



AgentArcEval: An architecture evaluation method for foundation model based agents

Qinghua Lu ^a*, Dehai Zhao ^a, Yue Liu ^a, Hao Zhang ^a, Liming Zhu ^a, Xiwei Xu ^a, Angela Shi ^b, Tristan Tan ^b, Rick Kazman ^c

^a Data61, CSIRO, Sydney, Australia

^b Empathetic AI, Sydney, Australia

^c University of Hawaii, Honolulu, HI, USA

ARTICLE INFO

Keywords:

Foundation model
Large language model
LLM
Agent
Architecture
Evaluation
Responsible AI
AI safety

ABSTRACT

The emergence of foundation models (FMs) has enabled the development of highly capable and autonomous agents, unlocking new application opportunities across a wide range of domains. Evaluating the architecture of agents is particularly important as the architectural decisions significantly impact the quality attributes of agents given their unique characteristics, including compound architecture, autonomous and non-deterministic behaviour, and continuous evolution. However, these traditional methods fall short in addressing the evaluation needs of agent architecture due to the unique characteristics of these agents. Therefore, in this paper, we present AgentArcEval, a novel agent architecture evaluation method designed specially to address the complexities of FM-based agent architecture and its evaluation. Moreover, we present a catalogue of agent-specific general scenarios, which serves as a guide for generating concrete scenarios to design and evaluate the agent architecture. We demonstrate the usefulness of AgentArcEval and the catalogue through a case study on the architecture evaluation of a real-world tax copilot, named Luna.

1. Introduction

Foundation models (FMs) are large-scale pretrained models with tens of billions of parameters (Bommasani et al., 2021), which can be adapted to perform a wide variety of downstream tasks. However, FMs exhibit inherent limitations, particularly when facing complex tasks which require users to provide prompts at each step, which can be inefficient and prone to errors. An agent is an autonomous system that perceives its environment and takes actions to achieve goals on behalf of humans (International Organization for Standardization, 2022). The emergence of FMs has enabled the development of highly capable and autonomous agents that can perceive context, reason, plan, and execute workflows to accomplish human goals (Lu et al., 2024). Given their huge potential to enhance productivity, there has been a surge of interest in FM-based agents across various domains, such as Devin¹ and AI Scientist.² Throughout this paper, we use the term “agents” specifically to refer to FM-based agents.

Software architecture evaluation is a key practice for assessing the extent to which architectural design decisions meet quality attributes

and address their tradeoffs (Bass et al., 2021). Evaluating the architecture of agents is particularly important, as agents are complex compound AI systems that integrate FMs with a variety of out-of-model components, including context engines, prompt optimisers, reasoning and planning, workflow execution, memory, external knowledge bases and tools, guardrails, etc. Given these complexities, architectural decisions significantly impact quality attributes of agents, requiring careful balancing of trade-offs. Effective architectural evaluation helps identify potential risks, validate design decisions and improve the agent’s architecture to better meet its intended qualities. It also fosters a deeper understanding of how architectural decisions affect the agent’s ability to operate, learn and adapt within dynamic environments in a responsible and safe manner.

Over the past decades, various architecture evaluation methods have been developed, such as Architecture Tradeoff Analysis Method (ATAM) (Kazman et al., 2000). However, these traditional methods fall short when it comes to evaluating the agent architecture due to their unique characteristics. First, goals are special in the context of agents because the primary purpose of using agents is that their users only

* Corresponding author.

E-mail address: qinghua.lu@data61.csiro.au (Q. Lu).

¹ <https://www.cognition.ai/blog/introducing-devin>.

² <https://sakana.ai/ai-scientist>.

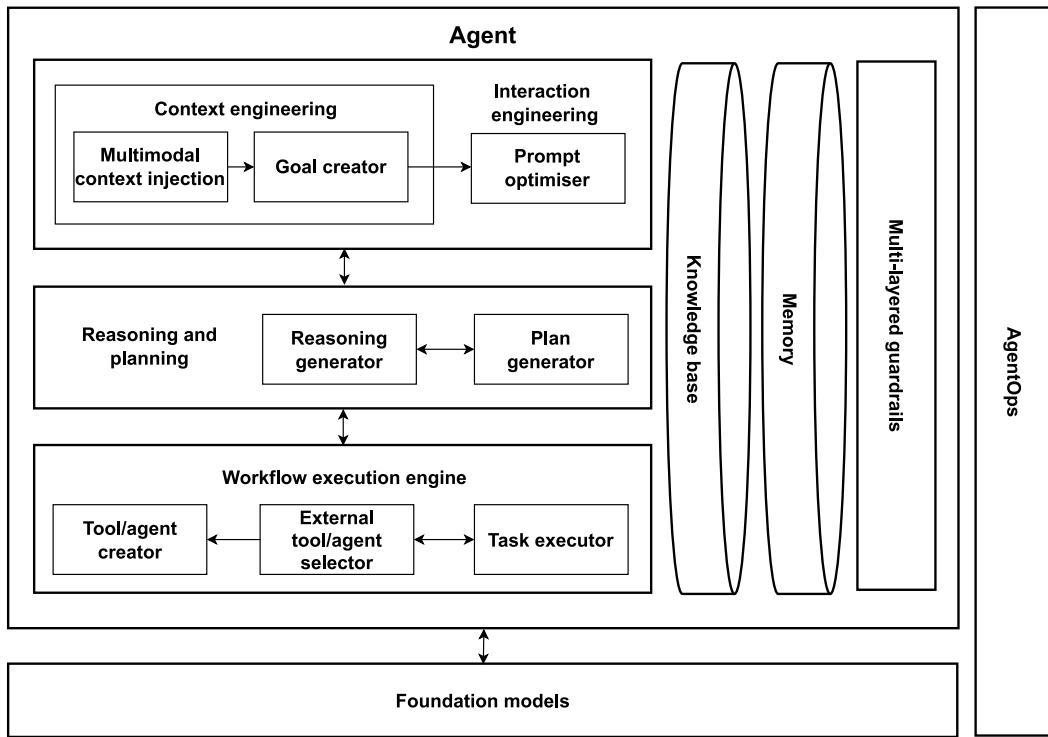


Fig. 1. Agent reference architecture (Lu et al., 2024).

need to provide high-level goals. The agents need to decompose these high-level user goals into open-ended fine-grained sub-tasks, generate system requirements dynamically, and design and assemble the system implementation at runtime, potentially generating a custom system structure for every user query. Each of the runtime architectural choices made by an agent may have direct implications for quality attributes. As a result, architecture evaluation must move beyond static structure analysis to consider high-level goals, exemplar context scenarios, and guardrails that guide these dynamic runtime trade-offs. The architecture should be assessed for its ability to surface and refine system requirements on the fly — such as goal decomposition, ambiguity resolution, uncertainty handling — rather than its ability to satisfy pre-defined requirements. Consequently, the evaluation focus shifts towards adaptability, observability, and transparency in goal translation, rather than simply functional coverage or satisfaction of stable quality attribute requirements. Second, agents can autonomously make user requirement-level tradeoff decisions (e.g., cost vs. convenience) within predefined guardrails, as long as they operate within those boundaries. Architects still need to consider the system requirement-level tradeoffs by designing the guardrail mechanism requirements, such as compute budget caps, trustworthiness scores, ethical constraints, and fall back behaviours. Thus, guardrail design becomes the principal architectural control point, replacing much of the fine-grained pre-specification that traditional architectures had to manage. Third, as agents continuously evolve, the previously assessed risks can change over time or emerge in unexpected ways. Compared to adaptive systems, agents can retrain, rewire workflows, and adjust external tool selection continuously based on fresh feedback. Also, the agency comes from FM's opaque internals, not explicit rules or well-understood machine learning (e.g., decision tree updates). Thus, runtime assurance mechanisms, behavioural monitoring, and periodic architecture re-evaluation are essential to detect when quality assumptions no longer hold. Lastly, there is a lack of agent-specific scenarios that capture the inter-artefact interactions, going beyond solely emphasising user-driven quality concerns.

Therefore, based on the classic ATAM method, we introduce **AgentArcEval**, an agent architecture evaluation method designed specifically to address the complexities of agent architecture and its evaluation. To support the agent architecture evaluation using AgentArcEval, we present a **catalogue of agent-specific general scenarios**, providing a structured guide for generating concrete scenarios to design and evaluate agent architecture. General scenarios have been shown to have great value in stimulating and guiding the creation of architectural requirements.

We demonstrate the usefulness of AgentArcEval and the catalogue through a case study evaluating the architecture of the Luna tax copilot.³ Please note that the proposed AgentArcEval method and the catalogue of general scenarios in this paper are also applicable to agentic systems, which are designed to prompt an FM multiple times using agent-like design patterns and exhibit varying degrees of agent-like behaviour.

The rest of the paper is organised as follows. Section 2 discusses the related work. Section 3 introduces the methodology. Section 4 presents the AgentArcEval method. Section 5 introduces the general scenario catalogue. Section 6 discusses the evaluation. Section 7 concludes the paper.

2. Background and related work

2.1. Background

There has been extensive research on the design of agents. In this work, we build on our previously proposed reference architecture for FM-based agents (Lu et al., 2024), which outlines a modular, layered view of agent components and their interactions. Fig. 1 illustrates a simplified version of the agent reference architecture. While goals are typically articulated by users and sent through prompts, some work explores proactive goal anticipation by analysing multimodal context

³ <https://empathetic-ai.com/home>.

information (Zhao et al., 2023; Zeng et al., 2023), such as screen recording, eye tracking, and gestures. Agent memory is structured into short-term memory and long-term memory. Short-term memory contains configurations, recent events, and working context (Packer et al., 2023; Jin et al., 2023; Zhao et al., 2024; Zhang et al., 2023). The capability of short-term memory is limited by the context window length of the FM. To extend the storage capacity of agent memory, long-term memory is adopted to store the entire history of processed events as well as accumulated knowledge and past experiences (Packer et al., 2023; Jin et al., 2023). Achieving a human goal needs reasoning and planning, such as Chain-of-Thought (Wei et al., 2022) and Tree of Thought (Yao et al., 2024). Plan reflection allows the agent refine the plan based on feedback (Park et al., 2023), which can be done through self-reflection (Yao et al., 2022; Madaan et al., 2024; Sumers et al., 2023; Shinn et al., 2023), cross-reflection (Chen and Chang, 2023; Shinn et al., 2023; Talebirad and Nadiri, 2023), or human reflection (Huang et al., 2022; Sarch et al., 2023). The execution engine is responsible for acting the plan, with a task monitor overseeing task status and managing queued tasks (Chen et al., 2023; Nascimento et al., 2023). The agent may rely on internal FM knowledge or external retrieval augmented generation (RAG) for information, cooperate with other agents or use external tools (Ruan et al., 2023; Kong et al., 2023) to execute the plans. The tool/agent selector (Xie et al., 2023; Talebirad and Nadiri, 2023) can search registries (Ruan et al., 2023; Wang et al., 2023; Kong et al., 2023) or on the web to find suitable tools and agents to task completion. Guardrails are designed to monitor and control the performance and behaviour of agents. Guardrails can apply to prompts, FMs, reasoning and planning, workflow execution, external tools and knowledge bases (Shamsujjoha et al., 2024), etc.

2.2. Related work

Software architecture evaluation plays a key role in ensuring that architectural decisions align with quality attributes and stakeholder needs. Bucaioni et al. recently proposed a high-level functional decomposition for systems that integrate large language models (Bucaioni et al., 2025). Their architecture emphasises modularity, interoperability, and quality attributes, and provides a general conceptual structure applicable to a wide range of LLM-based systems. In contrast, our reference architecture (Lu et al., 2024) targets agentic systems built on foundation models, where components such as context engineering, reasoning and planning, memory, and workflow execution are central. Moreover, our approach is pattern-oriented and explicitly embeds multi-layered guardrails into the architectural layers.

Sobhy et al. (2021) and Fatima and Lago (2023) are two recent studies performing extensive literature reviews on the software architecture evaluation methods developed over the years. From the perspective of autonomy, architecture evaluation methods can be categorised based on the degree of human involvement. Some methods are entirely reliant on human input, depending on stakeholders to evaluate the design options (Pooley and Abdullatif, 2010). Others adopt a semi-autonomous process, incorporating human-in-the-loop to guide the evaluation (Ghezzi and Molzam Sharifloo, 2013). A distinct subset of methods operate fully autonomously, conducting evaluation without human intervention (Sobhy et al., 2020; Van Der Donckt et al., 2018).

In terms of the evaluation stage, various methods cover design-time (Kazman et al., 2000; Nord et al., 2003; Bengtsson and Bosch, 1998; Bengtsson et al., 2004; Eloranta et al., 2014; Faniyi et al., 2011), run-time (Yang et al., 2009; Heaven et al., 2009; Moreno et al., 2016; Esfahani et al., 2013), and continuous evaluation (Christensen et al., 2010; Pooley and Abdullatif, 2010; Sobhy et al., 2020). Some of the design-time architecture evaluation approaches, such as ATAM (Kazman et al., 2000) and CBAM (Nord et al., 2003), handle tradeoffs manually through the analysis of tradeoff points elicited from stakeholders, or do not consider it at all (such as SAAM (Kazman et al., 1994) and ALMA (Lassing et al., 2002)). As for the run-time architecture

evaluation approaches, some run-time approaches provide automatic management of tradeoffs (Sobhy et al., 2020), whereas one noticeable investigation is that many approaches have no support for tradeoff management (Yang et al., 2009).

Architecture can experience epistemic uncertainty and/or aleatory uncertainty. Most design-time architecture evaluation methods focus on mitigating epistemic uncertainty (Bengtsson and Bosch, 1998; Bengtsson et al., 2004; Kazman et al., 1994). Conversely, aleatory uncertainty is more commonly encountered in run-time architecture evaluation methods (Faniyi et al., 2011; Kim and Park, 2009). A limited number of methods address both epistemic and aleatory uncertainty, including certain design-time methods (Faniyi et al., 2011; Ionita et al., 2004; Meedeniya et al., 2011), run-time methods (Esfahani et al., 2013, 2011), and continuous methods (Sobhy et al., 2020). Uncertainties and risks are often addressed by developing a set of scenarios that reflect stakeholder envision the system usage (Kazman et al., 2000; Kozolek, 2011). The accuracy and relevance of the evaluation highly depend on the choice of these scenarios. For instance, ATAM identifies and documents risks that could hinder the achievement of quality attribute goals. This includes risks stemming from architectural decisions that negatively impact quality attributes, points of sensitivity where minor changes result in significant quality shifts, and tradeoffs where one decision influences multiple attributes (Jones and Lattanzio, 2001). While ATAM focuses on the risks and benefits of architecture decisions, it does not explicitly take into account the cost. CBAM (Nord et al., 2003) extends ATAM by integrating cost-benefit tradeoff analysis into the evaluation process. Scenario-based design-time architecture evaluation methods, such as ATAM (Kazman et al., 2000), CBAM (Nord et al., 2003), ATMIS (Faniyi et al., 2011), and APTIA (Kazman et al., 2006) partially address uncertainty through the use of scenarios. However, these methods primarily operate at design time and are heavily reliant on stakeholder input.

Traditional software architecture evaluations (Babar et al., 2004; Sobhy et al., 2021; Fatima and Lago, 2023), such as ATAM (Kazman et al., 2000) and SAAM (Kazman et al., 1994), provide detailed insights into evaluation steps and artefacts. However, there is lack of guidance tailored for agent architecture evaluation, especially insufficient consideration of the different ways of building agents and their impact on architecture evaluation.

3. Methodology

To develop the AgentArcEval method and the corresponding catalogue of agent-specific general scenarios, we undertook a structured approach that combined insights from our hands-on project experience with a systematic literature review (SLR) of existing architecture evaluation papers. We began by leveraging our project experience in designing and evaluating the architecture of agents, such as tax copilot, scientific agent, and tender evaluation agent. Through these projects, we identified the unique characteristics of agents, quality attributes essential for agents, frequently encountered scenarios in agent deployment, common design patterns of agents.

We then conducted an SLR of prior software architecture evaluation methods by searching the key words “software architecture evaluation”, “software architecture assessment”, “software architecture analysis” on Google Scholar with time period constraint “until 2 Jan 2025”. The initial search result returned 1080 papers, and after screening, snowballing and selection through a set of criteria, the finalised set included 119 primary studies. Specifically, the inclusion and exclusion criteria are listed as follows.

Inclusion criteria:

1. A paper that presents a novel software architecture evaluation method/tool.
2. A paper that presents a case study.

Exclusion criteria:

1. A survey/review paper that analyses existing software architecture evaluation methods.
2. A paper that proposes analysis frameworks for architecture evaluation methods.
3. A paper that is not written in English.
4. Conference version of a study that has an extended journal version.
5. PhD/Master's dissertations, tutorials, editorials, books.

We examined existing architecture evaluation methods with four main research questions: (i) Can existing software architecture evaluation methods be applicable to evaluating the architecture of foundation model based agents? (ii) Does the method support autonomy? (iii) Does the method support continuous evolution? (iv) Does the method provide general scenarios? Through this SLR, we found that they are not appropriate “out of the box” for evaluating agent architectures due to the unique challenges of such systems.⁴ ⁵ By combining practical insights from real-world projects with an SLR of existing methods, we were able to design the AgentArcEval method and the general scenario catalogue to address both theoretical gaps and operational considerations of agent architecture evaluations. We used a real-world agentic system to evaluate the usefulness of the proposed method and catalogue.

In terms of methodology limitations and mitigation strategies, while the AgentArcEval method and scenario catalogue were developed based on insights from a limited number of real-world projects, we have previously performed an SLR on agents (Lu et al., 2024) and have a thorough understanding of agent design principles and practices. The general scenarios and quality attributes within the catalogue were initially derived from the authors' project experience and architecture expertise. To minimise potential bias, we reviewed the classic software/AI architecture books (Bass et al., 2021, 2025) as well as responsible AI principles (Lu et al., 2023). This allowed us to systematically identify the critical quality attributes essential for the development and operation of AI agents with confidence. Additionally, given the rapid evolution of FMs and agents, there is a risk that the AgentArcEval method or scenario catalogue may become outdated or misaligned with emerging best practices. To address this, we plan to adopt a community-driven approach by publishing the method and catalogue as a living document and inviting contributions from the broader research and development community.

4. The AgentArcEval method

In this section, we present the AgentArcEval method, a scenario-based architecture evaluation method tailored specially for FM-based agents. The method builds on established principles and process from the ATAM, particularly its use of quality attribute scenarios and stakeholder-driven analysis. However, FM-based agents present new architectural challenges, such as goal-directed autonomy and continuous evolution. These aspects are not adequately addressed by traditional evaluation methods and require tailored extensions to ensure meaningful assessment. AgentArcEval addresses this gap by incorporating agent-specific artefacts and guardrails into the evaluation process. The method enables teams to reason explicitly about common decision points in FM-based agent design. It supports early-stage analysis of quality trade-offs through structured, context-specific scenarios, helping stakeholders assess how an architecture responds to real-world operational demands.

⁴ Protocol for review purpose only: <https://docs.google.com/document/d/1cPuQN5PqPC81VNn8W7ktMFwE6pH-isR4/>.

⁵ Extraction sheet for review purpose only: https://docs.google.com/spreadsheets/d/1fHzD7EUT_hdE_76uSlOZ2-Sh_Po7jrwS/.

4.1. User types

The AgentArcEval method can be primarily applied by two types of users, each with distinct objectives.

- **Project team architects:**

- To assess and refine their own agent architectures to meet project goals and governance needs.
- To incorporate mechanisms for runtime architecture evaluation and continuous evolution into their architectures.

- **External architecture experts:**

- To conduct independent architecture evaluations, with a focus on improving the agent architecture.
- To audit the project for quality assurance and regulatory compliance.

4.2. Evaluation time

The AgentArcEval method can be used at various project stages.

- **Early in the project:** Project team architects can apply AgentArcEval after the initial architecture design to evaluate and refine critical design decisions before substantial resources are committed. This ensures that the architecture is aligned with the project goals, while also addressing potential issues/risks early on.
- **Before deployment:** Both project team architects and external architecture experts can apply AgentArcEval towards the end of development, before deployment, to conduct a comprehensive review of the architecture as a check of quality and governance needs.
- **After deployment:** Both project team architects and external architecture experts can use AgentArcEval post-deployment to assess how well the architecture performs in a real-world environment. This can help identify any emerging issues/risks that have not been identified during development and provides opportunities for improvement and evolution.
- **Continuously at runtime:** Although post-deployment architecture evaluation and adaptation is expensive, the AgentArcEval method can be integrated into the agent architecture to enable continuous risk assessment, runtime architecture evaluation and adaptive evolution, ensuring proactive risk management throughout the system's operation.

4.3. Inputs and outputs

The following inputs are required for using AgentArcEval.

- **Project specification:** A document outlining the agent goals, intended usage context, and targeted capabilities is valuable for architecture evaluation. Since such documentation is often incomplete or lacks depth in practice, it is important to actively engage with the project team to gather and refine this information ahead of time, or in real time (Bass and Kazman, 2002).
- **Architecture design specification:** A detailed document describing the overall architecture, including the architecture approaches employed, key design decisions made, and an initial tradeoff and risk analysis. As with the project specification, it is often necessary to interact with the project's leadership in advance to acquire, or generate, this information.
- **Governance needs:** The relevant AI governance regulations, standards and policies, which the project must comply with.

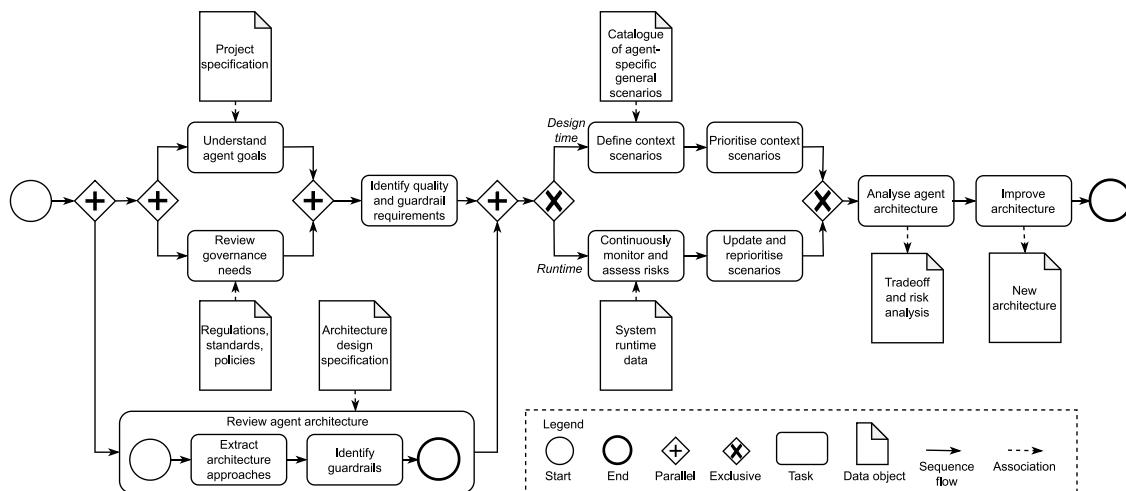


Fig. 2. The process of AgentArcEval.

- **Catalogue of agent-specific general scenarios:** Context scenarios can be defined based on the identified quality attributes and the catalogue of agent-specific general scenarios provided here.
- **System runtime data:** If the system has been deployed, data can be collected during its operation, providing useful insights into its real-world behaviour and performance, which are important for identifying changing or new risks and evaluating the architecture's effectiveness during runtime.
- **Stakeholder insights:** Inputs from project stakeholders regarding the project's goals, requirements, guardrail constraints, context scenarios, and the prioritisation of different scenarios.

The output of AgentArcEvaluation is architecture evaluation results, including a **tradeoff analysis**, a **list of identified risks**, and **recommendations for improving the architecture**.

4.4. Steps

Fig. 2 illustrates the key steps involved in the AgentArcEval method. To begin with, a user needs to comprehend the goals of a particular agent, review top-level governance requirements, and also the agent architecture design. In particular, the user should identify the quality and guardrail requirements after understanding the goals and scrutinising governance needs.

- **Understand agent goals:** The process starts with establishing a clear understanding of the agent's goals and intended capabilities. When the goals are only vaguely specified, the evaluation team should work closely with stakeholders to elicit, clarify and refine them as part of the architecture evaluation.
- **Review governance needs:** A thorough review of relevant AI governance regulations, standards, and policies is conducted to map to quality/guardrail requirements.
- **Identify quality and guardrail requirements:** The quality/guardrail requirements and associated metrics, which the agent architecture must meet, should be defined.
- **Review agent architecture:** An interactive review of the current agent architecture is performed, examining its components, structure, and interactions. This includes collaboratively identifying architecture approaches used such as key design patterns, tactics, design decisions, and guardrails implemented within the architecture.

Subsequently, the user needs to split the architecture evaluation in terms of design time and runtime. The Design time side include defining context scenarios and prioritising context scenarios, while

runtime side consists of considering continuous risk monitoring and assessment, as well as updating and reprioritising scenarios according to agent operation. Finally, the user can integrate the results to analyse and improve the agent architecture. Detailed process of each step is listed as follows.

- **Define context scenarios:** Based on the identified quality and guardrail requirements, context scenarios are developed to explore the various operational environments and use cases in which the agent will function. These scenarios help evaluate how well the architecture supports the required qualities under realistic conditions.
- **Prioritise context scenarios:** The context scenarios are ranked based on their importance, potential impact and risk, and relevance to agent goals.
- **Analyse agent architecture:** A thorough analysis of the architecture is conducted to evaluate how well it supports the agent's goals, meets quality requirements, and complies with guardrails and regulations. This analysis is guided by available artefacts such as design documents, architecture diagrams, system logs, evaluation results, and structured tools like checklists or assessment templates. Key architectural tradeoffs and risks are identified.
- **Improve architecture:** Based on the analysis, recommendations for improving the architecture are made to better meet the agent's goals and mitigate identified risks. This may involve refining design decisions, adding or changing architecture approaches.
- **Continuously monitor and assess risks:** Continuous monitoring and assessment are performed to detect any emerging or evolving risks during runtime.
- **Update and reprioritise scenarios:** As the agent evolves, it is important to periodically revisit and update the context scenarios and their prioritisation. This triggers a runtime evaluation of the architecture by revisiting key steps including **analysing agent architecture** and **improving architecture** to ensure the architecture can meet changing usage contexts and risks.

5. General scenarios

In this section, we present a catalogue of agent-specific general scenarios for quality attributes including accuracy, adaptability, efficiency, privacy, security, fairness, availability, observability, transparency, safety, and contestability. In particular, the quality attributes were carefully selected as they are regarded critical to the development and operation of AI systems (Bass et al., 2025, 2021) and their alignment with responsible AI principles (Lu et al., 2023). They

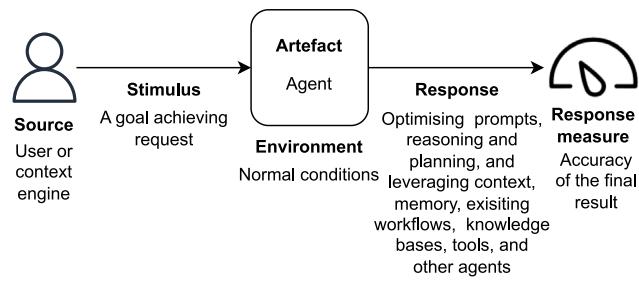


Fig. 3. Accuracy general scenario.

present a starting point derived from both literature review and practical considerations in agent systems. We constructed the catalogue by first extracting a long-list of situations from three deployed agent projects and a systematic literature review, then retaining one exemplar scenario for each of the eleven quality attributes, including accuracy, adaptability, efficiency, privacy, security, fairness, availability, observability, transparency, safety, and contestability. A scenario entered the baseline catalogue only when it met two checks: (1) the situation has occurred in an actual FM-agent deployment, guaranteeing practical relevance; (2) it can be written in the standard “Source → Stimulus → Environment → Artefact → Response → Measure” format, so its outcome can be verified objectively. This screening produces a lean but comprehensive starter set, which teams can further extend or refine during the “Define Context Scenarios” workshop to address emerging, domain-specific concerns. As the field evolves, new quality attributes and corresponding scenarios will inevitably surface; the eleven scenarios presented here are intended only as a foundational baseline for evaluation.

Each scenario is described using the traditional scenario template (Bass et al., 2021):

- **Source:** The entity (e.g., user or external auditor) or artefact (e.g., runtime evaluator) that generated the stimulus.
- **Stimulus:** The event or condition that requires a response from the system.
- **Environment:** A context where the stimulus occurs.
- **Artefact:** The agent artefacts that are stimulated.
- **Response:** The activity undertaken as a result of the arrival of the stimulus.
- **Response measure:** The metrics used to evaluate whether the response is satisfactory. The sources of response measures are from Bass et al. (2025, 2021).

5.1. Accuracy general scenario

Fig. 3 shows the portions of accuracy general scenario.

- **Source:** A user relying on the agent to achieve goals, or context engine proactively suggesting goals to the user.
- **Stimulus:** A goal achieving request initiated by the user or proactively suggested by the agent's context engine.
- **Artefact:** Agent.
- **Environment:** The agent operates in normal conditions.
- **Response:** The agent accurately accomplishes the goal through optimised prompts, reasoning and planning, past agent memory, existing workflows, knowledge bases, external tools, and other agents as needed.
- **Response Measure:** The accuracy of the final result in relation to the user expectation.

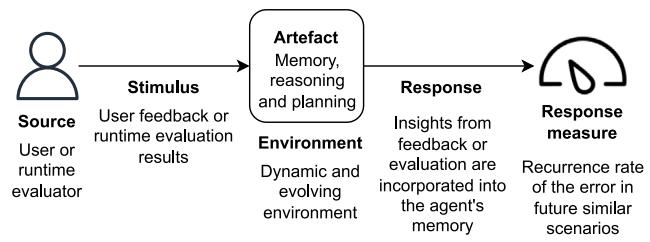


Fig. 4. Adaptability general scenario.

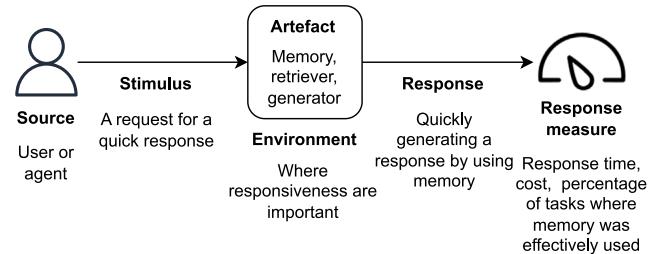


Fig. 5. Efficiency general scenario.

5.2. Adaptability general scenario

Fig. 4 shows the portions of adaptability general scenario.

- **Source:** A user who provides feedback on the agent's outputs (either final or intermediate ones), or an automated runtime evaluator that continuously monitors and evaluates these outputs.
- **Stimulus:** The agent receives feedback from the user, or it gets evaluation results from the runtime evaluator.
- **Artefact:** Agent memory, reasoning and planning.
- **Environment:** A dynamic environment where the agent continuously collects and processes user feedback and runtime evaluation results.
- **Response:** The agent updates its memory to incorporate the feedback or evaluation insights. The updated memory can adapt the agent's future prompts, reasoning and planning, and workflow execution.
- **Response Measure:** Rate of successful adaptations, time to adapt.

5.3. Efficiency general scenario

Fig. 5 shows the portions of efficiency general scenario.

- **Source:** A user or the agent itself triggering the need for a quick response based on previous interactions or learned experiences stored in the agent memory.
- **Stimulus:** A request where the agent can utilise relevant memory from past interactions or task executions to quickly generate a response, minimising the need for redundant processing or reasoning.
- **Artefact:** Agent memory, retriever, generator.
- **Environment:** An environment where responsiveness is important.
- **Response:** The agent quickly generates a response by using relevant agent memory, such as previous results, existing workflows, or past experience.
- **Response Measure:** The response time, cost.

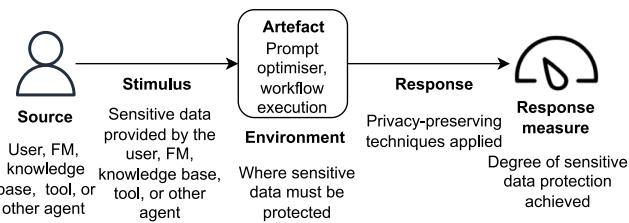


Fig. 6. Privacy general scenario.

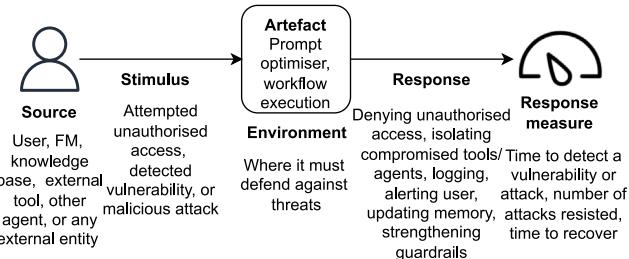


Fig. 7. Security general scenario.

5.4. Privacy general scenario

Fig. 6 shows the portions of privacy general scenario.

- **Source:** A user, FM, knowledge base, external tool, or other agent providing sensitive data.
- **Stimulus:** Sensitive data provided by the user, FM, knowledge base, external tool, or other agent.
- **Artefact:** Workflow execution.
- **Environment:** An environment where sensitive data must be protected.
- **Response:** Privacy-preserving techniques, such as data anonymisation, encryption, and differential privacy, are applied to protect sensitive data.
- **Response Measure:** Percentage of protected sensitive data.

5.5. Security general scenario

Fig. 7 shows the portions of security general scenario.

- **Source:** A user, FM, external tool, other agent, or any external entity attempting unauthorised access or posing potential vulnerabilities or threats.
- **Stimulus:** An attempted unauthorised access, a detected vulnerability, a malicious attack.
- **Artefact:** Prompt optimiser, workflow execution.
- **Environment:** A potentially hostile environment, where it must defend against a variety of threats while maintaining normal operations and overall system integrity.
- **Response:** The agent detects the security threats and takes immediate actions to mitigate them. This includes analysing vulnerabilities, denying unauthorised access, isolating compromised tools or agents, logging incidents, alerting the user or administrator, updating agent memory, and strengthening guardrails.
- **Response Measure:** Time to detect a vulnerability, time to recover from a successful attack.

5.6. Fairness general scenario

Fig. 8 shows the portions of fairness general scenario.

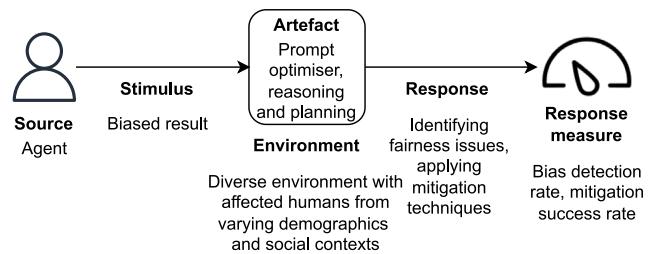


Fig. 8. Fairness general scenario.

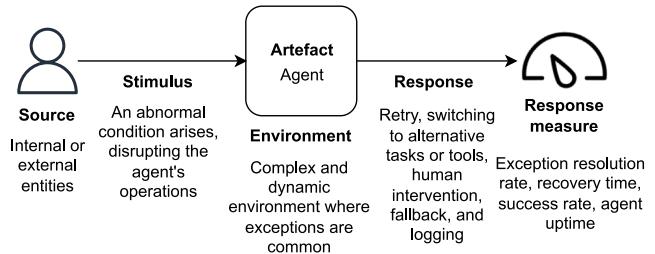


Fig. 9. Availability general scenario.

- **Source:** A user generating a biased result.
- **Stimulus:** Bias in the agent's final result, or intermediate result or user.
- **Artefact:** Agent.
- **Environment:** A diverse environment with affected humans from varying demographics and social contexts.
- **Response:** The agent identifies fairness issues and applies mitigation techniques to ensure equitable outcomes.
- **Response Measure:** The bias detection rate, mitigation success rate.

5.7. Availability general scenario

Fig. 9 shows the portions of availability general scenario.

- **Source:** Internal or external entities that triggers abnormal conditions interrupting the agent's operations.
- **Stimulus:** An abnormal condition arises, disrupting the agent's operations, such as unmet task dependencies, tool failures, data quality issues, or user interventions.
- **Artefact:** Agent.
- **Environment:** A complex and dynamic environment where exceptions are common in task execution, external tool API calls, data operations, and interactions with other agents.
- **Response:** The agent detects the exceptions and activates appropriate handling mechanisms, including retrying failed tasks, switching to alternative tasks or tools, requesting human intervention, applying fallback, using alternative resources, and logging the incident.
- **Response Measure:** The exception resolution rate, mean recovery time, task success rate, agent uptime.

5.8. Observability general scenario

Fig. 10 shows the portions of observability general scenario.

- **Source:** A monitoring system, system administrator, or external evaluator, aiming to assess the agent's performance and behaviour in real-time.

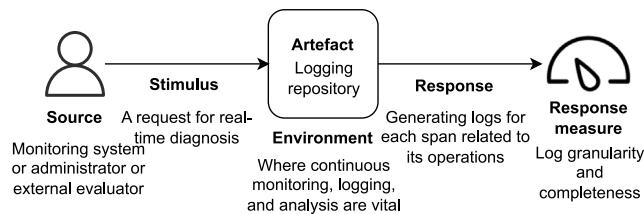


Fig. 10. Observability general scenario.

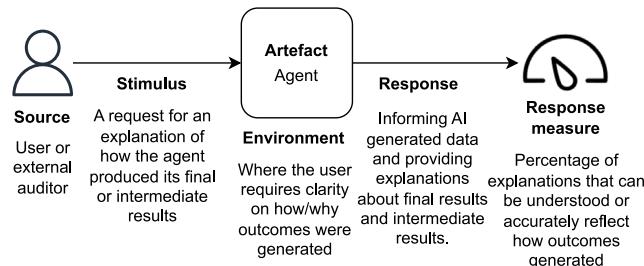


Fig. 11. Transparency general scenario.

- Stimulus:** A request for real-time diagnosis, triggered by events like performance degradation, abnormal execution.
- Artefact:** Log repository.
- Environment:** A complex, real-time environment where continuous monitoring, logging, and analysis are vital.
- Response:** The agent generates logs for each span, a traceable unit of operation, related to its behaviour, including reasoning span, planning span, workflow span, task span, tool span, evaluation span, and FM span. This provides stakeholders with real-time insights into the agent's health, goal completion progress, and potential anomalies.
- Response Measure:** Log granularity and completeness.

5.9. Transparency general scenario

Fig. 11 shows the portions of transparency general scenario.

- Source:** A user or external auditor seeking to understand the reasoning behind the agent's final/intermediate results.
- Stimulus:** A request for an explanation of how the agent produced its final or intermediate results.
- Artefact:** Agent.
- Environment:** A context where the user requires clarity on how/why outcomes were generated.
- Response:** The agent informs that the outcome was generated by an AI agent and provides a human-understandable explanation of its final and intermediate results. This explanation includes justifications for goal understanding, planning, agent memory and knowledge retrieval, external tool and agent selection, and workflow execution.
- Response Measure:** The percentage of users who can understand the explanations, the time taken to generate and deliver explanations to the user.

5.10. Safety general scenario

Fig. 12 shows the portions of safety general scenario.

- Source:** A user, external system, or internal monitoring component detecting potential unsafe behaviour.

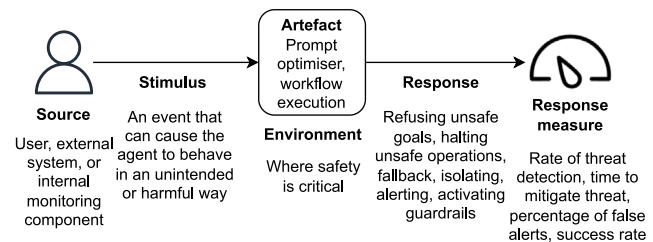


Fig. 12. Safety general scenario.

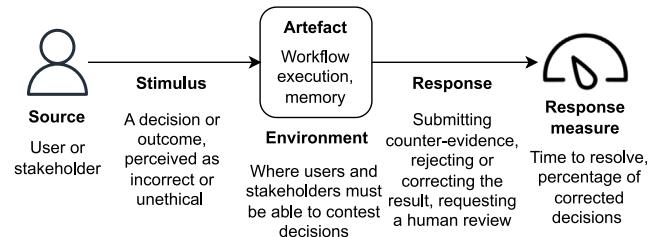


Fig. 13. Contestability general scenario.

- Stimulus:** An event that can cause the agent to behave in an unintended or harmful way.
- Artefact:** Prompt optimiser, workflow execution.
- Environment:** An environment where safety is critical.
- Response:** The agent detects potential safety threats and immediately triggers predefined safety mechanisms, including refusing unsafe human goals, halting unsafe operations, switching to a safe fallback mode, isolating faulty components, alerting humans, activating guardrails.
- Response Measure:** The rate of threat detection, percentage of unsafe events avoided, percentage of events that require human intervention, the time taken to mitigate a safety threat.

5.11. Contestability general scenario

Fig. 13 shows the portions of contestability general scenario.

- Source:** A user or stakeholder who wants to question or contest the agent's decision or outcome.
- Stimulus:** A specific decision or outcome generated by the agent, perceived as incorrect or unethical, which requires further justification or review.
- Artefact:** Workflow execution, agent memory.
- Environment:** An environment where users and stakeholders must be able to contest decisions, especially in high-stakes environments such as healthcare, legal, financial, or public-sector applications.
- Response:** The agent provides a detailed explanation of how outcome was produced. Users challenge the outcome by submitting counter-evidence, rejecting or correcting the result, requesting a human review.
- Response Measure:** The time taken to resolve contested decisions, the percentage of contested decisions that result in a meaningful correction after the review process.

6. Case study

In this section, we evaluate the usefulness of AgentArcEval and the catalogue of agent-specific general scenarios through a case study on

the architecture evaluation of the Luna tax copilot.⁶ The goal is to demonstrate the method's practical applicability in a real-world, operational context. We provide a structured analysis of the architecture evaluation process, demonstrating how the AgentArcEval methodology and the general scenario catalogue guide architectural analysis and decision making.

The evaluation followed a qualitative, scenario-based analysis, following the AgentArcEval process. We first defined a set of context-specific quality scenarios, derived from our general scenario catalogue and refined in collaboration with key stakeholders from the Luna project. These stakeholders include software architects, developers, and product managers, who contributed domain knowledge, technical insights, and operational constraints. Their input helped prioritise quality attributes, validate the realism of the scenarios, and assess how the proposed architectural responses aligned with actual system goals.

Each scenario focused on a critical quality attribute and was written in the standard stimulus-response-measure format. We then assessed how the revised architecture satisfied each scenario, highlighting relevant trade-offs and identifying risk areas where guardrails were necessary. This structured walkthrough helped surface several architectural tensions early in the design process and guided concrete design decisions.

6.1. Understand agent goals

After discussions with the project team, the architecture analysts gained a clear understanding that the Luna tax copilot is an agent system to provide professional and accurate tax recommendation, with clear references to the Australian Taxation Office database,⁷ specially for tax professionals.

6.2. Review governance needs

The architecture analysts reviewed the Australian Voluntary AI Safety Standard⁸ and identified the following relevant guardrails.

- Protect AI systems, and implement data governance measures to manage data quality and provenance.
- Test AI models and systems to evaluate model performance and monitor the system once deployed.
- Enable human control or intervention in an AI system to achieve meaningful human oversight.
- Inform end-users regarding AI-enabled decisions, interactions with AI and AI-generated content.
- Establish processes for people impacted by AI systems to challenge use or outcomes.
- Be transparent with other organisations across the AI supply chain about data, models and systems to help them effectively address risks.
- Keep and maintain records to allow third parties to assess compliance with guardrails.

6.3. Identify quality and guardrail requirements

Based on an understanding of the project goals, deployment context, and relevant AI safety standards, we worked with Luna project stakeholders to identify seven key quality attributes and guardrail types most critical to the system. While our general catalogue includes eleven, only those deemed directly relevant to the Luna tax copilot's operational risks and design priorities were selected.

- **Accuracy:** Providing correct and precise tax recommendation to tax professionals, ensuring it is aligned with the most up-to-date data from Australian Taxation Office legal database's latest data.
- **Adaptability:** Incorporating user feedback into the agent memory to improve future recommendation generation.
- **Efficiency:** Quickly delivering tax recommendation for queries that are similar to previously answered ones.
- **Transparency:** Providing clear practical explanations for each recommendation, detailing the copilot's reasoning process and citing the specific legal references and foundation model version were used.
- **Observability:** Enabling stakeholders to track historical queries, monitor user feedback, and receive alerts on copilot health, performance or users complaints.
- **Contestability:** Allowing users to challenge the copilot's recommendation and submit feedback for review.
- **Privacy:** Ensuring all input and output data is desensitised to protect personally identifiable information and sensitive company data.

6.4. Define and prioritise context scenarios

The architecture analysts defined and prioritised context scenarios based on the goals, identified quality requirements, and the general scenario catalogue. These scenarios and their priorities were then confirmed by the project team. The sources of the response measures are from the requirements of the project and governance needs. Below are the context scenarios for the identified quality attributes that are critical to the tax copilot.

Accuracy Context Scenario

- **Scenario ID:** 1
- **Quality Attribute:** Accuracy
- **Priority:** High
- **Source:** Tax professional.
- **Stimulus:** The tax professional submits a query about capital gains tax on the sale of a rental property.
- **Artefact:** Context engine, reasoning and planning, workflow execution, retriever, generator, vector database.
- **Environment:** The tax professional uses the copilot during a tax season where accurate recommendation is key.
- **Response:** The copilot retrieves the latest relevant tax regulations and provides accurate, context-specific tax recommendation.
- **Response Measure:** The copilot should achieve at least 95% accuracy in providing relevant responses and using 95% correct references to the applicable regulations.

Adaptability Context Scenario

- **Scenario ID:** 2
- **Quality Attribute:** Adaptability
- **Priority:** Medium
- **Source:** Tax professional or runtime evaluator component
- **Stimulus:** A mistake is identified in the system's recommendation on capital gains tax, either flagged manually by a tax professional or automatically detected by a runtime evaluation component.
- **Environment:** The feedback is submitted by the tax professional during live system operation via a structured correction interface, or auto-generated evaluation report.
- **Artefact:** Agent memory, reasoning and planning
- **Response:** The system updates its agent memory with the new information and adjusts its reasoning logic to avoid repeating the same error in similar future cases.
- **Response Measure:** 99% of valid feedback instances result in a correct update.

Efficiency Context Scenario

⁶ <https://empathetic-ai.com>.

⁷ <https://www.ato.gov.au/single-page-applications/legaldatabase>.

⁸ <https://www.industry.gov.au/publications/voluntary-ai-safety-standard>.

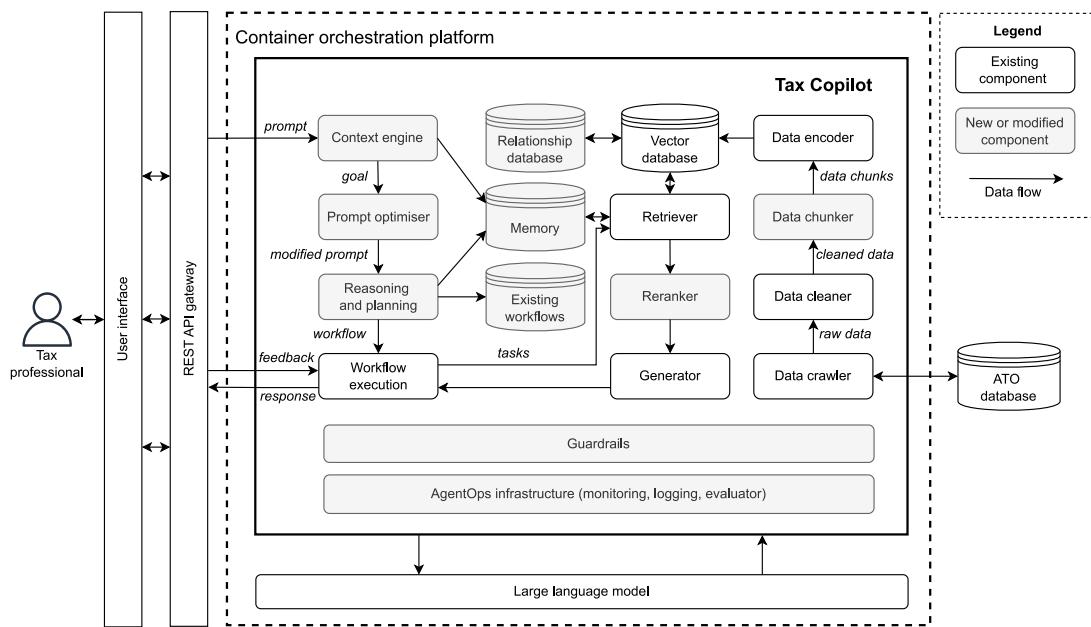


Fig. 14. Component View of the Luna Tax Copilot Architecture.

- **Scenario ID:** 3
- **Quality Attribute:** Efficiency
- **Priority:** Medium
- **Source:** Tax professional
- **Stimulus:** A query is submitted about vehicle tax deductions, which has been previously answered.
- **Environment:** The query is made during peak filling season, requiring a quick response.
- **Artefact:** Agent memory, retriever, generator
- **Response:** The tax copilot retrieves the previously provided recommendation on vehicle tax deduction from agent memory and delivers it promptly to the user.
- **Response Measure:** The response time must be under 1 s for queries previously answered.

Transparency Context Scenario

- **Scenario ID:** 4
- **Quality Attribute:** Transparency
- **Priority:** High
- **Source:** Tax professional.
- **Stimulus:** The tax professional requests an explanation of how the copilot arrived at its recommendation for income tax deduction.
- **Environment:** The query is made while preparing tax recommendation reports for a client.
- **Artefact:** Copilot.
- **Response:** The tax copilot provides its reasoning process, listing the legal sources consulted and explaining the FM version used.⁹
- **Response Measure:** At least 95% of users understand the explanation, at least 95% of the references are correct.

Observability Context Scenario

⁹ Recording the FM version enables users, developers, and auditors to verify how a specific response was generated, thereby supporting both traceability and reproducibility. If an error or bias is identified, knowing the exact FM version helps determine whether the issue stems from the model's inherent limitations or from other factors such as prompt design, retrieval context, or system configuration. In our case study, the Luna tax copilot employed OpenAI GPT-4o.

- **Scenario ID:** 5
- **Quality Attribute:** Observability
- **Priority:** Medium
- **Source:** Monitoring component.
- **Stimulus:** An alert is triggered due to an increasing number of low scores provided by the copilot user about recommendation on small business tax deductions.
- **Environment:** The copilot is continuously monitored.
- **Artefact:** Log repository.
- **Response:** The tax copilot automatically logs all queries, responses, and user feedback.
- **Response Measure:** Log 100% queries, responses, and feedback.

Contestability Context Scenario

- **Scenario ID:** 6
- **Quality Attribute:** Contestability
- **Priority:** High
- **Source:** Tax professional
- **Stimulus:** A tax professional disagrees with the recommendation provided on superannuation contributions and submits feedback to challenge the recommendation.
- **Environment:** An environment where tax professionals must be able to contest and review the recommendation provided.
- **Artefact:** Retriever, generator, agent memory.
- **Response:** The tax copilot logs the feedback and flags the case for review by tax experts.
- **Response Measure:** 100% of contested cases are reviewed and resolved within 48 h.

Privacy Context Scenario

- **Scenario ID:** 7
- **Quality Attribute:** Privacy
- **Priority:** High
- **Source:** Tax professional submitting sensitive data or the generator component producing sensitive data.
- **Stimulus:** The tax professional submits personal data for tax calculation, or the generator component produces responses that contain sensitive information.

- **Environment:** The copilot is operated in an environment where sensitive data must be protected.
- **Artefact:** Prompt optimiser, generator.
- **Response:** The tax copilot desensitised all sensitive data before processing request or generating responses.
- **Response Measure:** 99% of sensitive data must be filtered.

6.5. Analyse and improve architecture

[Fig. 14](#) illustrates one view of the architecture of the Luna tax copilot. The white boxes represent the original architecture before the review while the grey boxes indicate the components modified or added after the review. In this section, we will analyse the architecture by mapping to the defined context scenarios.

The proposed architecture components are deployed using containerisation technology with Docker, orchestrated via Kubernetes, ensuring scalability and resilience across various workloads ([Burns et al., 2016](#)). Data components such as the Vector Database and Relationship Database are hosted on cloud-managed services (e.g., AWS RDS and Pinecone), enabling efficient scaling, reliable storage, and low-latency access ([Amazon Web Services, 2025a](#); [Pinecone, 2025a](#)). Communication between internal components, including the Retriever, Reranker, and Generator, is handled via RESTful APIs using HTTP and JSON payloads, providing structured and efficient data transfer. Data ingestion components such as the Data Crawler fetch information from the ATO database through secure, encrypted channels to ensure compliance and security. All data flow within the system utilises secure internal networking infrastructure provided by the cloud environment (e.g., AWS Virtual Private Cloud) to maintain data integrity and security standards ([Amazon Web Services, 2025b](#)).

To clarify the relationship between the Luna Tax Copilot architecture shown in [Fig. 14](#) and the reference architecture shown in [Fig. 1](#), we provide a component-level mapping. The **context engine** in the Luna architecture directly corresponds to the **context engineering** component in the reference architecture. Both architectures include a **prompt optimiser** component and a **reasoning and planning**. Luna's **workflow execution** component is an instantiation of the **workflow execution engine** in the reference architecture. Similarly, both architectures include a **memory** component. In Luna, there is a combination of a **relational database** and a **vector database**, which together align with the **knowledge base** in the reference architecture. Luna also incorporates several data pre-processing modules, which can be considered auxiliary components operating either externally or internally, and are therefore not explicitly represented in the reference architecture. RAG components including Retriever, Re-ranker, and Generator functions collectively form part of the **Task Execution** process in the reference architecture. Furthermore, both architectures feature **Guardrails** and **AgentOps** layers, and Luna's **Large Language Model** corresponds to the **Foundation Model** in the reference architecture. This mapping shows that the Luna Tax Copilot architecture concretely instantiates the main functional components and layers of the reference architecture.

6.5.1. Architecture analysis for accuracy context scenario

To ensure accuracy, several architecture approaches have been applied to the previous architecture:

- **Context engine:** A topic classifier has been integrated into the system to categorise user queries before generating responses, allowing the classified topic to serve as contextual input for subsequent processing.
- **Prompt optimiser:** A prompt template is used to help large language model (LLM) understand the queries.

• **Generator:** Regarding the **response optimisation strategy**, the previous architecture design has included references for each generated piece of recommendation, such as IDs and links to rulings. Response templates have been employed to structure the recommendation, making it easier for tax professional to read and understand the reasoning behind the generated response.

• **Retriever:** The **retriever** fetches two types of information: private ruling and legislation. Legislation defines general rules, while private rulings cover the practical application of relevant provisions. If private ruling is successfully retrieved, the **generator** uses the template that combines both private ruling and legislation. If only legislation is retrieved, the legislation-specific template is applied.

• **Data crawler:** Legislation was crawled once since there was no updates after crawling, whilst for private rulings, the team decided to crawl ruling data from the ATO database on a weekly basis instead of daily regarding **crawling frequency**, to balance the trade-off between accuracy and cost.

In addition to the above, there are other approaches to further improve accuracy.

• **Context engine: Context definition** is a key design decision for better understanding the context. Beyond query and scenario classification, creating user profile is also helpful. The profile can either be created by having users input their information, or the copilot can continuously learn and update the profile based on interaction. To optimise accuracy, we opted to have users initially create their profile, with the copilot continuing to learn and update the profile based on the subsequent interactions.

• **Prompt optimiser:** Regarding **prompt optimisation strategy**, we considered generating multiple prompts to improve understanding but decided against the strategy due to the high cost associated with additional LLM queries. Instead, we will continue using the prompt template.

• **Reasoning and planning:** On the **planning strategy**, we chose to first break down the query and then search relevant legislation followed by relevant rulings. We also discussed creating a repository of **existing workflows** to potentially reuse of workflows from different companies. However, this is a lower priority, as collecting such workflows maybe difficult due to many companies treating them as proprietary intellectual properties.

• **Data chunker:** In the previous architecture, each crawled private ruling is indexed by date and assigned a unique identifier that maps to a dedicated webpage, and the content of this entire webpage is embedded into vector database. In this circumstance, embeddings can contain significant extraneous data beyond a specific question or subject matter, which may dilute the relevance score of subsequent retrieval results and lead to reduced accuracy. Consequently, a data chunker is introduced to split the clean data into different chunks with clearer semantic meaning. For the **chunking strategy**, we chose to store each question on a private ruling webpage as a single data chunk, rather than storing the entire private ruling page information as a single data chunk. Each question is represented as embeddings stored in the vector database. For **metadata storage**, an approach in LangChain's implementation, where metadata is stored within the vector database, facilitates efficient retrieval alongside embeddings, reducing the need for additional relational database queries and minimising indexing overhead ([LangChain, 2023](#)). This aligns with industry best practices, as seen in Pinecone's recommendation for storing metadata within vector databases when fast retrieval and contextual enrichment are key priorities ([Pinecone, 2025a](#)). Therefore, we decided to store the question-related information within the vector database metadata. When the copilot retrieves relevant results, the metadata is fetched simultaneously with the question embeddings, ensuring efficient contextual retrieval.

- **Vector database:** When evaluating the choice between a **vector database (DB)** vs. a **knowledge graph (KG)**, although a KG may capture relationship between entities, such as questions and rules more effectively, it is more cost-efficient to store plain text in a vector database. Considering the ruling data is updated on a weekly basis, it is costly to correctly and efficiently capture the entities and relationships, and create the KGs. For instance, constructing KGs often requires significant manual effort and poses challenges for frequent updates (Ontoforce, 2024). Additionally, integrating data from diverse sources into KGs can be a long, manual process, making seamless integration difficult (Coveo, 2023). Therefore, we decided to proceed with the vector database for now, balancing cost and complexity.
- **Reranker:** Once the **retriever** generates an initial list of ranked results, the **reranker** can further evaluate their relevance by using the LLM-based **evaluator** to refine the ranking. This may increase latency and cost due to additional tokens processed by the LLM. However, integrating a reranker significantly enhances accuracy, as rerankers are much more accurate than embedding models (Pinecone, 2025b). Additionally, rerankers can enhance the relevance of retrieved information in a RAG system, providing more accurate and contextually relevant results (Chen, 2024). Therefore, we decided to incorporate the **reranker** to the architecture design.

6.5.2. Architecture analysis for adaptability context scenario

We identified a gap in the architecture regarding adaptability, as no approaches were supporting it. We decided to include a **memory** component to store the user feedback for each query's response, enabling the copilot to learn and adapt over time. The user profile stored in **memory** can be refined base on feedback, enabling the copilot to tailor responses more effectively. Additionally, the copilot can dynamically select response templates based on the updated user profile, improving personalisation and accuracy in future interactions.

6.5.3. Architecture analysis for efficiency context scenario

We found a lack of architecture support for the efficiency context scenario. We decided to add logic in the **reasoning and planning** to first check whether similar queries have been previously answered in the memory. If a match is found, the copilot will directly return the previously response to the tax professional to improve efficiency and reduce cost associated with additional computation and LLM queries. To ensure accuracy, the copilot will also verify that the response aligns with the most up-to-date legal information, avoiding outdated recommendations when laws/regulations have changed.

6.5.4. Architecture analysis for transparency context scenario

The **generator** component in the previous architecture already includes reasoning behind the recommendation. To further improve transparency and assist a tax professional in understanding how the copilot arrived at its recommendation, while also reminding them that the recommendation is AI-generated, we decided to include a detailed list of all legal sources referenced by the copilot (e.g. IDs and links of tax rulings stored in the chunk metadata) and specify the version of the LLM in the response template.

6.5.5. Architecture analysis for observability context scenario

The previous architecture did not provide support for observability. To address this, we decided to introduce an **AgentOps infrastructure** layer that cross-cuts all copilot components. This layer will log all queries, responses, user feedback, and the input and output of each component, providing comprehensive visibility into the copilot's operations and improving **monitoring and evaluation** capabilities. Regarding **using ground truth or not**, we decided to gather a set of ground truth data from ATO open forum's questions and ATO certified replies. For complex scenarios, we chose to initially collect ground truth data directly from domain experts, then use LLM to generate more ground truth data for evaluation.

6.5.6. Architecture analysis for contestability context scenario

In the previous design, tax professionals were able to provide scores and feedback on the recommendation generated by the copilot. The improved architecture now includes a **memory** component that not only stores user feedback but also tracks the resolution process.

6.5.7. Architecture analysis for privacy context scenario

Guardrail scope is an important design decision. The previous architecture included a desensitiser to ensure that private information is not disclosed in the user prompt. However, sensitive information could also be present in the input and output of other components. Thus, the new architecture introduces **guardrails** that are applied across all components. The guardrails can also be extended to cover additional concerns such as relevance and fairness.

7. Conclusion

In this paper, we present AgentArcEval, a novel architecture evaluation method specially designed to address the unique characteristics of agents. Additionally, we introduced a catalogue of agent-specific general scenarios to guide the generation of context scenarios for specific agents. Through a real-world case study on the architecture evaluation of the Luna tax copilot, we demonstrated the usefulness of AgentArcEval in addressing the architectural complexities of agents. We are currently applying AgentArcEval to additional FM-based agentic systems and will report comparative results in future work.

Our near-term goal is to establish an evolvable community-driven approach to the evaluation of agent systems by publishing the method and catalogue as a living document and inviting contributions from the broader community.

CRediT authorship contribution statement

Qinghua Lu: Writing – review & editing, Writing – original draft, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Dehai Zhao:** Writing – review & editing, Writing – original draft, Validation, Methodology, Investigation, Formal analysis. **Yue Liu:** Writing – review & editing, Writing – original draft, Methodology, Investigation, Data curation, Conceptualization. **Hao Zhang:** Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Liming Zhu:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Funding acquisition. **Xiwei Xu:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Investigation. **Angela Shi:** Writing – review & editing, Validation, Software, Investigation, Conceptualization. **Tristan Tan:** Writing – review & editing, Validation, Investigation, Conceptualization. **Rick Kazman:** Writing – review & editing, Supervision, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

- Amazon Web Services, 2025a. Amazon relational database service (RDS). <https://aws.amazon.com/rds/>. (Accessed 14 March 2025).
- Amazon Web Services, 2025b. Amazon virtual private cloud (VPC). <https://aws.amazon.com/vpc/>. (Accessed 14 March 2025).
- Babar, Muhammad Ali, Zhu, Liming, Jeffery, Ross, 2004. A framework for classifying and comparing software architecture evaluation methods. In: 2004 Australian Software Engineering Conference. Proceedings. IEEE, pp. 309–318.
- Bass, Len, Clements, Paul, Kazman, Rick, 2021. Software Architecture in Practice: Software Architect Practice, third ed. Addison-Wesley.
- Bass, Len, Kazman, Rick, 2002. Making architecture reviews work in the real world. *IEEE Softw.* 19 (01), 67–73.
- Bass, Len, Lu, Qinghua, Weber, Ingo, Zhu, Liming, 2025. Engineering AI Systems: Architecture and DevOps Essentials. Addison-Wesley.
- Bengtsson, PerOlof, Bosch, Jan, 1998. Scenario-based software architecture reengineering. In: Proceedings. Fifth International Conference on Software Reuse (Cat. No. 98TB100203). IEEE, pp. 308–317.
- Bengtsson, PerOlof, Lassing, Nico, Bosch, Jan, van Vliet, Hans, 2004. Architecture-level modifiability analysis (ALMA). *J. Syst. Softw.* 69 (1–2), 129–147.
- Bommasani, Rishi, Hudson, Drew A, Adeli, Ehsan, Altman, Russ, Arora, Simran, von Arx, Sydney, Bernstein, Michael S, Bohg, Jeannette, Bosselut, Antoine, Brunskill, Emma, et al., 2021. On the opportunities and risks of foundation models. arXiv preprint [arXiv:2108.07258](https://arxiv.org/abs/2108.07258).
- Bucaioni, Alessio, Weyssow, Martin, He, Junda, Lyu, Yunbo, Lo, David, 2025. A functional software reference architecture for LLM-integrated systems. In: 2025 IEEE 22nd International Conference on Software Architecture Companion. ICSA-C, IEEE, pp. 1–5.
- Burns, Brendan, Beda, Joe, Hightower, Kelsey, 2016. Kubernetes: Up and Running. O'Reilly Media, Inc..
- Chen, Jiang, 2024. Optimizing RAG with rerankers: The role and trade-offs. <https://zilliz.com/learn/optimize-rag-with-rerankers-the-role-and-tradeoffs>. (Accessed 14 March 2025).
- Chen, Po-Lin, Chang, Cheng-Shang, 2023. Interact: Exploring the potentials of chatgpt as a cooperative agent. [arXiv:2308.01552](https://arxiv.org/abs/2308.01552).
- Chen, Guangyao, Dong, Siwei, Shu, Yu, Zhang, Ge, Sesay, Jaward, Karlsson, Börje F, Fu, Jie, Shi, Yemin, 2023. AutoAgents: A framework for automatic agent generation. [arXiv:2309.17288](https://arxiv.org/abs/2309.17288).
- Christensen, Henrik Bærbak, Hansen, Klaus Marius, Lindstrøm, Bo, 2010. Lightweight and continuous architectural software quality assurance using the asqa technique. In: European Conference on Software Architecture. Springer, pp. 118–132.
- Coveo, 2023. Knowledge graphs: High performance or high maintenance? <https://www.coveo.com/blog/knowledge-graphs/>. (Accessed 14 March 2025).
- Eloranta, Veli-Pekka, van Heesch, Uwe, Avgeriou, Paris, Harrison, Neil, Koskimies, Kai, 2014. Lightweight evaluation of software architecture decisions. In: Relating System Quality and Software Architecture. Elsevier, pp. 157–179.
- Esfahani, Naeem, Elkhodary, Ahmed, Malek, Sam, 2013. A learning-based framework for engineering feature-oriented self-adaptive software systems. *IEEE Trans. Softw. Eng.* 39 (11), 1467–1493.
- Esfahani, Naeem, Kouroshfar, Ehsan, Malek, Sam, 2011. Taming uncertainty in self-adaptive software. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. pp. 234–244.
- Faniyi, Funmilade, Bahsoon, Rami, Evans, Andy, Kazman, Rick, 2011. Evaluating security properties of architectures in unpredictable environments: A case for cloud. In: 2011 Ninth Working IEEE/IFIP Conference on Software Architecture. IEEE, pp. 127–136.
- Fatima, Ifatt, Lago, Patricia, 2023. A review of software architecture evaluation methods for sustainability assessment. In: 2023 IEEE 20th International Conference on Software Architecture Companion. ICSA-C, IEEE, pp. 191–194.
- Ghezzi, Carlo, Molzam Sharifloo, Amir, 2013. Dealing with non-functional requirements for adaptive systems via dynamic software product-lines. In: Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24–29, 2010 Revised Selected and Invited Papers. Springer, pp. 191–213.
- Heaven, William, Sykes, Daniel, Magee, Jeff, Kramer, Jeff, 2009. A case study in goal-driven architectural adaptation. *Softw. Eng. Self-Adaptive Syst.* 109–127.
- Huang, Wenlong, Xia, Fei, Xiao, Ted, Chan, Harris, Liang, Jacky, Florence, Pete, Zeng, Andy, Tompson, Jonathan, Mordatch, Igor, Chebotar, Yevgen, et al., 2022. Inner monologue: Embodied reasoning through planning with language models. [arXiv:2207.05608](https://arxiv.org/abs/2207.05608).
- International Organization for Standardization, 2022. Artificial intelligence — Concepts and terminology.
- Ionita, Mugurel T, America, Pierre, Hammer, Dieter K, Obbink, Henk, Trienekens, Jos JM, 2004. A scenario-driven approach for value, risk, and cost analysis in system architecting for innovation. In: Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture. WICSA 2004, IEEE, pp. 277–280.
- Jin, Ye, Shen, Xiaoxi, Peng, Huijing, Liu, Xiaoan, Qin, Jingli, Li, Jiayang, Xie, Jintao, Gao, Peizhong, Zhou, Guyue, Gong, Jiangtao, 2023. SurrealDriver: Designing generative driver agent simulation framework in urban contexts based on large language model. [arXiv:2309.13193](https://arxiv.org/abs/2309.13193).
- Jones, Lawrence G., Lattanzio, Anthony, 2001. Using the architecture tradeoff analysis method to evaluate a wargame simulation system: A case study.
- Kazman, Rick, Bass, Len, Abowd, Gregory, Webb, Mike, 1994. SAAM: A method for analyzing the properties of software architectures. In: Proceedings of 16th International Conference on Software Engineering. IEEE, pp. 81–90.
- Kazman, Rick, Bass, Len, Klein, Mark, 2006. The essential components of software architecture design and analysis. *J. Syst. Softw.* 79 (8), 1207–1216.
- Kazman, Rick, Klein, Mark, Clements, Paul, 2000. ATAM: Method for Architecture Evaluation. Carnegie Mellon University, Software Engineering Institute Pittsburgh, PA.
- Kim, Dongsun, Park, Sooyong, 2009. Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software. In: 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. IEEE, pp. 76–85.
- Kong, Yilun, Ruan, Jingqing, Chen, Yihong, Zhang, Bin, Bao, Tianpeng, Shi, Shiwei, Du, Guoqing, Hu, Xiaoru, Mao, Hangyu, Li, Ziyue, et al., 2023. TPTU-v2: Boosting task planning and tool usage of large language model-based agents in real-world systems. [arXiv:2311.11315](https://arxiv.org/abs/2311.11315).
- Koziolek, Heiko, 2011. Sustainability evaluation of software architectures: a systematic review. In: Proceedings of the Joint ACM SIGSOFT Conference-QoSA and ACM SIGSOFT Symposium-ISARCS on Quality of Software Architectures-QoSA and Architecting Critical Systems-ISARCS. pp. 3–12.
- LangChain, 2023. LangChain documentation. <https://python.langchain.com>. (Accessed 14 March 2025).
- Lassing, Nico, Bengtsson, PerOlof, Van Vliet, Hans, Bosch, Jan, 2002. Experiences with ALMA: architecture-level modifiability analysis. *J. Syst. Softw.* 61 (1), 47–57.
- Lu, Qinghua, Zhu, Liming, Whittle, Jon, Xu, Xiwei, et al., 2023. Responsible AI: Best Practices for Creating Trustworthy AI Systems. Addison-Wesley.
- Lu, Qinghua, Zhu, Liming, Xu, Xiwei, Xing, Zhenchang, Harrer, Stefan, Whittle, Jon, 2024. 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, pp. 119–126.
- Madaan, Aman, Tandon, Niket, Gupta, Prakhar, Hallinan, Skyler, Gao, Luyu, Wiegriffe, Sarah, Alon, Uri, Dziri, Nouha, Prabhumoye, Shrimai, Yang, Yiming, et al., 2024. Self-refine: Iterative refinement with self-feedback. *Adv. Neural Inf. Process. Syst.* 36.
- Meedeniya, Indika, Moser, Irene, Aleti, Aldeida, Grunske, Lars, 2011. Architecture-based reliability evaluation under uncertainty. In: Proceedings of the Joint ACM SIGSOFT Conference-QoSA and ACM SIGSOFT Symposium-ISARCS on Quality of Software Architectures-QoSA and Architecting Critical Systems-ISARCS. pp. 85–94.
- Moreno, Gabriel A, Cámaras, Javier, Garlan, David, Schmerl, Bradley, 2016. Efficient decision-making under uncertainty for proactive self-adaptation. In: 2016 IEEE International Conference on Autonomic Computing. ICAC, IEEE, pp. 147–156.
- Nascimento, Nathalia, Alencar, Paulo, Cowan, Donald, 2023. Self-adaptive large language model (LLM)-based multiagent systems. In: 2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion. ACSOS-C, IEEE, pp. 104–109.
- Nord, Robert, Barbacci, Mario R, Clements, Paul C, Kazman, Rick, Klein, Mark H, O'Brien, Liam, Tomayko, James E, 2003. Integrating the architecture tradeoff analysis method (ATAM) with the cost benefit analysis method (CBAM).
- Ontoforce, 2024. Knowledge graphs and GenAI: integration architecture and challenges. <https://www.ontoforce.com/blog/knowledge-graphs-and-genai-integration-architecture-and-challenges>. (Accessed 14 March 2025).
- Packer, Charles, Fang, Vivian, Patil, Shishir G, Lin, Kevin, Wooders, Sarah, Gonzalez, Joseph E, 2023. MemGPT: Towards LLMs as operating systems. [arXiv:2310.08560](https://arxiv.org/abs/2310.08560).
- Park, Joon Sung, O'Brien, Joseph, Cai, Carrie Jun, Morris, Meredith Ringel, Liang, Percy, Bernstein, Michael S, 2023. Generative agents: Interactive simulacra of human behavior. In: UIST'23. pp. 1–22.
- Pinecone, 2025a. Pinecone vector database. <https://www.pinecone.io>. (Accessed 14 March 2025).
- Pinecone, 2025b. Rerankers and two-stage retrieval. n.d., <https://www.pinecone.io/learn/series/rag/rerankers/>. (Accessed 14 March 2025).
- Pooley, R.J., Abdullatif, AAL, 2010. CPASA: Continuous performance assessment of software architecture. In: 2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems. IEEE, pp. 79–87.
- Ruan, Jingqing, Chen, Yihong, Zhang, Bin, Xu, Zhiwei, Bao, Tianpeng, Du, Guoqing, Shi, Shiwei, Mao, Hangyu, Zeng, Xingyu, Zhao, Rui, 2023. Tptu: Task planning and tool usage of large language model-based ai agents. [arXiv:2308.03427](https://arxiv.org/abs/2308.03427).
- Sarch, Gabriel, Wu, Yue, Tarr, Michael J, Fragkiadaki, Katerina, 2023. Open-ended instructable embodied agents with memory-augmented large language models. [arXiv:2310.15127](https://arxiv.org/abs/2310.15127).
- Shamsujohha, Md, Lu, Qinghua, Zhao, Dehai, Zhu, Liming, 2024. Towards ai-safety-by-design: A taxonomy of runtime guardrails in foundation model based systems. arXiv preprint [arXiv:2408.02205](https://arxiv.org/abs/2408.02205).
- Shinn, Noah, Labash, Beck, Gopinath, Ashwin, 2023. Reflexion: an autonomous agent with dynamic memory and self-reflection. [arXiv:2303.11366](https://arxiv.org/abs/2303.11366).
- Sobhy, Dalia, Bahsoon, Rami, Minku, Leandro, Kazman, Rick, 2021. Evaluation of software architectures under uncertainty: A systematic literature review. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* 30 (4), 1–50.

- Sobhy, Dalia, Minku, Leandro, Bahsoon, Rami, Chen, Tao, Kazman, Rick, 2020. Runtime evaluation of architectures: A case study of diversification in IoT. *J. Syst. Softw.* 159, 110428.
- Sumers, Theodore, Yao, Shunyu, Narasimhan, Karthik, Griffiths, Thomas L, 2023. Cognitive architectures for language agents. [arXiv:2309.02427](https://arxiv.org/abs/2309.02427).
- Talebirad, Yashar, Nadiri, Amirhossein, 2023. Multi-agent collaboration: Harnessing the power of intelligent LLM agents. [arXiv:2306.03314](https://arxiv.org/abs/2306.03314).
- Van Der Donckt, M Jeroen, Weijns, Danny, Iftikhar, M Usman, Singh, Ritesh Kumar, 2018. Cost-benefit analysis at runtime for self-adaptive systems applied to an internet of things application. In: ENASE. pp. 478–490.
- Wang, Guanzhi, Xie, Yuqi, Jiang, Yunfan, Mandlekar, Ajay, Xiao, Chaowei, Zhu, Yuke, Fan, Linxi, Anandkumar, Anima, 2023. Voyager: An open-ended embodied agent with large language models. [arXiv:2305.16291](https://arxiv.org/abs/2305.16291).
- Wei, Jason, Wang, Xuezhi, Schuurmans, Dale, Bosma, Maarten, Xia, Fei, Chi, Ed, Le, Quoc V, Zhou, Denny, et al., 2022. Chain-of-thought prompting elicits reasoning in large language models. In: NeurIPS'22. Vol. 35, pp. 24824–24837.
- Xie, Tianbao, Zhou, Fan, Cheng, Zhoujun, Shi, Peng, Weng, Luoxuan, Liu, Yitao, Hua, Toh Jing, Zhao, Junning, Liu, Qian, Liu, Che, et al., 2023. OpenAgents: An open platform for language agents in the wild. [arXiv:2310.10634](https://arxiv.org/abs/2310.10634).
- Yang, Jie, Huang, Gang, Zhu, Wenhui, Cui, Xiaofeng, Mei, Hong, 2009. Quality attribute tradeoff through adaptive architectures at runtime. *J. Syst. Softw.* 82 (2), 319–332.
- Yao, Shunyu, Yu, Dian, Zhao, Jeffrey, Shafran, Izhak, Griffiths, Tom, Cao, Yuan, Narasimhan, Karthik, 2024. Tree of thoughts: Deliberate problem solving with large language models. *Adv. Neural Inf. Process. Syst.* 36.
- Yao, Shunyu, Zhao, Jeffrey, Yu, Dian, Du, Nan, Shafran, Izhak, Narasimhan, Karthik, Cao, Yuan, 2022. React: Synergizing reasoning and acting in language models. [arXiv:2210.03629](https://arxiv.org/abs/2210.03629).
- Zeng, Xin, Wang, Xiaoyu, Zhang, Tengxiang, Yu, Chun, Zhao, Shengdong, Chen, Yiqiang, 2023. GestureGPT: Zero-shot interactive gesture understanding and grounding with large language model agents. [arXiv:2310.12821](https://arxiv.org/abs/2310.12821).
- Zhang, Hongxin, Du, Weihua, Shan, Jiaming, Zhou, Qinrong, Du, Yilun, Tenenbaum, Joshua B, Shu, Tianmin, Gan, Chuang, 2023. Building cooperative embodied agents modularly with large language models. [arXiv:2307.02485](https://arxiv.org/abs/2307.02485).
- Zhao, Andrew, Huang, Daniel, Xu, Quentin, Lin, Matthieu, Liu, Yong-Jin, Huang, Gao, 2024. Expel: Llm agents are experiential learners. In: Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 38, pp. 19632–19642.
- Zhao, Dehai, Xing, Zhenchang, Xia, Xin, Ye, Deheng, Xu, Xiwei, Zhu, Liming, 2023. Seehow: Workflow extraction from programming screencasts through action-aware video analytics. In: 2023 IEEE/ACM 45th International Conference on Software Engineering. ICSE, IEEE, pp. 1946–1957.