L6. A draft answer that fails this *number check* is returned to the model once for revision.

B. Worked Example

In question G3 (Table IV), a portfolio manager holds four semiconductor companies (STMicroelectronics 30%, Infineon 25%, Micron 25%, TSMC 20%) and asks: *What is the carbon intensity of my portfolio in 2023, and which holdings drive it?* Fig. 3 shows the execution of this design as recorded. We count an *interaction occurrence* as one directed exchange conforming to an interaction type of Table I, a request and its return separately; numbers in parentheses below refer to the numbered occurrences in Fig. 3.

The request is admitted at L2 and passed to L3 with identity, role, and trace context (1–2). The model asks which metrics the *Greenhouse Gas Emissions* category contains (5–6) and retrieves the definition of *GHG Emission Intensity*, a calculation model (*Scope 1 + Scope 2*)/*Revenue* (9–10). It then requests the metric for each holding (13–20). The computation service returns 45.47, 28.99, and 439.94 t CO₂e per USD million for STMicroelectronics, Infineon, and Micron, and reports that TSMC’s revenue is unavailable. The model requests the portfolio figure (23–24): WACI, computed over the 80% of the portfolio with data and re-weighted to it, is 163.59, of which Micron contributes 84%. The answer, stating this figure, the drivers, and the excluded holding with its reason, passed the number check and returned to L1 (27). The execution comprises 27 occurrences realizing five of the eight interaction types, T1–T5.

C. Goal-Conditioned Variation and Conditional Capabilities

To examine whether the responsibilities act as boundaries rather than as stages of a fixed pipeline, and whether the conditional capabilities activate only when their conditions hold, we executed the same deployment against seven portfolio and trade questions that differ in what they require (Table IV).

TABLE IV
GOAL-CONDITIONED VARIATION OVER SEVEN QUESTIONS (THREE
EXECUTIONS EACH; OCC. RANGES REFLECT THE VARIATION DESCRIBED
IN THE TEXT).

Question	Occ.	L6 calls	Outcome
G1 Intensity and drivers (4 holdings, all with data)	27	5	Answered
G2 Change since 2022	39–41	10	Answered
G3 As G1, one holding lacking revenue (worked example)	27	5	Answered; gap reported
G4 Intensity and drivers (7 holdings)	35	8	Answered
G5 Trade within the mandate; submit	31–33	6–7	Cleared
G6 Trade breaching the mandate; override	33	7	Held for override
G7 Share prices	5	0	Declined

Because the model’s choices are not deterministic, each question was executed three times.

The seven questions produced goal-dependent execution paths with 5 to 41 interaction occurrences. Comparing years (G2) or adding holdings (G4) multiplied the computations. A trade within the mandate (G5) activated L6 policy enforcement and was cleared, while a trade breaching two limits (G6) additionally activated the approval gate. A question about share prices (G7), which the data cannot answer, was declined after a single model decision without engaging L5 or L6. Questions that only read data bypassed both controls. These conditions altered which capabilities were invoked and which controls became active, while responsibility ownership remained unchanged. Within a question, executions differed only in that one G5 execution also computed the current WACI and that four drafts (one for G2, three for G4) with model-derived figures were returned once by the number check.

VI. EVALUATION

Addressing RQ4, we evaluate the instantiation of Section V and a mapping of an independently published architecture along the three dimensions of Section III. Each checks a part of the design: whether the required responsibilities fit within L0–L9 (coverage), whether the controls placed under the invariants hold when probed (separation), and whether design decisions can be followed from source evidence to runtime behavior (traceability). Responsibilities are thus assessed through coverage and separation, interactions through the recorded occurrences, and variation points through goal-conditioned variation (Section V-C).

A. Architectural Coverage

Case components were assigned to seven of the ten responsibilities (Table III). Each had one primary responsibility, and none required a responsibility outside L0–L9. The seven instantiated responsibilities contain 33 of the 44 normalized capabilities of Section III; this count is descriptive rather than a completeness score, because capabilities are condition-dependent. The executions realized five of the eight interaction types (T1–T5), and the two enabled conditional capabilities

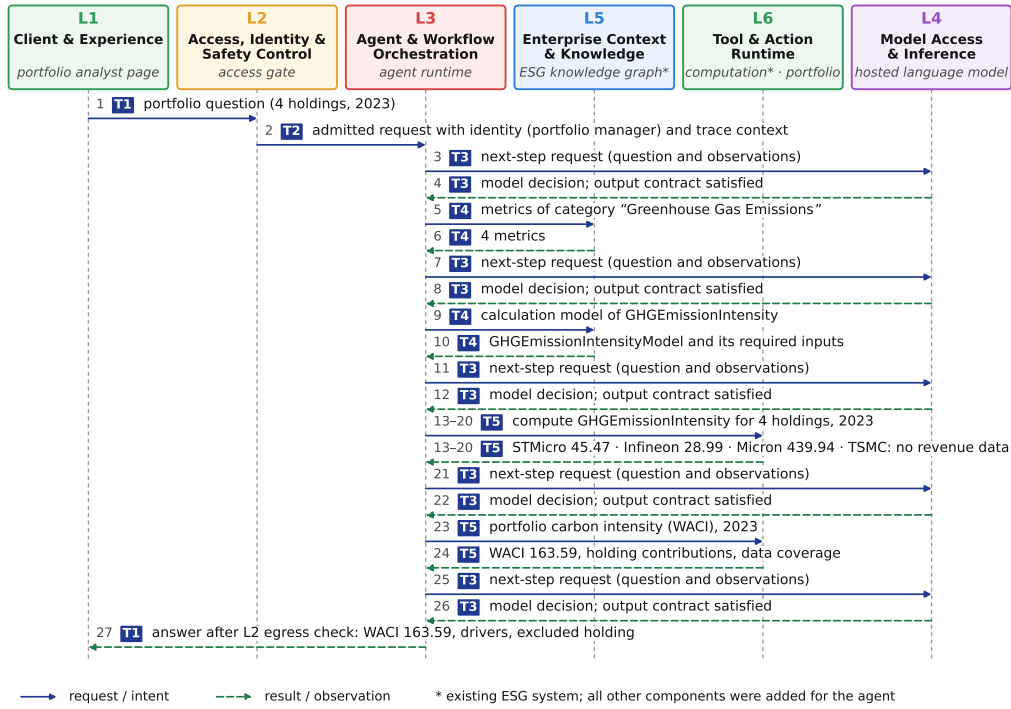


Fig. 3. Recorded execution of the worked example. Each lifeline is a responsibility and the case component realizing it; components marked * belong to the existing ESG system. Each message is labeled with its occurrence number, which the text refers to, its interaction type from Table I, and the content exchanged; the four per-holding computations (13–20) are drawn as one request and one result.

activated only when their conditions held (Section V-C). The independent mapping (Section VI-C) touches L0–L6 and L9. Across the two analyses nine of the ten responsibilities are represented; L7 appears in neither, because neither configuration is asynchronous.

B. Responsibility Separation

Each boundary invariant of Section IV-A was probed with operations that would expose a violation if its boundary were not enforced (Table V). All probe operations produced outcomes consistent with the invariants. For I4 the case shows evidence ownership and a shared correlation identifier but has no L9 correlation service, so I4 is assessed as consistent rather than preserved.

C. Independent Architecture Mapping

The mapping serves two dimensions: whether an architecture designed without reference to the proposed one can be represented within L0–L9 (coverage), and whether its elements that cross the proposed boundaries divide where the invariants predict (separation). We mapped Lu et al.’s reference architecture for foundation-model-based agents [12], which is not among the synthesis sources. Its mapped elements are the component groups of its architecture figure, with responsible-AI plugins grouped by concern. Elements were interpreted by their stated purpose, and those crossing a boundary were recorded as spanning. All eleven elements correspond to responsibilities within L0–L9, and none required an additional

responsibility: four map directly, one is a conditional capability the case does not instantiate, and six span a boundary (Table VI).

The spanning elements are the informative cases, because the source organizes components by agent function and the proposed structure by ownership. Its memory separates in-context working state from long-term knowledge retrieved into it, consistent with I2. Its execution engine both decides and performs tasks, which the proposed structure separates into coordination in L3 and capability execution in L6, consistent with I1. Its guardrails act at the user boundary, during retrieval, and around tool execution, consistent with resource-specific enforcement under I3. L7 and L8 have no counterpart, as the source addresses agent design rather than deployment. These distinctions arise independently of the case and the synthesis, but do not show that the boundaries are universal.

D. Traceability

For the case, each architectural element can be followed along the chain source evidence → normalized capability → responsibility → case component (Table III) → recorded interaction occurrence (Fig. 3); the independent mapping adds the complementary link from external architecture element to responsibility (Table VI). The deployment, experiment scripts, recorded results, and the generator for Fig. 3 are available as a replication package.¹

¹<https://github.com/Inspiring-Ming/AI-Agent-for-ESG>

TABLE V
BOUNDARY-INVARIANT PROBES AND OBSERVED OUTCOMES.

Invariant	Probe	Observed outcome	Result
I1 Coordination–Execution	Submit question G1; count invocations of the L6 computation and portfolio services; compare the reported intensities with L6 results; inspect L3 for an execution mechanism.	One computation per holding and one portfolio calculation, each invoked through L6; reported intensities equal the L6 results; L3 exposes no execution mechanism.	Preserved
I2 Execution-State–Knowledge	Inspect the state L3 retains after the request; submit a second request and count knowledge-graph retrievals.	L3 retained only execution-local state (question, plan, computations, results, status); definitions and provenance came from L5 through CQ1–CQ5, and the second request retrieved them again.	Preserved
I3 System-Access–Resource-Authority	Admit a principal at L2 without the computation entitlement and invoke L6; submit a request with a forged session token; submit a trade breaching the mandate and approve an override as the portfolio manager, as the submitter holding the override entitlement, and as the compliance officer.	The admitted principal was refused at L6; the forged request was refused at L2 before coordination; the breaching trade was held, the override was refused for the portfolio manager and for the submitter, and approved for the compliance officer.	Preserved
I4 Evidence–Ownership–Correlation	Inspect the recorded occurrences for their emitting responsibility and correlation identifier.	Occurrences were emitted by four responsibilities, and the identifier established at L2 was carried into the L5 requests.	Consistent

TABLE VI
MAPPING OF AN INDEPENDENTLY PUBLISHED AGENTIC ARCHITECTURE [12] TO THE PROPOSED RESPONSIBILITIES.

Source element	Source-described purpose	Resp.	Observation
Context engineering	Dialogue interface; goal creators interpreting the user’s goals	L1/L3	Spanning: interaction L1, goal interpretation L3
Prompt/response engineering	Prompt and response generation from templates and personas	L3	Prompt and runtime configuration
Memory	Short-term (in context window) and long-term memory	L3/L5	Spanning: consistent with I2
Planning	Plan generation and reflection, including human reflection	L3	Coordination
Cooperation	Voting-, role-, and debate-based cooperation among agents	L3	Conditional; the case uses one agent
Execution engine	Task executor using tools or agents; task monitor	L3/L6	Spanning: consistent with I1
Tool/agent selector	Find or generate tools and agents via a registry	L3/L6	Spanning: agent discovery L3, tools L6
Guardrails	Input, output, RAG, and execution guardrails	L2/L5/L6	Spanning: consistent with I3
Recording, risk assessment	Black box recorder, continuous risk assessor, explainer	L9	Accountability and operational evidence
AIBOM, co-versioning	Provenance of tools, agents, models; model variants	L0/L4	Spanning: provenance L0, model variants L4
AI models	External, fine-tuned, or sovereign foundation models	L4	Model access and selection

VII. DISCUSSION

A. Architectural Implications and Trade-offs

Separating orchestration from model access, context, and action lets these evolve independently but requires explicit interfaces and consistent propagation of execution and security context; separating system access from resource-specific enforcement supports least privilege, as the case’s override rule illustrates. The case also shows that model-driven coordination need not transfer authority to the model. The model acts as an adaptive planner within an execution harness in which deterministic services, verifiers, and policy controls retain authority over consequential operations [18]. Independent verification therefore constrains action selection without requiring the model itself to be trustworthy.

Separation has costs and limits. Physical separation adds interfaces, latency, and distributed failure modes; the worked example needs 27 exchanges for one question. Responsibilities should be physically separated where they require distinct credentials, trust, scaling, or change cadence, as L4 providers and L6 actions typically do. Otherwise they may be colocated, provided ownership and enforcement points remain distinct. Separation also does not provide confidentiality by itself: context sent to a hosted model, persistent memory, and telemetry are disclosure paths, so retrieval authorization (L5), provider

data policy (L4), and retention (L9) need coordinated policy set at L0.

B. Limitations

The architecture rests on a targeted review and on researcher judgment in normalizing concepts. The instantiation is a research prototype of an enterprise integration scenario rather than a production deployment. It covers one domain and one existing system: semiconductor holdings for which emissions and revenue data are available, an illustrative fund mandate, and one hosted language model whose choices are not deterministic, with consistency reported over three executions per question. Because the instantiation was designed around the proposed boundaries, the probes show that the boundaries are enforceable and consistent, not that they are preferable to alternative decompositions. Section III-B argues that they reduce ambiguity of ownership, allow realizations to be substituted, and localize assurance, but comparative evaluation remains future work. The independent mapping interprets one source, and the evaluation does not address effects on quality attributes such as performance, maintainability, security, or cost.

VIII. CONCLUSION

We presented a technology-neutral reference architecture for enterprise agentic AI integration comprising ten responsibilities, four boundary invariants, eight typed interactions,

and explicit variation points that separate stable responsibility ownership from condition-dependent capabilities and replaceable mechanisms. Instantiated as an ESG portfolio-analysis and pre-trade compliance agent over an existing ESG system, it realized seven responsibilities; different task and action conditions altered runtime interactions and activated conditional controls, consistent with the evaluated boundaries. An independently published agentic architecture was representable within L0–L9 without an additional responsibility. Future work will address further domains, multi-agent and asynchronous configurations, monitoring-driven runtime intervention, and effects on quality attributes.

REFERENCES

- [1] M. Abou Ali, F. Dornaika, and J. Charafeddine, “Agentic AI: A comprehensive survey of architectures, applications, and future directions,” *Artificial Intelligence Review*, vol. 59, no. 1, p. 11, 2025, article 11; published online 14 November 2025.
- [2] G.-D. Koo and M. Chung, “Reference architecture for enterprise AI transformation using agentic AI: A design science approach,” *The Journal of Korean Institute of Communications and Information Sciences*, vol. 51, no. 8, pp. 1588–1609, 2026.
- [3] Amazon Web Services, “Agentic AI architecture in the enterprise,” AWS Prescriptive Guidance: Governing and Architecting the Diversity of Agentic AI at Scale, 2026, accessed: 2026-09-21. [Online]. Available: <https://docs.aws.amazon.com/prescriptive-guidance/latest/govern-architect-agentic-ai/enterprise-architecture.html>
- [4] Microsoft, “Multi-agent reference architecture,” GitHub Pages, 2026, accessed: 2026-09-21. [Online]. Available: <https://microsoft.github.io/multi-agent-reference-architecture/>
- [5] Google Cloud, “Choose your agentic AI architecture components,” Google Cloud Architecture Center, 2026, last reviewed 21 April 2026. Accessed: 2026-09-21. [Online]. Available: <https://docs.cloud.google.com/architecture/choose-agentic-ai-architecture-components>
- [6] Model Context Protocol Project, “Model context protocol specification, version 2026-07-28,” Agentic AI Foundation, The Linux Foundation, Specification 2026-07-28, Jul. 2026, originally developed by Anthropic; contributed to the Linux Foundation’s Agentic AI Foundation, December 2025. Accessed: 2026-09-21. [Online]. Available: <https://modelcontextprotocol.io/specification/2026-07-28>
- [7] A2A Project, “Agent2agent (A2A) protocol specification, version 1.0.1,” The Linux Foundation, Specification v1.0.1, May 2026, originally developed by Google; hosted by the Linux Foundation. Accessed: 2026-09-21. [Online]. Available: <https://a2a-protocol.org/v1.0.1/specification/>
- [8] P. Peykani, S. Ghanidel, I. Javadi-Sisi, V. Snášel, and S. Mirjalili, “A holistic review of agentic AI frameworks, applications, and research trajectories,” *Archives of Computational Methods in Engineering*, 2026, published online 15 June 2026.
- [9] C. Morales, L. A. L. F. da Costa, S. J. Rigo, R. Kunst, V. C. de Souza, É. P. Silva, G. L. E. Prado, T. d. C. ScharDOSim, and A. Roehrs, “A systematic literature review of agentic AI: Definitions, architectures, and challenges,” *IEEE Access*, vol. 14, pp. 36 176–36 190, 2026.
- [10] Google Cloud, “Multi-tenant agentic AI system,” Google Cloud Architecture Center, 2026, last reviewed 18 June 2026. Accessed: 2026-09-21. [Online]. Available: <https://docs.cloud.google.com/architecture/multi-tenant-agentic-ai-system>
- [11] O. Elsis, F. De Abreu Santos, and G. Melo, “Lifecycle-aware GenAI assistance with MCP via context refinement loops: A reference architecture,” in *Proceedings of the 34th ACM International Conference on the Foundations of Software Engineering (FSE Companion '26)*. Association for Computing Machinery, 2026, pp. 1614–1622.
- [12] Q. Lu, L. Zhu, X. Xu, Z. Xing, S. Harrer, and J. Whittle, “Towards responsible generative AI: A reference architecture for designing foundation model based agents,” in *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2024, pp. 119–126.
- [13] M. Galster and P. Avgeriou, “Empirically-grounded reference architectures: A proposal,” in *Proceedings of the Joint ACM SIGSOFT Conference on Quality of Software Architectures and ACM SIGSOFT Symposium on Architecting Critical Systems (QoSA-ISARCS)*. ACM, 2011, pp. 153–158.
- [14] E. Y. Nakagawa, M. Guessi, J. C. Maldonado, D. Feitosa, and F. Oquendo, “Consolidating a process for the design, representation, and evaluation of reference architectures,” in *2014 IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2014, pp. 143–152.
- [15] V. Garousi, M. Felderer, and M. V. Mäntylä, “Guidelines for including grey literature and conducting multivocal literature reviews in software engineering,” *Information and Software Technology*, vol. 106, pp. 101–121, 2019.
- [16] M. Galster, D. Weyns, D. Tofan, B. Michalik, and P. Avgeriou, “Variability in software systems—a systematic literature review,” *IEEE Transactions on Software Engineering*, vol. 40, no. 3, pp. 282–306, 2014.
- [17] S. S. Chand, B. A. Kumar, and E. Y. Nakagawa, “Design and evaluation of software reference architecture: A systematic mapping study,” *Applied Computing and Informatics*, pp. 1–17, 2026, ahead of print.
- [18] Q. Lu and Y. Bengio, “AI safety: Not optional, not later,” arXiv preprint arXiv:2609.10630, 2026. [Online]. Available: <https://arxiv.org/abs/2609.10630>
- [19] M. Yu, F. Rabhi, B. Suleiman, N. Liu, and Y. Li, “An ontology-driven service-oriented system for ESG metric computation and reporting,” in *2026 IEEE International Conference on Web Services (ICWS)*. IEEE, Jul. 2026, pp. 347–353.
- [20] Task Force on Climate-related Financial Disclosures, “Implementing the recommendations of the task force on climate-related financial disclosures,” Financial Stability Board, Tech. Rep., Oct. 2021, defines weighted average carbon intensity for asset owners and managers. Accessed: 2026-10-07. [Online]. Available: https://assets.bbhub.io/company/sites/60/2021/07/2021-TCFD-Implementing_Guidance.pdf