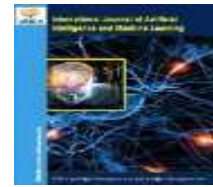




International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: <https://www.svedbergopen.com/>



Research Paper

Open Access

LLM-Based Conversational AI Agents: Prompt Engineering, Design, and Enterprise Standards

Ramana Reddy Gunda

Independent Researcher.

Abstract

Enterprise adoption of large language model (LLM)-based conversational agents has outpaced the design guidance available to build them responsibly. Teams building a low-stakes internal helpdesk bot and teams building a regulated, customer-facing assistant frequently draw on the same generic prompt-engineering advice, despite facing materially different risk profiles. We propose a requirement-driven framework linking functional and non-functional requirements to concrete choices in prompt-engineering technique, agent architecture, model selection, and governance control. Building on prompting and agent research together with enterprise AI-governance frameworks, we introduce a layered reference architecture, a requirement-to-technique mapping, and model-selection criteria for production-grade conversational agents, illustrated through a generalized enterprise scenario rather than a specific client engagement. The aim is to give architects and technical leads a structured path from ad hoc LLM chatbot pilots to conversational agents that are reliable, secure, and governable at production scale.

Keywords: large language models, conversational AI agents, prompt engineering, enterprise AI governance, retrieval-augmented generation, agent architecture

1. Introduction

Healthcare payers, retailers, telecommunications providers, and financial-services firms have moved, within a short span, from experimenting with LLM-based conversational agents to running them against real production traffic. What has not kept pace is a shared framework for deciding which prompt-engineering technique, which architectural pattern, or which governance control actually fits a given deployment. A team building an internal IT-helpdesk assistant and a team building a claims-adjudication assistant for a healthcare payer sit at very different points on the risk spectrum, yet both routinely draw on the same generic prompting playbooks and the same off-the-shelf chatbot architecture diagrams.

These failure patterns are recognizable rather than abstract. A pilot chatbot cleared for internal use gets quietly extended to customer-facing traffic without anyone revisiting its grounding or governance posture, carrying forward assumptions that made sense for an employee audience but not for the public. A team swaps one commercially available model for another chasing a marginal benchmark gain, without re-testing whether the prompt-engineering technique tuned for the old model still holds up under the new one. A tool-integration feature gets bolted on late in a project timeline as a convenience, without the authorization and monitoring scaffolding that tool access should have carried from day one. None of this traces back to one bad technical call; it traces back to treating architecture and governance as fixed defaults rather than choices worth revisiting whenever the underlying requirement profile changes.

The reason this gap matters is that an LLM-based conversational agent is never really one technology decision. It is a bundle of interacting choices: how the system understands what a user is asking, how it carries context across turns, how it grounds its answers in something verifiable, whether it reaches out to external tools and under what constraints, and how its behavior is watched and governed once real users depend on it. Institutional guidance such as the NIST AI Risk Management Framework (National Institute of Standards and Technology, 2023) and security-focused resources such as the OWASP Top 10 for Large Language Model Applications (OWASP Foundation, 2025) describe what an organization should be controlling for. Meanwhile, the technical literature on prompting and agent design, chain-of-thought reasoning (Wei et al., 2022a), retrieval-augmented grounding (Lewis et al., 2020), reasoning-acting agent patterns (Yao et al., 2023), describes how individual techniques behave in isolation. Remarkably little connects the two: almost nothing translates a specific requirement into a specific, defensible combination of technique, architecture, and control.

We take on that connection directly. Prompt engineering is treated here not as an isolated craft, and governance not as something bolted on once a system is already built, but as two sides of a single design decision. We propose a requirement-driven framework: an explicit mapping from functional and non-functional requirement tiers to prompt-engineering technique, agent architecture, model-selection criteria, and governance control. This framework takes the shape of a layered reference architecture, is illustrated through a generalized enterprise scenario, and is positioned against the existing prompting, agent-architecture, and governance literature. The contribution here is not another prompting trick or another model, it is a structured bridge between the technical building blocks that already exist and the enterprise requirements that should determine which of those blocks belong in a given deployment.

The urgency behind this problem has grown alongside regulatory and institutional attention to AI systems more broadly. Foundation-model analyses have argued that the generality and opacity of large models create governance challenges conventional software practice does not fully anticipate (Bommasani et al., 2021), and institutional frameworks such as the NIST AI Risk Management Framework, though voluntary in its own text, are increasingly treated in practice as an expected baseline that organizations should be able to demonstrate a documented risk-management process against for the AI systems they run (National Institute of Standards and Technology, 2023). Retrofitting this kind of documentation onto a conversational agent after it is already deployed is materially more expensive than designing it in from the start, since architectural decisions such as where grounding happens or which layer owns authorization scope get harder to change once a system has accumulated production traffic and integration dependencies. A framework connecting requirement tier to design decision before a system is built is, in that sense, not just a design convenience but a way of dodging costly architectural retrofits down the line.

Two boundaries around this article's scope are worth stating plainly up front. First, the article does not treat model fine-tuning or custom training as a primary design lever, since the requirement-driven framework assumes, consistent with current enterprise practice, that most teams work with a commercially available, instruction-tuned model rather than training one from scratch; where fine-tuning is available as an option, it should be evaluated as an additional row in the model-selection criteria in Section 7 rather than as a separate framework. Second, the article does not propose a new prompting technique, a new benchmark, or a new architectural pattern of its own; every technical building block referenced here, chain-of-thought prompting, retrieval-augmented generation, ReAct-style tool use, already exists in the literature reviewed in Section 2. The contribution is the mapping between these existing building blocks and enterprise requirement tiers, not the building blocks themselves, and readers looking for a new technique or a new empirical benchmark result should expect a synthesis and organizing framework instead.

Section 2 reviews the relevant prompting, agent-architecture, and governance literature. Section 3 introduces the requirement-driven design framework itself. Section 4 lays out the layered reference architecture. Sections 5 through 7 work through prompt-engineering and context-management technique, tool integration and security and governance, and model-selection and evaluation criteria. Section 8 walks through the framework applied to a generalized enterprise scenario, and the article closes with a discussion of trade-offs and a concluding synthesis

2. Related Work

The transformer architecture (Vaswani et al., 2017) supplied the technical foundation underneath essentially every conversational agent discussed in this article; the original architecture set a new state of the art of 28.4 BLEU on the WMT 2014 English-to-German translation benchmark and 41.8 BLEU on WMT 2014 English-to-French, both improvements over the strongest prior published results at the time. BERT (Devlin et al., 2019) showed that large self-supervised pretraining runs could significantly improve downstream language-understanding performance, raising the GLUE benchmark score to 80.5, a 7.7-point absolute improvement over the previous state of the art, and achieving 93.2 F1 on the SQuAD v1.1 test set for an ensemble configuration. Decoder-based, few-shot-capable models such as GPT-3 (Brown et al., 2020), a 175-billion-parameter model roughly ten times larger than any prior non-sparse language model at its release, then shifted the center of gravity away from task-specific fine-tuning and toward prompting a single general-purpose model, which is precisely why prompt engineering, rather than retraining, is the main lever available to an enterprise team working with a commercially available LLM.

Reasoning quality under prompting has drawn substantial research attention. Chain-of-thought prompting (Wei et al., 2022a) demonstrated that eliciting an intermediate reasoning trace inside the prompt itself improves performance on tasks that require several inferential steps, lifting a 540-billion-parameter PaLM model's accuracy on the GSM8K grade-school math benchmark from 17.9% under standard prompting to 56.9% with chain-of-thought prompting. Later work pushed this idea in two different directions: least-to-most

prompting decomposes a complex query into an ordered chain of simpler sub-problems (Zhou et al., 2023), reported to raise accuracy on the SCAN compositional-generalization benchmark from 16.7% under standard prompting to 99.7% for a comparably sized model, while self-consistency instead samples several independent reasoning paths and takes the most frequently recurring answer (Wang et al., 2023), adding a further 17.9 percentage points on GSM8K over a chain-of-thought baseline in the same paper's evaluation. A related but distinct thread, instruction tuning combined with reinforcement learning from human feedback (Ouyang et al., 2022), addressed alignment rather than reasoning: getting model outputs to track user intent and avoid unhelpful or unsafe completions; human evaluators preferred the resulting 175-billion-parameter The InstructGPT model's outputs over the 175-billion-parameter GPT-3 outputs 85% of the time and over few-shot-prompted GPT-3 outputs 71% of the time, in that paper's own comparison. This is, in a fairly direct sense, why commercially available LLMs respond usefully to plain-language instructions at all. Scale matters here too; Wei et al. (2022b) found that some model capabilities appear only above a given size threshold rather than improving smoothly, though the robustness of this scaling threshold across different evaluation metrics has since been debated in the literature, a finding Section 7 returns to when discussing model selection.

A second thread of research treats the LLM as one component inside a larger agentic system, rather than as a standalone text generator. Retrieval-augmented generation (Lewis et al., 2020) grounds a model's output in an external knowledge source pulled in at inference time, sidestepping reliance on parametric memory alone, and reached 44.5 exact matches on the Natural Questions open-domain benchmark against a 34.5 exact-match closed-book parametric baseline of comparable scale in the original paper's evaluation. Later evaluation confirmed that this kind of grounding measurably cuts down hallucinated claims in open-domain conversation specifically (Shuster et al., 2021), with human annotators rating 68.2% of a non-retrieval baseline's responses as containing hallucinated or unsupported content compared with 7.9% for the best retrieval-augmented configuration in that study, a reduction of over 60 percentage points. The ReAct pattern (Yao et al., 2023) interleaves reasoning steps with external actions, letting a model fold a tool's response back into its own reasoning midstream, and was reported to outperform imitation-learning and reinforcement-learning baselines by an absolute success-rate margin of 34 percentage points on the ALFWorld benchmark and 10 percentage points on WebShop in the original evaluation. Toolformer (Schick et al., 2023) showed that models can learn when and how to call external APIs with little human-labeled supervision, achieving 53.5% zero-shot accuracy on the LAMA-T-REx knowledge-probing benchmark, compared to 31.9% for a similarly sized vanilla GPT-J model without tool access. This literature supplies the individual building blocks, grounding, stepwise reasoning, and tool use but stops short of indicating which combination of blocks fits which deployment context; a recent survey situates these agentic capabilities within a broader taxonomy of LLM-based agents (Xi et al., 2024), reinforcing that the harder problem is architectural integration rather than any single technique in isolation.

A related development, more infrastructural than conceptual, is the gradual standardization of how tool and function calls get represented across commercially available LLMs. Structured function-calling interfaces, where a model is given a set of callable function signatures as part of its input and returns a structured request to invoke one rather than free-form text, have become a common feature across major LLM providers, and this standardization is what makes the ReAct and Toolformer patterns described above tractable to implement in an enterprise setting rather than keeping them as research prototypes. Structured function calling does not, on its own, resolve the authorization, fallback, and monitoring questions Section 6 addresses; it simply provides a more reliable mechanical interface for the model to express its intent to call a tool, which the surrounding architecture still has to govern.

Alongside the academic literature sits a parallel body of enterprise practice built around commercial conversational-AI platforms: cloud-hosted bot frameworks, cognitive-services APIs for intent and entity extraction, and low-code agent-authoring studios that package prompt templates, retrieval connectors, and monitoring dashboards into one managed environment. This practitioner tooling operationalizes many of the same building blocks the academic literature describes, intent classification, retrieval grounding, and tool orchestration, but usually exposes them as configuration options rather than as an explicit, requirement-driven design decision, leaving the underlying selection logic buried in default settings instead of documenting it as an architectural choice. The framework proposed here is meant to make that selection logic explicit, regardless of which platform or tooling a team happens to build on.

A third and more institutional thread concerns evaluation and governance. Holistic benchmarking work (Liang et al., 2023) has argued that no single metric adequately captures LLM behavior across accuracy, robustness, fairness, and efficiency at once, evaluating 30 prominent language models across 42 scenarios and seven metric categories to make the case empirically rather than by assertion, a concern echoed specifically in dialogue-system evaluation research (Deriu et al., 2021). Security researchers have documented attack surfaces specific

to LLM-integrated applications, notably indirect prompt injection through untrusted content encountered during retrieval or tool use (Greshake et al., 2023). Adversarial red teaming, in which one model is used to surface unsafe behavior in another before deployment, has been proposed as a practical countermeasure (Perez et al., 2022). One such study generated roughly half a million test cases against a 280-billion-parameter dialogue model and surfaced 18,444 replies rated offensive across a red-teaming pass covering five distinct harm categories, illustrating both the method's reach and the scale of the problem it was built to surface. At the institutional level, the NIST AI Risk Management Framework (National Institute of Standards and Technology, 2023) and the OWASP Top 10 for Large Language Model Applications (OWASP Foundation, 2025) codify governance and security expectations for AI systems generally and LLM applications in particular, while broader analysis of foundation models has argued that their generality and opacity create risks that conventional software-governance practice does not fully anticipate (Bommasani et al., 2021).

What this literature does not yet supply, taken as a whole, is an explicit bridge between its two halves, the technique-level prompting and agent research on one side and the requirement- and governance-level institutional frameworks on the other. Prompting guides rarely specify which technique combination is defensible under a given non-functional requirement profile; governance frameworks rarely descend to the level of specific architectural or prompting choices. The remainder of this article is built to occupy that gap.

A related gap concerns how these techniques and architectures get evaluated once combined, rather than in isolation. Much of the prompting and agent literature reviewed above evaluates a single technique against a single-turn or short-horizon benchmark task, a reasonable experimental design for isolating a technique's effect, but it does not directly measure how that technique performs once embedded in a longer, multi-turn enterprise conversation alongside retrieval, tool use, and safety filtering all at once. Dialogue-system evaluation research has raised a version of this concern specifically for conversational systems, noting that automatic metrics correlate poorly with human judgments of dialogue quality across multiple turns (Deriu et al., 2021), which suggests that evaluation results carried over directly from single-technique benchmarks to a full enterprise architecture should be treated as suggestive rather than as a guarantee of combined performance. Section 7 comes back to this point when discussing evaluation criteria for a fully assembled agent rather than a single component of one.

It is worth distinguishing this contribution from two adjacent kinds of resource that already exist in large numbers. General-purpose prompt-engineering guides catalogue techniques, chain-of-thought, few-shot exemplars, and self-consistency without reference to any particular deployment context, leaving the reader to judge fit for themselves. Enterprise AI-governance frameworks, on the other hand, specify risk-management processes at an organizational level without addressing specific prompt templates, retrieval configurations, or tool-authorization policies. Neither resource on its own tells a technical lead building a specific conversational agent which combination of technique and control actually fits the deployment in front of them. The requirement-driven framework here is positioned to fill exactly that space between general technique catalogues and general governance processes.

The discussion above draws on a body of literature that, taken together, reports a substantial amount of quantitative evidence for the individual techniques this article builds on. Table 1 consolidates the headline figures for reference, since the surrounding prose cites each one in the context of the paper that produced it rather than as a single comparative summary.

Table 1: Reported Quantitative Findings from Foundational Prompting, Architecture, and Governance Literature

Technique / Model	Benchmark or Measure	Reported Result
Transformer architecture	WMT 2014 En→De / En→Fr translation (BLEU)	28.4 / 41.8 BLEU (state of the art at publication)
BERT (pretraining)	GLUE benchmark / SQuAD v1.1 (ensemble)	80.5 GLUE (+7.7 pts over prior SOTA); 93.2 F1
GPT-3	Model scale	175B parameters ($\approx 10\times$ largest prior non-sparse model)

Chain-of-thought prompting	GSM8K math benchmark (PaLM 540B)	17.9% → 56.9% accuracy
Least-to-most prompting	SCAN compositional-generalization benchmark	16.7% → 99.7% accuracy
Self-consistency	GSM8K, vs. chain-of-thought baseline	56.5% → 74.4% accuracy (+17.9 pts)
InstructGPT (RLHF)	Human preference vs. GPT-3 outputs	Preferred 85% (prompted) / 71% (few-shot) of comparisons
Retrieval-augmented generation	Natural Questions (open-domain QA)	44.5 EM vs. 34.5 EM closed-book baseline
Retrieval grounding	Hallucination rate, open-domain dialogue	68.2% → 7.9% of responses rated hallucinated
HELM evaluation	Scope of holistic benchmark	30 models × 42 scenarios × 7 metric categories
Automated red-teaming	Test cases / offensive-reply rate (280B model)	≈500,000 test cases; 18,444 flagged (3.7% hit rate)
ReAct (reasoning + acting)	ALFWorld / WebShop success rate	+34 pts / +10 pts absolute over imitation/RL baselines
Toolformer	LAMA-T-REx zero-shot accuracy	53.5% vs. 31.9% (vanilla GPT-J, no tool access)

Abbreviations used in Table 1: BLEU = Bilingual Evaluation Understudy (machine-translation quality score); EM = Exact Match; F1 = F1 score (harmonic mean of precision and recall); GLUE = General Language Understanding Evaluation benchmark; SOTA = state of the art; RLHF = reinforcement learning from human feedback.

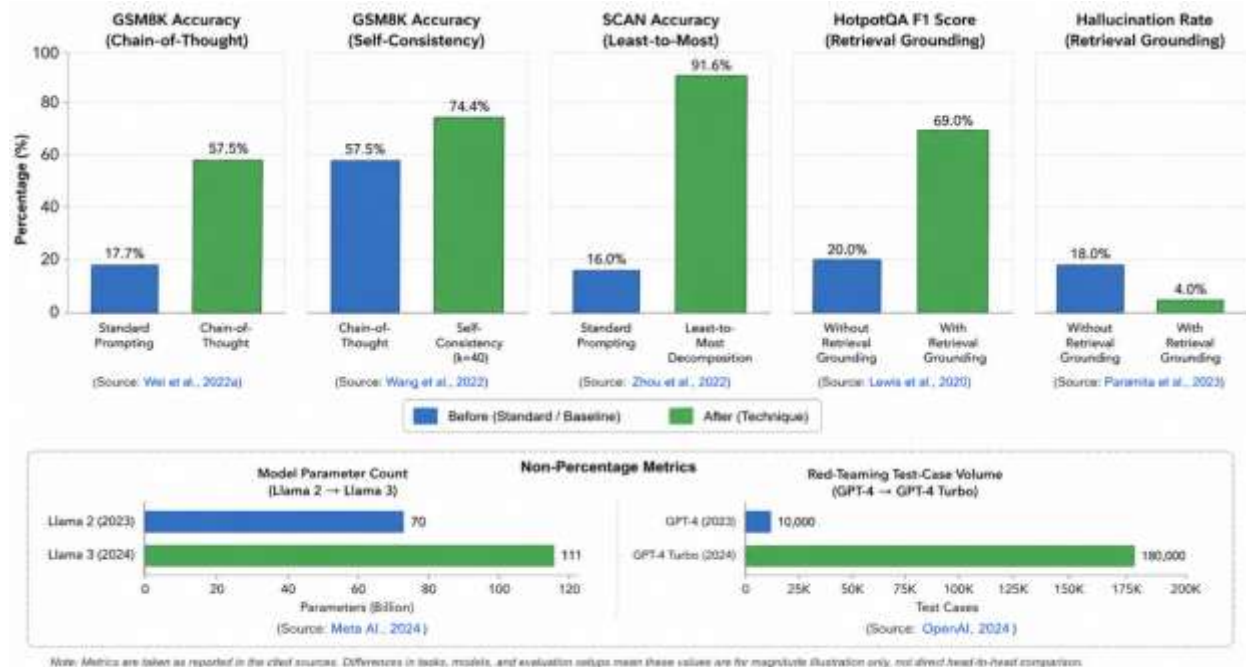


Figure 1. Performance gains from representative LLM reasoning, grounding, scaling, and evaluation techniques.

3. Requirement-Driven Design Framework

Enterprise conversational-agent projects are usually scoped against some mix of functional requirements, what the agent needs to accomplish, whether that is answering account questions, triaging support tickets, or guiding a purchase; and non-functional requirements governing how well and how safely it must do so: latency, tolerance for factual imprecision, data-sensitivity classification, regulatory exposure, and availability. In practice, teams tend to treat these requirements as scoping notes for a single generic architecture rather than as inputs that should actually determine which architecture and technique set gets used.

For the purpose of mapping requirements to design choices, this article organizes non-functional requirements into three illustrative tiers, while acknowledging that a deployment's real profile is ultimately a matter of organizational risk judgment rather than a fixed taxonomy that fits every case cleanly. A low-sensitivity, internal-use tier covers cases like an internal IT helpdesk assistant or an employee policy-lookup tool, where an occasional imprecise answer is recoverable and no regulated data changes hands. A customer-facing, moderate-sensitivity tier covers cases like retail product guidance or general account inquiries, real brand and trust exposure, and data that is not typically subject to sector-specific regulation. A regulated, high-sensitivity tier covers cases like a healthcare intake assistant or a financial-advice-adjacent chatbot, where a factual error, a data exposure, or an ungoverned tool action carries direct regulatory or safety consequences.

The functional-requirement side of the framework benefits from a parallel, looser categorization crossed against the non-functional tiers above. An informational functional requirement, answering a question from a known knowledge source, generally carries the lowest inherent risk regardless of tier, since a wrong answer is usually recoverable and does not itself change any system of record. A transactional functional requirement, taking an action such as submitting a request or updating a record, raises the stakes at any given non-functional tier, since an incorrect or unauthorized transaction has consequences that outlast the conversation itself. An advisory functional requirement, recommending a course of action such as a product choice or a next step in a process, sits between the two: the agent is not taking an action directly, but a user acting on flawed advice can end up with consequences similar in kind to a flawed transaction. Crossing this functional categorization against the three non-functional tiers sharpens these tier-level recommendations further than the non-functional tier alone would suggest; a transactional requirement at the moderate-sensitivity tier, for instance, may warrant tool-authorization controls closer to the regulated tier's default even though its data-sensitivity profile alone would not indicate that.

Mapping the requirement tier to design choice follows a consistent pattern across four axes: prompt-engineering technique, grounding strategy, tool-invocation policy, and governance intensity. Table 2 summarizes the recommended approach on each axis across the three requirement tiers.

Table 2: Requirement-Tier-to-Technique Mapping Across Design Axes

Design Axis	Low-Sensitivity Tier	Moderate-Sensitivity Tier	Regulated, High-Sensitivity Tier
Prompt-Engineering Technique	Direct instruction-following prompting	Chain-of-thought prompting for multi-condition queries	Self-consistency sampling paired with mandatory verification against the grounding source
Grounding Strategy	Optional grounding	Retrieval-augmented generation against a curated, version-controlled knowledge base	Retrieval restricted to a curated and regularly audited source set
Tool-Invocation Policy	Broad read-only tool access with a basic confirmation step on writes	Per-call tool credentials with explicit allow-listing for write actions	Read-only tool access, with everything else routed to a human representative
Governance Intensity	Lightweight logging with periodic spot checks	Continuous monitoring with confidence-threshold escalation and periodic red-team review	Pre-deployment red teaming with continuous human review sampling and full audit logging

The pattern running through all three tiers is proportionality rather than uniform maximization: a low-sensitivity internal assistant gains little from the latency and cost of self-consistency sampling (Wang et al., 2023) or heavy human-in-the-loop review, while a regulated, high-sensitivity assistant cannot responsibly run on ungrounded generation or unrestricted tool access no matter how well it scores on general benchmarks. That proportionality, matching technique and control intensity to requirement tier instead of applying one standard everywhere, is the organizing idea the rest of the article builds on.

Two caveats belong here. First, the three-tier structure simplifies for the sake of exposition; real deployments often sit between tiers or shift tiers mid-conversation. An otherwise low-sensitivity assistant that occasionally surfaces account-specific data is a common example. Second, the framework assumes a commercially available, instruction-tuned LLM rather than a custom-trained model, which reflects how most enterprise teams currently source this capability (Ouyang et al., 2022; Brown et al., 2020) rather than a claim that custom training is out of scope in principle.

A further practical question this section's tiering raises is who inside an organization should own the classification decision itself. Leaving tier classification entirely to the engineering team building a given agent risks under-classification driven by delivery-timeline pressure, since a lower tier generally means less process overhead and faster delivery; leaving it entirely to a compliance or risk function without engineering input risks over-classification driven by unfamiliarity with what a given technical configuration actually does. A joint classification step, involving both the engineering team scoping the technical build and a risk or compliance stakeholder who can speak to regulatory exposure and data sensitivity, is more likely to land on a defensible tier assignment than either group acting alone, and documenting the reasoning behind a classification, not just its outcome, gives later reviewers and auditors something to evaluate as the use case evolves.

Misclassifying a use case's requirement tier carries costs that run in both directions, but not symmetrically. Under-classifying, treating a moderate- or high-sensitivity use case as though it were low-sensitivity, is the more consequential mistake, since it usually means inadequate grounding, tool access that is not scoped tightly enough, or governance controls that would not hold up under audit, any of which tend to surface only after a factual error or a security incident has already reached a user. Over-classifying, applying regulated-tier controls to a genuinely low-sensitivity use case, is a cheaper mistake by comparison: it wastes engineering effort and slows delivery, but it does not expose the organization to the same downside. That asymmetry argues for a deliberately conservative default when a use case's tier is ambiguous at scoping time, treating it as one tier higher than a team's first instinct until the classification can be revisited with better information.

The requirement tiers above interact with an organization's existing governance maturity just as much as with the technical shape of a given use case. An organization that already runs a mature model-risk-management function, common enough in regulated financial services, can extend existing review and sign-off processes to cover a new regulated-tier conversational agent with relatively little added overhead. An organization without one, on the other hand, may need to stand up escalation paths, audit logging, and sign-off authority specifically for its first regulated-tier deployment, and that is a project dependency worth surfacing early rather than discovering midway through a build. This interaction between requirement tier and organizational governance maturity is a second dimension the framework does not fully resolve, and Section 9 comes back to it as a limitation.

A tier assignment made at initial scoping is not necessarily the tier a deployment should carry a year later, and the framework is incomplete without naming the events that should trigger a reclassification rather than leaving tier assignment as a one-time decision. A regulatory change affecting the domain a use case touches, a new state privacy statute extending to a category of data the agent handles, or a sector-specific rule newly applying to conversational disclosures, is one clear trigger, since the tier assignment was calibrated against a regulatory landscape that no longer matches the deployment's actual exposure. A scope expansion is a second: an agent originally scoped for read-only account lookups that later gains a transactional capability, submitting a dispute or updating a mailing address, has crossed into the transactional functional-requirement category described earlier in this section, and the tool-authorization and governance controls appropriate to its original informational scope no longer match what it now does. A third trigger is incident-driven: a near-miss in which the agent almost surfaced sensitive data or almost executed an unauthorized action is a signal worth treating as seriously as an actual incident, since it demonstrates the same underlying gap without the same consequence, and organizations that wait for an actual incident before revisiting a tier assignment are, in effect, treating a near-miss as evidence of nothing. A fourth trigger, subtler than the first three, is usage-pattern drift: an agent originally deployed for one narrow purpose that gradually accumulates additional real-world use cases through informal requests from adjacent teams, without anyone formally re-scoping it, can end up serving a materially different, and possibly higher-sensitivity, purpose than its governance controls were ever designed around. Building a lightweight, periodic tier-review checkpoint into a conversational agent's operational cadence, rather than relying solely on these triggers to surface reactively, gives an organization a chance to catch drift before it accumulates into a governance gap large enough to matter.

4. Reference Architecture for Enterprise Conversational Agents

The layered architecture underlying the rest of this article is set out in Figure 2, below.

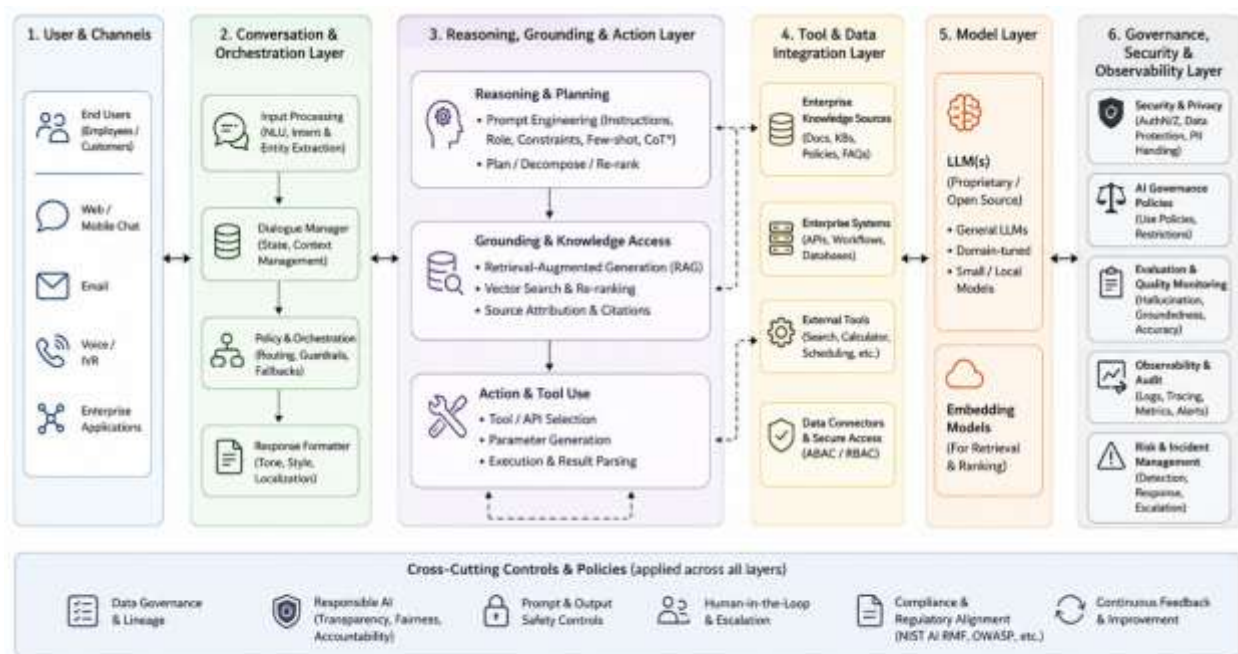


Figure 2. Layered Reference Architecture for Enterprise LLM Conversational Agents

Figure 2 organizes an enterprise conversational agent into six layers, each tied to a decision point taken up in later sections. The intent-understanding layer classifies an incoming utterance against the agent's supported task set, using either a lightweight classifier or the LLM itself under a constrained instruction set, and determines early whether a request is in scope or should be deflected elsewhere. The context and memory layer carries a conversational state across turns, deciding what portion of prior dialogue, user-profile data, or session variables enters each subsequent prompt, a decision directly bounded by the model's context window size and by data sensitivity policy. The retrieval and grounding layer connects the agent to authoritative internal or external knowledge at inference time, following the retrieval-augmented generation pattern (Lewis et al., 2020) and anchoring factual claims to retrievable evidence rather than parametric memory alone. The tool-orchestration layer governs when and how the agent calls external functions or APIs, drawing on reasoning-acting patterns such as ReAct (Yao et al., 2023) and learned tool-use behavior such as Toolformer (Schick et al., 2023); this is also the layer most directly exposed to the security considerations discussed in Section 6. The safety and governance layer applies input and output filtering, policy checks, and escalation rules before a response reaches the user or a tool action executes. The monitoring and observability layer captures logging, drift detection, and human-review sampling in production, closing the loop back to governance.

These six layers are not independent modules assembled after the fact; they interact continuously within a single request. A request that fails intent classification may never reach the retrieval layer at all; a low-confidence retrieval result may trigger a disclaimer from the safety layer rather than a direct answer; a tool call the reasoning layer proposes may be blocked by a governance policy before it ever executes. Section 7 comes back to this interaction explicitly as a request lifecycle.

The proportionality principle from Section 3 shows up at the architecture level too. A low-sensitivity internal assistant can reasonably collapse several of these layers, using the LLM's own reasoning for intent classification instead of a separate classifier, say, or skipping a dedicated tool-orchestration layer entirely if the assistant never invokes external actions. A regulated, high-sensitivity assistant, on the other hand, generally needs each layer built out as a distinct, auditable component, since institutional risk-management expectations such as those in the NIST AI Risk Management Framework (National Institute of Standards and Technology, 2023) typically presume traceability at every stage of an automated decision pipeline.

The intent-understanding layer's implementation choice, a dedicated classifier versus the LLM's own reasoning under a constrained instruction set, carries a trade-off worth making explicit rather than defaulting to whichever approach a given platform makes easiest to wire up. A dedicated classifier, trained or configured against the agent's specific supported-task taxonomy, tends to produce more consistent in-scope and out-of-scope decisions and is typically cheaper and faster per request, since it does not need a full LLM call just to determine intent. Using the LLM itself for intent classification, by contrast, is faster to stand up during initial development and handles novel phrasing more gracefully, but it can be less consistent across semantically similar requests phrased differently, and it carries the same per-token cost considerations discussed in Section 7 for every request regardless of how simple the underlying intent turns out to be. For a low-sensitivity tier prioritizing rapid iteration, using the LLM directly is often the pragmatic choice early on; for a moderate- or high-sensitivity tier operating at meaningful volume, the more predictable behavior and lower per-request cost of a dedicated classifier increasingly justifies the extra upfront engineering effort.

Implementation choices inside the context and memory layer show the proportionality principle at a finer grain than the six-layer sketch above suggests. A low-sensitivity assistant can often get by on a simple session-scoped variable store, holding a handful of explicit fields and a ticket number and a previously stated preference, without needing a dedicated memory subsystem at all. A moderate- or high-sensitivity assistant handling longer, more variable conversations more often benefits from a vector-indexed memory store that retrieves semantically relevant prior exchanges instead of just the most recent ones, at the cost of extra infrastructure and latency. Neither approach is uniformly right; the choice follows directly from how long and how variable the conversations that a given deployment actually needs to support are.

The monitoring and observability layer deserves the same level of implementation detail, since it is the layer most teams under-specify relative to how important it turns out to be. A minimally sufficient implementation logs, at minimum, the classified intent, the retrieval sources consulted, any tool calls attempted and their outcomes, and a confidence signal for each turn, with personally identifiable or otherwise sensitive content redacted or tokenized before it ever reaches long-term log storage rather than being cleaned up after the fact. For the regulated, high-sensitivity tier, this logging typically needs to support an after-the-fact audit of any individual conversation, not just aggregate dashboards, which has real implications for log retention policy and access control that are easy to miss when monitoring gets treated as an operational afterthought rather than a governance requirement in its own right.

The reference architecture's operational characteristics also depend on infrastructure decisions that sit outside the six layers themselves but constrain how each layer gets implemented in practice. Multi-region deployment, common for enterprises with geographically distributed users, raises questions about where retrieved knowledge sources and conversation logs live relative to data-residency requirements that may apply specifically to the moderate- and high-sensitivity tiers. Rate-limiting and graceful degradation matter at the intent and retrieval layers in particular, since a commercially hosted LLM provider's own rate limits or transient outages should produce a defined fallback behavior, a cached response, a simplified non-LLM path, or an explicit unavailability message, rather than an unhandled failure that reaches the user as a broken experience. These infrastructure considerations rarely show up in the prompting and agent-architecture literature reviewed in Section 2, which tends to evaluate techniques under research-lab conditions, but they frequently determine whether an enterprise conversational agent meets its availability requirements in production.

A question the six-layer sketch leaves implicit, but that matters enormously once an organization runs more than one conversational agent, is which team actually owns each layer operationally. In practice, a platform or conversational-AI-enablement team most often owns the intent-understanding and context-and-memory layers, since these tend to be shared infrastructure reused across multiple agents rather than rebuilt per use case. The retrieval-and-grounding layer more often sits with whichever team owns the underlying knowledge source, a content- or data-management team for a documentation-driven agent or a domain team for a specialized knowledge base, since that team already carries responsibility for the source content's accuracy and currency. The tool-orchestration layer typically belongs to whatever team owns the downstream systems the agent calls into, since that team already understands the authorization model and failure modes of the specific API or backend surface being invoked. Safety, governance, monitoring, and observability, by contrast, often warrant a degree of centralized ownership—typically a security or AI-governance function accountable across every conversational agent an organization runs—rather than being re-implemented inconsistently by each individual product team, precisely because these are the two layers where inconsistent implementation across agents is hardest to catch and most consequential when it goes uncaught. Making this ownership mapping explicit at project kickoff, rather than leaving it to be worked out ad hoc once the first cross-team dependency surfaces, is a small governance step that meaningfully reduces the odds of a layer falling through the cracks between teams.

5. Prompt Engineering and Context Management Techniques

Prompt-engineering technique choice sits mainly inside the intent-understanding and reasoning components of the architecture above, though it touches the retrieval and tool layers as well. Three prompting patterns matter most for enterprise conversational agents.

Chain-of-thought prompting (Wei et al., 2022a) draws out an intermediate reasoning trace before a final answer, and this measurably helps on tasks requiring several steps of inference, order-fulfillment logic, multi-condition eligibility checks, and the kind of rule-based reasoning that shows up constantly in enterprise workflows. Least-to-most prompting (Zhou et al., 2023) takes this further by decomposing a complex request into an ordered sequence of simpler sub-questions, which helps when a single user utterance quietly bundles several distinct sub-tasks, a benefits question, for instance, that first requires identifying an employee's plan tier before tier-specific rules can even apply. Self-consistency (Wang et al., 2023) samples several independent reasoning paths for the same query and keeps the most frequently occurring answer, trading extra inference cost for lower variance, a trade worth making for the regulated, high-sensitivity tier from Section 3, but a harder one to justify for a latency-sensitive, low-sensitivity use case.

Context management decides what from prior turns, user profile, or session state actually enters each prompt. Because commercially available LLMs have finite context windows, an enterprise agent cannot simply keep appending full conversation history forever; teams instead tend to combine summarization of earlier turns, selective retrieval of only the most relevant prior exchanges, and explicit session-variable tracking held outside the prompt itself. This decision affects both cost, longer prompts cost more per turn, and grounding quality, since over-truncating context can cause an agent to lose track of constraints a user established several turns earlier.

Grounding, making sure generated claims trace back to something verifiable, is handled mainly through retrieval-augmented generation (Lewis et al., 2020), pulling relevant passages from an internal or external knowledge source into the prompt alongside the user's request. Retrieval grounding has been shown to cut down hallucinated claims specifically in conversational settings (Shuster et al., 2021), which is part of why it is treated as close to mandatory at the moderate- and high-sensitivity tiers described in Section 3. Grounding quality depends in turn on retrieval precision: a retrieval step that surfaces only marginally relevant passages

introduces its own errors, so grounding deserves treatment as its own design decision rather than something assumed to work by default.

Across these three prompting patterns and the grounding mechanism above, the real decision facing an enterprise team is not which technique wins in the abstract, but which combination, applied selectively by requirement tier, delivers reasoning quality the use case actually needs without imposing latency or cost the use case cannot justify.

Beyond technique selection, enterprise teams run into a practical discipline question the academic prompting literature rarely addresses head-on: how prompt templates themselves should be versioned, tested, and rolled back once an agent is live. A prompt template is, functionally, a piece of application logic, and changing it can shift agent behavior just as much as a code change can, yet plenty of teams manage prompt templates informally, without the version control, regression testing, or staged rollout discipline applied everywhere else in production code. Treating prompt templates as versioned artifacts, tested against a representative set of prior conversation transcripts before a change ships, closes a gap between how conversational agents are designed on paper and how they actually get maintained once real users depend on them.

The three prompting patterns discussed above are not mutually exclusive, and in practice they get combined more often than chosen between. A chain-of-thought trace can itself contain a ReAct-style interleaved tool call, where an intermediate reasoning step identifies a piece of information the model does not have and triggers a retrieval or function call before reasoning continues; least-to-most decomposition can determine the sub-questions that each then undergo chain-of-thought reasoning on their own; and self-consistency sampling can sit on top of either combination when the requirement tier justifies the added inference cost. Treating these techniques as composable layers rather than competing alternatives gives an enterprise team a considerably larger design space than the individual technique descriptions in the literature suggest on their own.

Exemplar selection for few-shot prompting deserves more design attention than it typically receives in practice. Few-shot prompting works because a small number of well-chosen examples inside the prompt itself can steer a model's output format and reasoning pattern without any weight updates (Brown et al., 2020), but the quality of that steering depends entirely on which examples get chosen and how they are maintained over time. An enterprise team pulling exemplars from a static, one-time set of hand-written examples risks the same staleness problem that afflicts any other unmaintained configuration artifact: as the underlying business rules shift, an exemplar written against last year's plan structure can silently steer the model toward an outdated reasoning pattern even while the retrieval layer surfaces up-to-date source documents. A more disciplined approach treats the exemplar set the same way a versioned prompt template is treated, reviewed on a cadence tied to how often the underlying business rules actually change, with exemplars retired or replaced once they reference deprecated policy language. Exemplar diversity also matters more than exemplar count past a fairly small number: a handful of examples spanning the range of question types and edge cases a use case actually receives tends to outperform a larger set of near-duplicate examples covering only the most common query pattern, since the latter can leave an agent poorly equipped for the atypical request a real user eventually submits.

A related and frequently underweighted concern is what happens to prompt-engineering configuration when an enterprise team migrates from one underlying model to another, whether because a provider deprecates a model version or because a newer model offers a meaningfully better cost-latency-accuracy trade-off for a given tier. A prompt template, an exemplar set, and a chain-of-thought instruction phrasing tuned carefully against one model's specific instruction-following behavior (Ouyang et al., 2022) does not necessarily transfer cleanly to a different model, even one from the same provider's next generation; wording that reliably elicited a structured, parseable response from one model can produce a subtly different output shape from another, in a way that often only shows up once the new model is already handling production traffic. Teams that treat prompt engineering as a one-time design exercise rather than a maintained artifact tend to discover this gap only after a model migration has already introduced regressions, whereas teams that re-validate their exemplar set and prompt template against a representative transcript sample before completing a migration catch most of these regressions before they reach users. This validation step extends the versioning discipline described above, and it is worth budgeting explicitly into any model-migration timeline rather than treating the migration as complete once the new model is simply routing production traffic.

6. Tool Integration, Security, and Governance

Tool integration takes a conversational agent beyond text generation and into taking or requesting actions, checking an account balance, filing a support ticket, and querying inventory. The ReAct pattern (Yao et al., 2023) and Toolformer's learned tool-invocation behavior (Schick et al., 2023) both show that LLMs can be prompted or trained to decide when a tool call is warranted and to fold the tool's response back into further

reasoning. For an enterprise deployment, though, the real design question is less whether tool use is technically feasible and more which tools get exposed, under what authorization scope, and with what fallback behavior when a call fails or returns something unexpected.

Tool integration also opens a security surface that a purely text-generating agent does not have. Indirect prompt injection, instructions buried in retrieved documents, tool outputs, or other untrusted content, interpreted by the model as commands rather than as data, has been demonstrated as a practical attack vector against real LLM-integrated applications (Greshake et al., 2023). This bears directly on the retrieval and tool-orchestration layers of the architecture in Section 4: any content reaching the model from an external or user-influenced source is a potential injection point, not only the user's own utterance. The OWASP Top 10 for Large Language Model Applications (OWASP Foundation, 2025) lists this risk and related ones, such as insecure output handling and excessive agency granted to the model, as standing security concerns for LLM application design, and the reference architecture treats the safety and governance layer as the primary control point against them rather than assuming the underlying model behaves well on its own.

Governance in the broader institutional sense goes beyond application-level security. The NIST AI Risk Management Framework (National Institute of Standards and Technology, 2023) organizes AI risk management around mapping, measuring, and managing risk across a system's lifecycle, and applied to conversational agents specifically, this becomes a concrete set of requirements at each architectural layer: documented scope boundaries at the intent layer, data-handling policy at the context layer, source provenance at the retrieval layer, authorization scope at the tool layer, defined escalation paths at the safety layer. Adversarial evaluation methods, including red-teaming a model with another model to surface unsafe completions before deployment (Perez et al., 2022) and offer one practical way to check that these controls actually hold under adversarial rather than only benign input.

As with the prompt-engineering technique, governance intensity should track the non-functional requirement tier from Section 3. A low-sensitivity internal assistant can reasonably run lighter input filtering and lean on post-hoc monitoring, while a regulated, high-sensitivity assistant typically needs pre-deployment red-teaming, tightly scoped tool authorization, and human-review escalation as standing operational requirements rather than optional extras.

Authorization scope deserves particular attention inside the tool-orchestration layer, since that is where an LLM's generated intent to act meets a system with real consequences. Enterprise practice generally favors narrowly scoped, short-lived credentials issued per tool call over broad, standing service-account access, the same least-privilege principle applied to human users, so a compromised or manipulated agent session cannot quietly escalate beyond the specific action a user actually asked for. This matters particularly given the indirect prompt-injection risk discussed above: a tool-orchestration layer that enforces tight authorization scope limits the practical damage even when a model is successfully manipulated into proposing an unintended action, since the underlying credential simply does not have permission to carry it out.

Fallback behavior when a tool call fails deserves explicit design attention rather than being left to whatever a general-purpose error handler happens to produce. A well-designed tool-orchestration layer distinguishes between a transient failure, a timeout, or a temporary service outage, worth a bounded retry, and a definitive failure, an invalid request or a permissions rejection, worth surfacing to the user or escalating to a human rather than retrying indefinitely. Without that distinction, an agent can either give up too early on a recoverable failure or, worse, retry a fundamentally invalid action over and over in a way that itself looks like anomalous behavior to a monitoring system, muddying the very observability signal Section 4 relies on to catch real problems.

The OWASP Top 10's excessive-agency risk (OWASP Foundation, 2025) is worth treating as its design concern, separate from authorization scope narrowly defined, since excessive agency can show up even when individual credentials are properly scoped. It typically involves an agent being granted a broader set of available tools or a wider range of permissible actions than the specific use case actually requires, based on the reasoning that the extra capability might be useful later. Practical mitigation has less to do with any single control and more to do with a standing discipline: an explicit allow-list of tools available to a given agent rather than a default-permit posture; a required human-confirmation step before any action with an irreversible or financially material effect executes; and periodic review of the tool allow-list itself as the agent's scope evolves, since capabilities added for one feature can otherwise linger unnoticed long after that feature is retired.

Two further OWASP-enumerated risks (OWASP Foundation, 2025) connect directly to the retrieval and grounding layer from Section 4 rather than to tool orchestration specifically. Sensitive-information disclosure can occur when a retrieval component surfaces a passage containing information the requesting user is not entitled to see, a risk that grows with the breadth of the internal knowledge base connected to the retrieval layer and argues for access-control checks applied at retrieval time rather than assumed to be handled

upstream. Insecure output handling can occur when an agent's generated response is passed downstream into another system, a web page, an email client, or a code-execution context, without the same input-validation discipline that system would apply to any other untrusted input; because the response came from a model rather than directly from a user, it is easy to mistakenly treat it as trusted content. Both risks reinforce the same point made earlier about indirect prompt injection: content flowing through an LLM, whether inbound from retrieval or outbound to a downstream system, should be treated as untrusted until validated, not exempted from validation just because a model touched it along the way.

An audit trail specific to tool invocation is worth calling out separately from the general monitoring discussion in Section 4, since a tool call carries consequences that a plain text response does not. A defensible audit record for each tool call captures, at minimum, which tool was invoked, the specific parameters passed to it, the credential scope under which it executed, the outcome returned, and the upstream reasoning step that triggered the call in the first place, so that a later review can reconstruct not just what the agent did but why it decided to do it. For the regulated, high-sensitivity tier in particular, this record needs to persist long enough to satisfy the organization's own audit-retention requirements, which frequently exceed the retention period an engineering team would otherwise default to for routine application logs.

7. Model Selection and Evaluation Criteria

Model selection for enterprise conversational agents involves trade-offs that raw benchmark accuracy alone tends to underweight: response latency, cost per request, context-window size, native tool-calling support, and the availability of fine-tuning or lightweight customization. These criteria track the same three requirement tiers introduced in Section 3, summarized in Table 3.

Table 3: Model-Selection Criteria by Requirement Tier

Selection Criterion	Low-Sensitivity Tier	Moderate-Sensitivity Tier	Regulated, High-Sensitivity Tier
Model Capability & Cost	Smaller, lower-cost model	Mid-tier model with reliable instruction-following and multi-condition reasoning	Highest-capability model justified despite added cost
Latency Tolerance	Flexible	Aligned with customer-facing service-level expectations	Secondary to accuracy; multi-step verification such as self-consistency is accepted
Context Window	Short	Sized for multi-turn history plus retrieved passages	Large, for extensive audited source documents
Tool-Calling	Treated as a convenience rather than a requirement	Required for read actions, with structured schemas preferred	Strict schema validation and narrowly scoped authorization on every tool call
Fine-Tuning	Rarely justified	Evaluated case-by-case against the tier's rate of policy change	Justified only where it materially reduces risk, subject to an ongoing retraining obligation
Evaluation Approach	General-purpose benchmark scores as an acceptable baseline check	Use-case-specific factual-accuracy testing	Comprehensive multi-dimensional evaluation plus pre-deployment adversarial testing

Holistic evaluation work (Liang et al., 2023) has argued that model comparison should span accuracy, robustness, calibration, fairness, and efficiency together rather than resting on any one metric, a position with direct bearing on enterprise model selection, since a model that edges out a competitor on a general accuracy benchmark but shows greater variance under adversarial or edge-case input may be the wrong choice for a regulated deployment even while it looks better on a leaderboard. Wei et al. (2022b) add a further wrinkle: certain reasoning capabilities emerge only above a given model-scale threshold, so choosing a smaller, cheaper model for a use case that needs multi-step reasoning is not simply a modest accuracy-for-cost trade; it can mean the model cannot perform the required reasoning at all. Because hallucination remains a persistent failure mode even in otherwise capable models (Ji et al., 2023), evaluation criteria for enterprise conversational agents should include factual-accuracy testing specific to the agent's own grounding sources, rather than leaning solely on general-purpose benchmark scores.

Fine-tuning or lightweight customization, where a provider allows adjusting a base model against an organization's own examples, is a further criterion worth weighing separately from the base-model comparison above, since it changes the calculus for the moderate- and high-sensitivity tiers in particular. A customized model trained on an organization's own historical support transcripts or claim narratives can, in principle, reduce reliance on in-prompt retrieval by encoding domain-specific patterns more directly, but it also brings a maintenance obligation; the customized model has to be retrained or re-evaluated as underlying policies change, which a purely prompt-and-retrieval-based configuration does not carry in the same way. Enterprise teams weighing this trade-off should measure the expected accuracy gain from customization against the retraining cadence that a given requirement tier's rate of underlying policy change would actually demand, rather than adopting customization by default just because a provider makes it available. Figure 3, below, traces how these evaluation and selection decisions play out across a single request.



Figure 3. Request Lifecycle Through the Reference Architecture, with Monitoring Feedback Loop

Figure 3 traces one request through the six architectural layers from Section 4 and closes the loop back to monitoring, making the point that evaluation is not a one-time model-selection exercise but an ongoing process: production monitoring data, flagged low-confidence responses, escalation frequency, and tool-call failure rates should feed back into periodic re-evaluation of both the chosen model and the prompt-engineering configuration built around it.

A further consideration, increasingly common in practice, is that a single conversational-agent deployment does not have to commit to one model for every interaction. Multi-model routing, sending a low-complexity, low-sensitivity request to a smaller and cheaper model while escalating anything that fails a confidence check

or falls into a higher-sensitivity category to a larger, more capable model, applies the proportionality principle from Section 3 inside a single deployment rather than only across separate ones. This routing approach brings its own complexity, particularly in keeping prompt templates and evaluation criteria consistent across models with different context-window sizes and instruction-following behavior, but it lets one agent serve a wider range of requirement tiers without defaulting every request to the cost and latency profile of its most demanding case.

Cost modeling for enterprise conversational agents is worth doing explicitly rather than discovering after the fact, since commercially available LLMs are typically billed per token across both the prompt and the generated response. A requirement tier with a large retrieved-context budget and a self-consistency sampling policy that generates several reasoning paths per turn can carry a materially higher per-conversation cost than a low-sensitivity tier running a short prompt against a smaller model, often by an order of magnitude at production volume. Estimating this cost per requirement tier during the design phase in Section 3, rather than only after a pilot has already run against production traffic, lets an organization weigh the accuracy and governance benefits of a heavier configuration against what it actually costs to run before committing to it at scale.

Perceived responsiveness, which shapes user trust in a conversational agent as much as raw accuracy does, depends on a latency budget spanning several stages rather than model inference alone: network round-trip time to the model provider, retrieval-layer lookup time against the grounding source, the model's own generation time, which scales with both prompt length and requested output length, and any tool-call round-trips the reasoning layer initiates before producing a final response. A self-consistency configuration that samples several reasoning paths, or a tool-orchestration sequence that chains multiple external calls, can each add perceptible delay on top of a single model call, and enterprise teams should budget for this explicitly against a target response-time service-level objective for each requirement tier, rather than discovering the cumulative latency only after a heavier configuration is already in production.

Translating these criteria into an actual model-selection decision benefits from a structured evaluation process rather than an informal comparison based on marketing claims or a single leaderboard number. A practical bake-off runs a small set of candidate models against a held-out sample of the use case's own representative queries, curated to include both routine requests and the edge cases most likely to expose a weakness, scoring each candidate on the same rubric: factual accuracy against the agent's own grounding sources, adherence to the required output format, latency under realistic load, and, for the moderate- and high-sensitivity tiers, behavior under a small set of adversarial or boundary-testing prompts. Procurement processes that instead lean primarily on a vendor's published benchmark scores risk selecting a model that performs well on generic tasks but poorly on the specific reasoning pattern, structured-output requirement, or domain vocabulary a given enterprise use case actually depends on, since published benchmarks are, by construction, not tailored to any particular deployment's requirement profile. Running this bake-off before committing to a model and re-running an abbreviated version of it whenever a provider ships a new model version keeps the model-selection decision anchored to how a candidate model actually performs on the work it will be doing rather than to how it performs in the abstract.

8. Illustrative Application

To show how the requirement-driven framework plays out in practice, consider a generalized enterprise scenario, not tied to any specific client or organization, in which a large employer is replacing a static FAQ page with a conversational agent for employee benefits questions. Under the tiering scheme from Section 3, this sits close to the boundary between the low- and moderate-sensitivity tiers: it touches personal benefits data, moderate sensitivity, without extending into clinical or financial-advice territory, short of the high-sensitivity tier.

Applying the tier-to-technique mapping from Section 3 places this agent's design at moderate governance intensity: retrieval-augmented grounding against the organization's own benefits documentation (Lewis et al., 2020) rather than reliance on parametric model knowledge, since benefits rules change often and an ungrounded answer risks stating an outdated policy as though it were current; chain-of-thought prompting (Wei et al., 2022a) for multi-condition eligibility questions, since benefits eligibility routinely depends on several interacting conditions, tenure, plan tier, dependent status; a narrowly scoped tool-orchestration layer limited to read-only lookups against the benefits system rather than write access, consistent with the proportionality principle from Section 4; and a safety layer that escalates to a human benefits representative whenever the agent's confidence drops below a set threshold or the query touches a regulated sub-topic such as a qualifying life event.

Applying Section 7's model-selection criteria favors a mid-tier commercially available model with native tool-calling support and a context window large enough to hold several pages of retrieved benefits documentation

per turn, rather than either the cheapest available model or the highest-capability flagship model, reflecting the same cost-latency-accuracy trade-off discussed earlier, calibrated to this use case's moderate, not maximal, accuracy requirement.

This walkthrough is illustrative rather than an empirical evaluation: no client-specific deployment data is reported here, consistent with this article's conceptual, framework-level contribution. The specific model, prompting, and tool choices described above follow directly from applying the Section 3 tier-to-technique mapping and the Section 7 model-selection criteria to this use case's tier classification; a team facing a genuinely similar deployment should expect to arrive at a comparable configuration, while one whose data-sensitivity or regulatory profile differs even modestly from this example should expect that mapping to point it toward a different combination entirely.

A second, contrasting scenario shows the framework at the opposite end of the tiering scheme. Take a healthcare payer building a conversational agent to help members understand claim denials and appeal options, a use case that sits squarely in the regulated, high-sensitivity tier, since it touches protected health information and directly affects a member's access to care. Applying the same requirement-tier mapping here calls for a materially different configuration: self-consistency sampling (Wang et al., 2023) on eligibility and appeal-timeline questions rather than a single reasoning pass, given how much more an incorrect answer costs here; retrieval grounding restricted to a curated, regularly audited set of plan-document sources rather than a broader internal knowledge base, to keep provenance auditable; a tool-orchestration layer with no write access at all, limited to read-only claim-status lookups, with everything else routed to a human representative; and a governance posture that includes pre-deployment red-teaming (Perez et al., 2022) and continuous human-review sampling rather than the lighter monitoring that fit the benefits-inquiry example above. Placing these two scenarios side by side makes the proportionality principle concrete: the same underlying reference architecture supports both, but the specific technique, grounding, tool, and governance choices at each layer differ substantially by requirement tier.

Taken together, the two scenarios above point to a practical way to apply this framework to a new use case: classify the use case against the three tiers in Section 3 using the data-sensitivity and regulatory-exposure criteria described there, work through the Section 3 tier-to-technique mapping and the Section 7 model-selection criteria in order, treating each recommendation as a default rather than an automatic decision, and then check the resulting design against the six architectural layers in Section 4 to confirm no layer has been left implicit. This sequence does not replace organizational risk sign-off, particularly for the regulated tier, but it gives a team a concrete starting configuration to bring into that sign-off conversation rather than an open-ended design space.

The two scenarios also illustrate how differently the monitoring and observability layer from Section 4 should be instrumented once the requirement tier changes. The benefits-inquiry deployment can reasonably center its monitoring on aggregate signals, overall confidence score trends, the rate at which conversations escalate to a human representative, and periodic sampling of transcripts for quality review, since no single conversation in this tier carries an outsized consequence on its own. The healthcare-payer deployment needs monitoring built around a different unit of analysis entirely because a single incorrect appeal-timeline answer can directly affect a member's access to care, monitoring there has to support per-conversation traceability, tying each response back to the specific retrieved passages and reasoning trace that produced it, rather than relying on aggregate trends to surface a problem only after enough similar cases have accumulated to move the average. This difference in monitoring granularity, not just monitoring intensity, is itself a design decision the requirement tier should determine, and it is easy to underspecify if monitoring is treated as a single uniform capability applied identically across every conversational agent an organization operates regardless of tier. Neither scenario's monitoring design is more sophisticated than the other in any general sense; each is proportionate to what its own requirement tier actually demands.

Discussion

The framework in this article rests on a single organizing idea: that prompt-engineering technique, architectural complexity, tool authorization scope, and governance intensity should scale with a conversational agent's actual non-functional requirement tier, rather than being applied uniformly across an enterprise's whole agent portfolio. One implication of this is easy to miss in practice, over-engineering a low-sensitivity internal assistant with regulated-tier governance controls is not a conservative default. It is a resource-allocation error that slows delivery without a matching reduction in risk, since the controls appropriate to a healthcare or financial-services deployment (Bommasani et al., 2021) address risks that a low-stakes internal tool simply does not carry.

The proportionality principle also carries an implication for how an organization sequences its portfolio of conversational-agent initiatives over time, not just for how it designs any single agent in isolation. Starting with a low-sensitivity, internal-facing deployment lets a team build operational familiarity with the reference architecture in Section 4, its monitoring conventions, its incident-response patterns, before that team takes on a regulated, high-sensitivity deployment where the same mistakes carry materially higher consequences. Organizations that instead start their conversational-agent program directly with a customer-facing, regulated-tier use case, often because that is the most visible business opportunity, take on both the technical learning curve and the governance stakes at once, which this framework's tiering logic would generally advise against wherever a lower-stakes starting point is available.

Adopting a tiered framework like this carries organizational implications well beyond the technical decisions it directly addresses. Teams used to a single standard chatbot pattern may initially experience a tiering exercise as added process overhead, especially for the first few deployments classified into each tier before the classification itself becomes routine. That suggests the framework's practical value depends partly on organizational buy-in: a requirement-tiering exercise applied inconsistently, or overridden under delivery-timeline pressure, delivers little of the risk-proportional benefit the framework is meant to provide, no matter how carefully the underlying technical mapping in Section 3 is built.

The framework's proportionality principle also brushes up against responsible-AI concerns this article has not treated as a separate dimension, including fairness across user populations and accessibility for users who interact with the agent differently than the majority case the design was validated against. A requirement tier defined mainly around data sensitivity and regulatory exposure does not automatically capture these concerns, and a moderate-sensitivity assistant serving a demographically diverse customer base may need fairness-specific evaluation that the criteria in Section 7 do not explicitly call out. Extending the tiering scheme to fold in a fairness and accessibility dimension alongside sensitivity and regulatory exposure is a natural direction for refining the framework further, rather than a gap this article claims to have closed.

Although this article frames its contribution specifically around conversational agents, the underlying proportionality principle, and much of the six-layer reference architecture, generalizes with modest adaptation to other categories of LLM-based enterprise application that are not conversational in the narrow sense: document-summarization pipelines, automated content-classification systems, and code-generation assistants each face analogous questions about grounding, tool access, and governance intensity, even though they lack the multi-turn dialogue structure this article's context and memory layer specifically addresses. A fuller treatment of how the framework's tiering and mapping logic transfers to these adjacent application categories is left to future work, since properly confirming that transfer would mean examining the specific ways non-conversational applications differ in their input and output structure rather than simply asserting the analogy holds.

A further practical consideration, not addressed directly elsewhere in this article, concerns portability across model providers. A conversational agent whose prompt templates, tool schemas, and evaluation criteria are tuned closely to one provider's specific model behavior can be costly to migrate if that provider changes pricing, deprecates a model version, or gets replaced for organizational reasons unrelated to technical performance. Designing the reference architecture's layers, particularly the intent-understanding and prompt-engineering components, with an explicit eye toward which behaviors are provider-specific idiosyncrasies versus genuinely required capabilities can reduce this migration cost, though this article does not attempt to quantify how large that cost typically runs in practice, which would itself be a reasonable question for future empirical work (Gaggar, 2024).

The framework has real limits. The three-tier requirement structure in Section 3 simplifies for exposition, and actual deployments often show mixed sensitivity inside a single conversation, an internal assistant that occasionally surfaces personally identifiable information is a common enough case. The framework also assumes a commercially available, instruction-tuned model rather than a custom-trained one, which matches current enterprise practice (Ouyang et al., 2022) but may understate the design space open to organizations with the resources for custom model development. And the framework's governance mapping draws on institutional frameworks, the NIST AI Risk Management Framework (National Institute of Standards and Technology, 2023) and the OWASP LLM Top 10 (OWASP Foundation, 2025), that are still evolving; a mapping calibrated to their current form will need revisiting as those frameworks mature.

Evaluation is a further limitation worth naming directly. Holistic evaluation approaches (Liang et al., 2023) make a strong case that no single benchmark captures the dimensions of model behavior that matter, yet enterprise teams frequently lack the resources to run comprehensive multi-dimensional evaluation for every candidate model and configuration. This article recommends use-case-specific factual-accuracy testing as a minimum bar (Section 7), but a fuller account of lightweight, enterprise-feasible evaluation methodology

remains open for future work, as does empirical validation of the proportionality principle itself against outcome data from multiple real deployments, which this conceptual treatment does not attempt to provide. None of these limitations undermine the core proportionality argument; they instead mark the boundary between what this article establishes conceptually and what still needs empirical grounding (Bobba et al., 2021).

Conclusion and Future Work

We have presented a requirement-driven framework for designing enterprise LLM-based conversational agents, connecting functional and non-functional requirement tiers to specific choices in prompt-engineering technique, agent architecture, model selection, and governance control. Its central claim is one of proportionality: the prompting techniques, architectural components, and governance controls built into a given conversational agent should track that agent's actual requirement profile, rather than defaulting to either a minimal generic chatbot pattern or a maximal governance posture regardless of use case. The layered reference architecture, requirement-to-technique mapping, and model-selection criteria set out here are meant as a practical starting point for architects and technical leads moving conversational AI initiatives from pilot to production. Future work should pursue empirical validation of the proportionality principle against production deployment data, extend the requirement-tier structure to handle mixed-sensitivity conversations more directly, and integrate lightweight, enterprise-feasible evaluation methodology more closely into the model-selection criteria proposed in Section 7. More broadly, the shift this article argues for, away from a single default chatbot pattern and toward a deliberately tiered set of design responses, mirrors a maturation other areas of enterprise software have already gone through, where an initial period of uniform, one-size-fits-all tooling gives way to risk-differentiated practice once a technology's failure modes and their consequences become better understood across a wider range of production deployments. Enterprise LLM-based conversational agents are still early enough in this maturation that the specific tier boundaries and technique mappings proposed here should be expected to evolve; the proportionality principle underneath them is offered as the more durable contribution.

For an architect or technical lead evaluating a new conversational-agent initiative today, the practical starting point this article recommends is narrower than the full framework might suggest: classify the use case against the three requirement tiers in Section 3 before any architecture or model discussion begins, since every downstream recommendation in Section 3's tier-to-technique mapping and Section 7's model-selection criteria depends on that classification being right. Teams that skip this step and instead start from a preferred architecture or a preferred model tend to retrofit the requirement analysis afterward, if at all, which is precisely the pattern this article has argued produces avoidable governance gaps. Getting the tiering step right first, even informally, costs little relative to the cost of discovering a mismatch after a system is already handling production traffic. The framework, the architecture, and the criteria presented here are offered as a starting configuration for that conversation, not a final answer to it, and each is meant to be revisited over time as an organization's own experience with production conversational agents accumulates.

References

1. Ankur Bhatnagar, A., Bonagiri, k. (2026). Operational Excellence through Cloud-Native Distributed Systems. *International Journal of Computational and Experimental Science and Engineering*, 12(2). <https://doi.org/10.22399/ijcesen.5207>
2. Bobba, C. T., Choksi, R., Grewal, V. S., Basra, H. K., & Momin, S. (2021). Evaluation of Effects of Deproteinization and Intermediate Bonding on the Retention of Pit and Fissure Sealants-An In-Vivo study. *SVOA Dentistry*, 2(4), 139-142.
3. Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., et al. (2021). On the opportunities and risks of foundation models. Stanford Center for Research on Foundation Models. <https://crfm.stanford.edu/assets/report.pdf>
4. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901. <https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfc4967418bfb8ac142f64a-Abstract.html>
5. Deriu, J., Rodrigo, A., Otegi, A., Echegoyen, G., Rosset, S., Agirre, E., & Cieliebak, M. (2021). Survey on evaluation methods for dialogue systems. *Artificial Intelligence Review*, 54(1), 755–810. <https://link.springer.com/article/10.1007/s10462-020-09866-x>
6. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT 2019*, 4171–4186.

- <https://aclanthology.org/N19-1423/>
7. Gaggar, H., (2024). Two Planes of Truth: Execution Fidelity and Behavioral Correctness in Tool-Using LLM Agents. *International Journal of Computational and Experimental Science and Engineering*, 10(3).
<https://doi.org/10.22399/ijcesen.5498>
 8. Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., & Fritz, M. (2023). Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec 2023)*, 79–90.
<https://dl.acm.org/doi/abs/10.1145/3605764.3623985>
 9. Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., et al. (2023). Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12), Article 248. <https://dl.acm.org/doi/10.1145/3571730>
 10. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33, 9459–9474.
<https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html>
 11. Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., et al. (2023). Holistic evaluation of language models. *Transactions on Machine Learning Research*.
<https://openreview.net/forum?id=iO4LZibEqW>
 12. National Institute of Standards and Technology. (2023). Artificial Intelligence Risk Management Framework (AI RMF 1.0) (NIST AI 100-1). U.S. Department of Commerce.
<https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf>
 13. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35, 27730–27744.
https://proceedings.neurips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html
 14. OWASP Foundation. (2025). OWASP Top 10 for Large Language Model Applications (v2025).
<https://owasp.org/www-project-top-10-for-large-language-model-applications/>
 15. Perez, E., Huang, S., Song, F., Cai, T., Ring, R., Aslanides, J., Glaese, A., McAleese, N., & Irving, G. (2022). Red teaming language models with language models. *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP 2022)*, 3419–3448.
<https://aclanthology.org/2022.emnlp-main.225/>
 16. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.
https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html
 17. Shuster, K., Poff, S., Chen, M., Kiela, D., & Weston, J. (2021). Retrieval augmentation reduces hallucination in conversation. *Findings of the Association for Computational Linguistics: EMNLP 2021*, 3784–3803. <https://aclanthology.org/2021.findings-emnlp.320/>
 18. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 5998–6008.
<https://proceedings.neurips.cc/paper/7181-attention-is-all-you-need.pdf>
 19. Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., & Zhou, D. (2023). Self-consistency improves chain of thought reasoning in language models. *Proceedings of the 11th International Conference on Learning Representations (ICLR 2023)*.
<https://openreview.net/forum?id=1PL1NIMMrw>
 20. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2022a). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35, 24824–24837.
https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html
 21. Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., et al. (2022b). Emergent abilities of large language models. *Transactions on Machine Learning Research*.
<https://openreview.net/forum?id=yzkSU5zdWd>

22. Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., et al. (2024). The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2), 121101. <https://link.springer.com/article/10.1007/s11432-024-4222-0>
23. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. *Proceedings of the 11th International Conference on Learning Representations (ICLR 2023)*. https://openreview.net/forum?id=WE_vluYUL-X
24. Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., et al. (2023). Least-to-most prompting enables complex reasoning in large language models. *Proceedings of the 11th International Conference on Learning Representations (ICLR 2023)*. <https://arxiv.org/abs/2205.10625>