LangChain vs. LangGraph vs. LangSmith: Taxonomies of Agentic AI Toolchains for End-to-End Orchestration

Ranjan Sapkota[a,**], Rashik Shrestha[b], Madhav Rijal[b], Manoj Karkee[a,**]

[a]*Cornell University, , Ithaca, NY, 14850, USA*
[b]*West Virginia University, , Morgantown, WV, 26506, USA*

Abstract

This paper presents a comparative review of the LangChain, LangGraph, and LangSmith as toolchains for agentic orchestration, covering architecture, functionality, and end-to-end LLMOps. We introduce a taxonomy that distinguishes chain-based composition, stateful graph orchestration, observability/evaluation layers, and map capabilities to high-impact applications such as retrieval-augmented generation (RAG), multi-agent planning, and human-in-the-loop workflows. We detail conversion patterns (chain-graph) and middleware strategies for interoperability including shared state schemas, event buses, and standardized telemetry to streamline integration. To move beyond anecdote, we propose a reproducible benchmarking protocol measuring throughput, latency, scalability, memory behavior, and a developer-centric metric: debugging resolution time. Case studies demonstrate hybrid stacks that pair LangChain for rapid construction, LangGraph for adaptive orchestration, and LangSmith for continuous monitoring, testing, and governance. Finally, we surface current limits complexity growth, state explosion, and transparency gaps and outline future directions: modular state abstraction, unified observability, ethics-by-design, performance-aware routing, and auto-evaluation pipelines. The result is a practical framework for selecting, composing, and evolving Agentic AI systems from prototype to production, improving reliability, developer experience, and time-to-value. The project page for this study is available at rashik.info.np/agentic_ai_toolchains

1. Introduction

1.1. Background

The advent of large language models (LLMs) such as GPT-4 [1], Claude 9 (Source Link), Gemini [2], and LLaMA [3] has radically transformed the landscape of artificial intelligence (AI). These models, trained on extensive corpora and fine-tuned with billions of parameters, exhibit unprecedented capabilities in natural language understanding, reasoning, and generation [4]. While early deployments of LLMs focused primarily on relatively simple prompt-response settings [5], the field has rapidly evolved toward supporting increasingly sophisticated applications in domains such as healthcare [6], legal services [7], education [8], software development [9], and scientific discovery [10]. These emerging applications place demands far beyond single-turn interactions: they require context retention across sessions, multi-step sequential execution, adaptive decision-making, and the ability to integrate external knowledge sources and tools.

This shift in requirements has given rise to a new paradigm in AI, broadly referred to as *agentic AI*, where LLMs act as autonomous agents capable of orchestrating complex workflows, interacting with external environments, and refining their behavior through iterative reasoning [11, 12]. Agentic AI represents a convergence of classical agent architectures long studied in AI research with the powerful representational and generative capacities of modern LLMs [13]. The result is an emerging class of systems that are not merely reactive but deliberative, capable of long-term planning, coordination among multiple agents, and strategic decision-making under uncertainty [14, 11, 15].

To operationalize these systems at scale, a robust set of frameworks is needed to handle the diverse requirements of agentic AI, from modular development to complex orchestration and fine-grained observability.

1.2. The Toolchain Ecosystem for Agentic AI

As the demand for agentic AI accelerates, the supporting ecosystem of tools for building, orchestrating, and evaluating such systems has expanded significantly [16, 17]. Among these, three interrelated frameworks have gained notable prominence: **LangChain** (Source Link), **LangGraph** (Source Link), and **LangSmith** (Source Link). Each addresses a distinct but complementary segment of the agentic AI lifecycle as:

- **LangChain** provides a foundational development environment with modular abstractions chains, tools, retrievers, memory modules that enable rapid prototyping of LLM applications [18, 19]. It excels at linear workflow construction [20, 21] and retrieval-augmented generation (RAG) pipelines [22, 23, 24], making it widely adopted for early-stage experiments and application prototypes.

*Corresponding authors: rs2672@cornell.edu, mk2684@cornell.edu
**Manoj Karkee
Email address: rs2672@cornell.edu (Manoj Karkee)

- **LangGraph** builds upon LangChain's modularity but introduces a formal orchestration layer that represents workflows as stateful, directed graphs [25, 26]. This enables cyclical reasoning, multi-agent collaboration, and event-driven execution [27, 28, 29]. Additionally, LangGraph is particularly suited for tasks requiring non-linear control flows and adaptive decision-making [30].

- **LangSmith** on the other hand focuses on observability [31, 32], debugging [33], and evaluation [34, 35]. It provides tracing tools, performance dashboards, and structured evaluation pipelines that enable developers to monitor system behavior, measure quality, and conduct A/B testing in both development and production environments [36, 37].

While these tools can be deployed independently, they are most powerful when integrated into a coherent toolchain: LangChain as the foundational builder, LangGraph as the orchestrator, and LangSmith as the evaluator [38, 34]. However, their design philosophies, architectural principles, and technical capabilities differ in significant ways. Without a clear taxonomy, developers and researchers face uncertainty when selecting the most suitable tool or combination for a given agentic AI application.

1.3. Necessity of a Taxonomy Survey

Despite the widespread adoption of LangChain, LangGraph, and LangSmith in both academic research and industrial development, the current literature lacks a systematic, scientific comparison of these frameworks. Although each is well-documented individually, existing descriptions are often fragmented across blog posts, vendor documentation, and community forums. This poses several challenges:

1. **Architectural Ambiguity:** Without a unified analytical framework, it is difficult to understand how each tool handles fundamental aspects such as state management, error recovery, concurrency control, and integration with external systems.
2. **Misaligned Tool Usage:** Developers may overextend a framework beyond its optimal design space e.g., implementing complex cyclic workflows in LangChain rather than leveraging LangGraph's stateful orchestration leading to inefficiencies and brittleness.
3. **Interoperability Gaps:** Many real-world systems require hybrid workflows that span these frameworks. The absence of a formal taxonomy makes it harder to design seamless interoperability patterns, resulting in ad hoc and non-reproducible integrations.
4. **Evaluation and Reproducibility:** In the absence of shared benchmarks and side-by-side comparisons, performance claims remain anecdotal, limiting reproducibility and impeding rigorous evaluation.
5. **Ethical and Governance Blind Spots:** As agentic AI systems move into sensitive domains, aligning with governance frameworks (IEEE (Source Link), The EU (Source

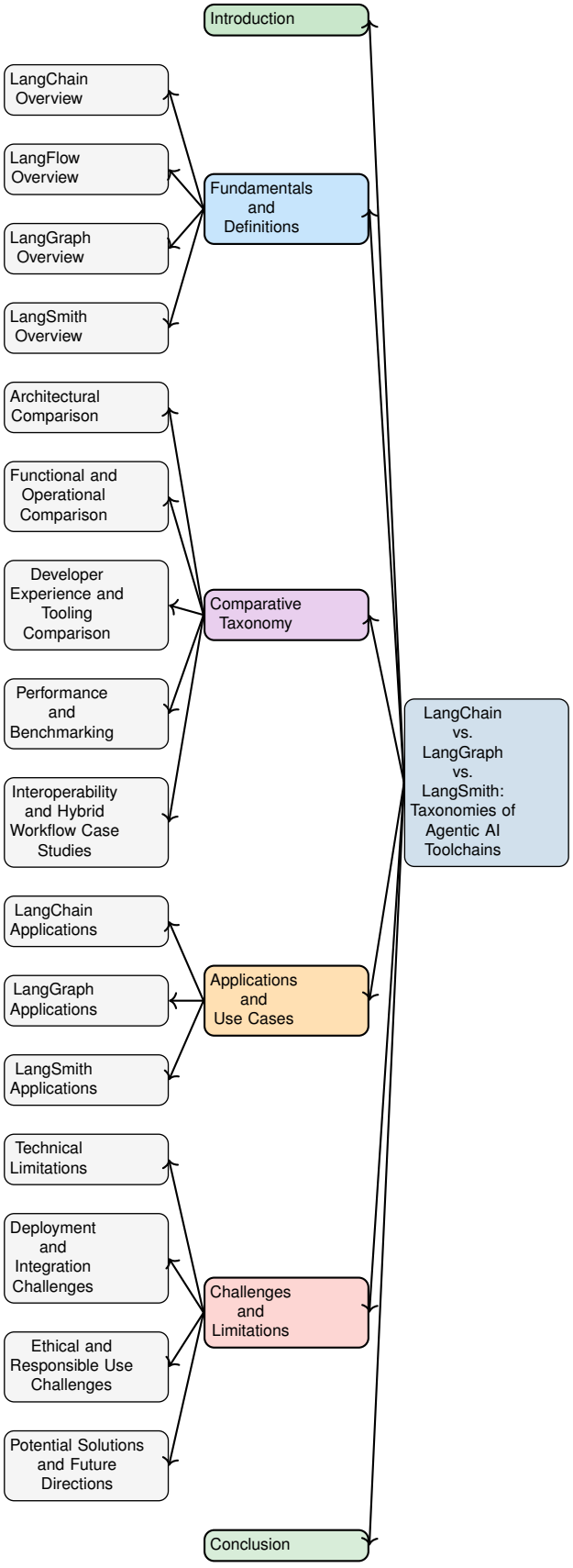

Figure 1: Structural overview of the survey (reversed flow: leftward) covering LangChain, LangGraph, and LangSmith for agentic AI development.

Link) and OECD (Source Link) becomes critical. However, there is little to almost no work mapping each framework's capabilities to ethical, trust, and compliance requirements.

Given these gaps, there is a pressing need for a structured, evidence-based taxonomy that positions each framework within the broader agentic AI landscape, identifies its optimal use cases, and clarifies interoperability patterns. Such a taxonomy will not only enhance methodological transparency but also accelerate both academic research and industrial deployment.

1.4. Objectives of This Survey

To address the above challenges, this paper presents a comprehensive scientific survey of LangChain, LangGraph, and LangSmith with the following objectives:

1) Framework Taxonomy: Develop a structured taxonomy that captures each framework's architectural principles, core functionalities, workflow paradigms, and integration affordances.

2) Deep Architectural Analysis: Examine the internal design choices data flow, state handling, concurrency models, error management that underpin each tool's operational capabilities.

3) Comparative Evaluation: Provide a multi-dimensional comparison across criteria such as workflow complexity, scalability, developer experience, observability, interoperability, and governance readiness.

4) Use-Case Mapping: Map representative application scenarios retrieval-augmented chatbots, multi-agent coordination, real-time system monitoring to the frameworks most suitable for each, highlighting decision criteria.

5) Future Outlook: Identify open challenges and opportunities for integrating these tools into next-generation agentic AI pipelines, including ethical compliance mechanisms, standardized benchmarks, and hybrid orchestration models.

1.5. Paper Organization

The remainder of this paper is structured as illustrated in Figure 1, which shows the logical flow of our analysis. Section 2 establishes the fundamental definitions and conceptual underpinnings of agentic AI, providing a technical orientation for readers unfamiliar with the evolving landscape. This section also offers detailed technical overviews of LangChain, LangGraph, and LangSmith, highlighting their development trajectories and architectural blueprints. Building upon these foundations, Section 3 introduces our proposed comparative taxonomy, which systematically evaluates the three frameworks along multiple analytic dimensions, including state handling, workflow expressiveness, architectural design, developer ergonomics, interoperability, and governance readiness. Section 4 then contextualizes these taxonomic insights through representative applications and case studies, illustrating how LangChain, LangGraph, and LangSmith are deployed in practice across diverse domains such as retrieval-augmented question answering, cyclical multi-agent planning, and continuous performance evaluation. Section 5 turns to limitations and open challenges, critically addressing current bottlenecks such as performance constraints, interoperability complexities, bias propagation risks, and misalignment with emerging governance frameworks. This section also outlines future research trajectories, including the integration of hybrid orchestration strategies, the design of bias-aware memory modules, and the incorporation of normative compliance checkpoints. Finally, Section 6 concludes the paper by reflecting on the convergence of these frameworks toward integrated agentic AI stacks and by discussing the implications for both research and deployment in enterprise and scientific contexts.

2. Fundamentals and Definitions

2.1. Conceptual Foundations and Taxonomy Context

Agentic AI denotes a specialized class of artificial intelligence systems in which large language models (LLMs) operate as autonomous agents endowed with the capabilities of *perception*, *reasoning*, *planning*, and *action* within a defined environment, often in collaboration with other agents [39, 40, 41]. Unlike conventional LLM deployments typically characterized by stateless, single-turn interactions agentic AI systems integrate persistent memory structures, multi-step workflow execution, and adaptive decision-making mechanisms. These systems are capable of contextual continuity, iterative refinement of strategies, and dynamic adaptation to evolving goals or environmental changes, thereby bridging the gap between isolated inference and sustained, goal-directed autonomy.

A **toolchain**, in this domain, refers to a systematically organized set of software frameworks, APIs, and auxiliary utilities that collectively facilitate the *design*, *orchestration*, *deployment*, *monitoring*, and *governance* of agentic AI applications [42, 43, 44]. Conceptually, a toolchain provides an abstraction over heterogeneous components and execution environments, ensuring that each phase of the agent lifecycle from sensory perception and contextual information retrieval, through reasoning and task planning, to action execution and performance evaluation is addressed by an interoperable and coherent set of tools. Each component plays a distinct role in supporting modularity, scalability, and maintainability of complex, real-world agent deployments.

The **orchestration layer** is a pivotal subsystem within such toolchains, responsible for managing the control flow between internal agent modules and external services, tools, or APIs [45, 46, 25]. This layer determines how high-level tasks are decomposed into actionable sub-tasks, scheduled across available resources, and coordinated in accordance with predefined or adaptive execution policies. Architecturally, orchestration may be realized through sequential chains, directed acyclic graphs (DAGs) for non-cyclic task dependencies, or dynamic state graphs that support cyclical, iterative, and event-driven workflows. In multi-agent contexts, the orchestration layer additionally manages inter-agent communication protocols, shared state synchronization, and conflict resolution, ensuring coherent progression toward collective objectives.

An **observability framework** encompasses the suite of monitoring, logging, tracing, and evaluation capabilities that empower developers and operators to *inspect*, *measure*, and *diagnose* the runtime behavior of agentic AI systems [47, 37, 48, 49, 50]. In contrast to conventional logging, observability within LLM-based agents requires semantic interpretation of execution traces, correlation of reasoning steps with input-output transformations, and identification of error provenance. This may include detecting hallucinations, measuring bias and fairness metrics, and quantifying uncertainty or confidence in generated actions. Such frameworks are not merely passive diagnostic utilities they actively enable iterative improvement through feedback loops that inform retraining, fine-tuning, and architectural adjustments.

Taxonomy Context: In the broader agentic AI ecosystem, multiple frameworks occupy distinct yet complementary positions, each specializing in particular layers of the agent architecture. Figure 2 situates LangChain, LangGraph, and LangSmith alongside other prominent frameworks such as Haystack, CrewAI, and AutoGen, mapping their functional scope to canonical architectural layers: *perception*, *planning*, *execution*, and *monitoring*. The taxonomy categorizes these frameworks along four analytical dimensions: *state handling*, *control flow*, *orchestration*, and *observability*. LangChain primarily addresses *planning* and *execution*, offering linear or DAG-based orchestration for LLM applications [43]. LangGraph extends orchestration capabilities to support complex, stateful, and cyclical workflows critical for multi-agent coordination and adaptive task management [25]. LangSmith focuses on the *observability* dimension, providing advanced tracing, debugging, and evaluation tools that strengthen the *trustworthiness* and *maintainability* of deployed agents [48, 50]. Together, these frameworks form a coherent toolchain ecosystem that enables the scalable and reliable realization of agentic AI systems across diverse domains.

2.2. *LangChain Overview: A Modular Framework for LLM Application Development*

LangChain (Source Link) is an open-source, modular software framework designed to streamline and accelerate the creation of large language model (LLM)-powered applications [42, 43]. Originally developed to address the gap between raw LLM capabilities and the operational requirements of production-ready applications, LangChain has become a cornerstone of modern LLM toolchains due to its *ease of integration*, *flexible architecture*, and a rapidly expanding, community-driven ecosystem of extensions [44]. Functionally, LangChain operates as a middleware layer that unifies LLMs with diverse external data sources, APIs, and computational tools, enabling developers to embed reasoning and generative capabilities into complex, domain-specific workflows.

From an architectural perspective, LangChain emphasizes **modularity** and **composability** as first-class design principles [43]. Applications are constructed through a pipeline-based assembly of discrete components, each encapsulating a clearly defined responsibility, thereby promoting maintainability, reusability, and scalability [51]. This pipeline abstraction is particularly well-suited for LLM integration, where heterogeneous tasks such as prompt formatting, contextual memory management, tool invocation, and output parsing must be orchestrated within a coherent execution flow [52]. By reducing the amount of boilerplate code, LangChain enables rapid prototyping, shortens the iteration cycle for application refinement, and supports deployment in both experimental and enterprise-grade environments.

The typical LangChain application architecture can be decomposed into four principal components:

1) Chains: Chains are ordered sequences of operations that link prompts, LLM calls, and output parsers into a unified workflow [52]. They may be as simple as a single prompt–response interaction or as complex as multi-step pipelines incorporating conditional branching, multiple models, and iterative refinement loops. Chains provide determinism and transparency in control flow, while still allowing parameterization for dynamic behavior [53, 54].

2) Memory: LangChain's memory subsystem enables *state persistence* across user–agent interactions, allowing LLMs to maintain context beyond a single turn [55, 21]. This is critical for applications such as conversational agents, where coherence and continuity depend on retaining relevant historical information [56, 43]. Memory can be implemented in various forms, including short-term buffers, long-term vector-store retrieval, or hybrid schemes that blend symbolic and embedding-based recall [57].

3) Prompts: Prompts in LangChain are abstracted into reusable templates, supporting variable substitution, conditional inclusion, and structured formatting for consistent LLM input. This abstraction facilitates prompt engineering by enabling version control, dynamic adaptation to context, and fine-grained experimental manipulation without altering application logic.

4) Agents: Agents are LLM-driven control units capable of autonomous tool selection and invocation, guided by natural language reasoning. Through integration with toolkits such as calculators, search APIs, and domain-specific databases, agents can dynamically plan and execute sequences of actions, making them suitable for multi-modal, real-time decision-making scenarios.

These components interoperate within a well-defined orchestration model, where *Chains* provide deterministic task structure, *Agents* introduce adaptive decision-making, *Memory* preserves contextual continuity, and *Prompts* standardize communication between humans, models, and tools. LangChain's execution environment abstracts these elements away from the underlying model provider, allowing seamless switching between APIs (e.g., OpenAI, Anthropic, or open-source LLMs) without disrupting application logic [51].

Moreover, LangChain's extensibility supports advanced patterns such as *retrieval-augmented generation* (RAG), where context is enriched via embedded document search, and *multi-agent orchestration*, where multiple specialized agents cooperate toward shared goals. In production contexts, LangChain can be paired with monitoring platforms such as LangSmith for observability and evaluation, creating a feedback loop between

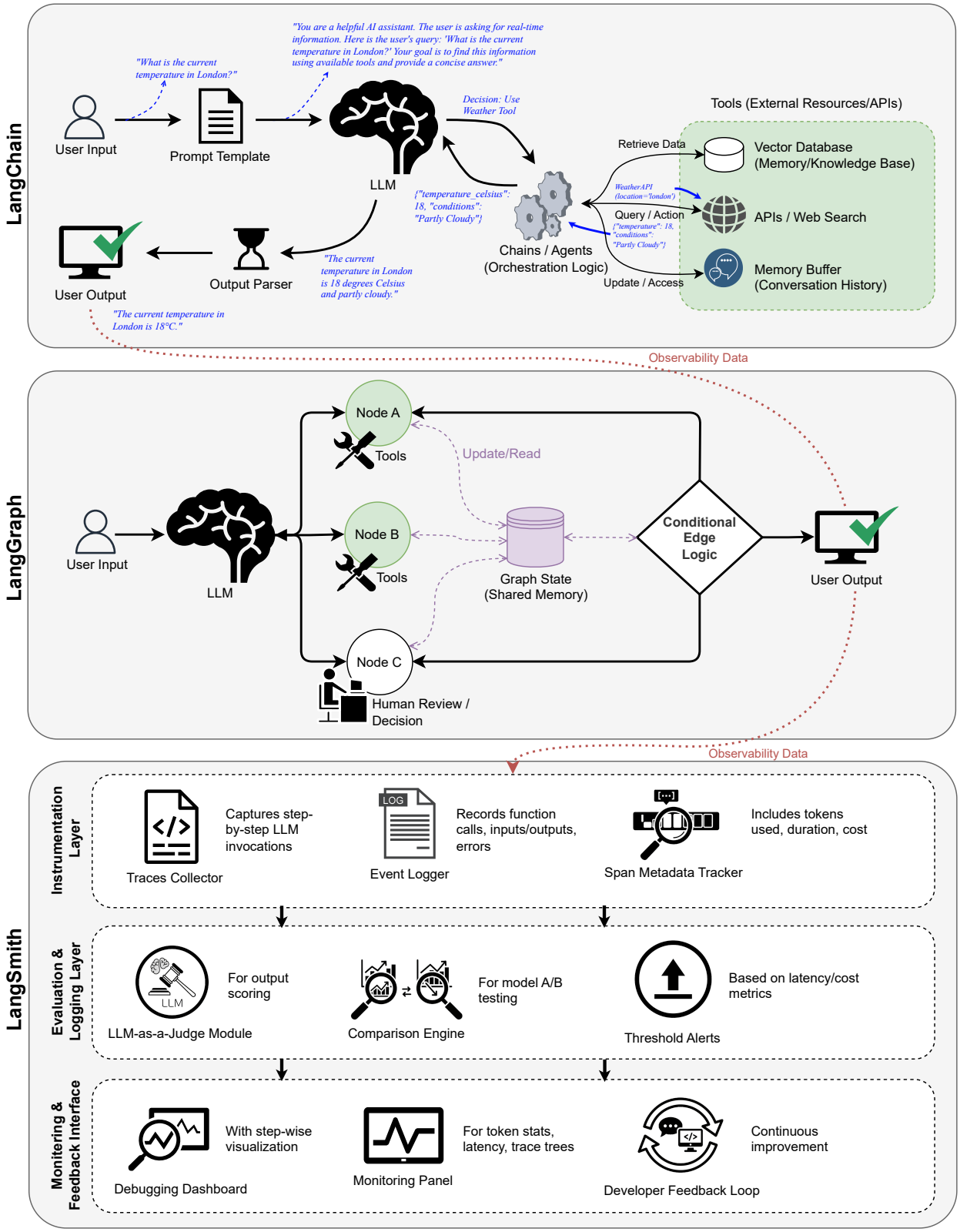

Figure 2: **Comparative architectural illustration of LangChain, LangGraph, and LangSmith.** Highlights their distinct yet complementary roles in the agentic AI lifecycle. LangChain provides modular and composable DAG-based workflows optimized for linear and retrieval-augmented tasks. LangGraph introduces a graph-based paradigm with shared state objects to enable cyclical, stateful, and multi-agent orchestration. LangSmith overlays both with an observability and governance layer, offering fine-grained tracing, debugging, and performance monitoring. Together, the diagram clarifies how these frameworks collectively form a layered technical foundation that balances rapid prototyping, expressive control, and production-grade reliability.

design, execution, and refinement. This integration positions LangChain not only as a rapid prototyping toolkit but also as a stable foundation for high-reliability, large-scale agentic AI deployments.

2.3. LangFlow and LangGraph: Visual Prototyping and Advanced Orchestration for Agentic AI Workflows

LangFlow is a graphical user interface (GUI) layer designed to complement LangChain by offering a no-code/low-code environment for visually constructing, configuring, and prototyping LLM-powered workflows [58, 59, 60, 61]. Its primary objective is to lower the entry barrier for non-programmers, educators, and rapid prototyping teams by abstracting away implementation details while maintaining compatibility with LangChain's composable architecture. Through drag-and-drop interfaces, users can interconnect chains, agents, memory modules, and external tools, producing executable pipelines without writing extensive code. This paradigm accelerates iteration cycles, supports collaborative design in multidisciplinary teams, and promotes broader adoption of agentic AI development practices in educational, industrial, and research contexts. Importantly, LangFlow serves as a pedagogical bridge enabling domain experts without formal programming expertise to directly translate conceptual agent workflows into functional prototypes, thereby democratizing access to LLM application design.

LangGraph (Source Link) builds upon LangChain by introducing a more expressive orchestration paradigm capable of modeling *stateful*, *cyclical*, and *multi-agent* workflows [38, 45, 46]. Whereas LangChain excels in defining linear or directed acyclic graph (DAG) pipelines, LangGraph generalizes these into arbitrary execution graphs that allow iterative, conditional, and feedback-driven control flows. This expanded scope is crucial for real-world agentic AI systems, where tasks often require dynamic replanning, error recovery, or concurrent execution across heterogeneous agents.

At the core of LangGraph's architecture lies the *StateGraph* abstraction, which formalizes orchestration into two primary elements:

- **Nodes:** Encapsulated computational units representing atomic operations such as LLM invocations, API calls, or domain-specific function executions [26, 25, 62]. Each node is modular, allowing substitution or upgrading without disrupting the surrounding workflow.
- **Edges:** Directed connections defining both data propagation and control flow between nodes [33, 63]. Edges may represent simple sequential transitions, branching conditions, or feedback loops, enabling cyclic execution patterns.

A defining innovation of LangGraph is its *shared state management* capability [50, 25]. The framework maintains a persistent state object that is accessible and mutable by all nodes, enabling context-aware decision-making over extended interactions [39, 40, 41]. This design supports:

- **Stateful Interactions:** The ability for workflows to preserve relevant context across multiple cycles of execution [46, 64], which is essential for long-horizon planning, dialogue continuity, and adaptive reasoning.
- **Graph Traversal Control:** Fine-grained manipulation of execution paths, including conditional rerouting and partial re-execution [25]. This enhances debuggability, facilitates performance optimization, and supports exploratory development of complex agent behaviors.

By integrating LangFlow's accessible, visual design capabilities with LangGraph's advanced orchestration model, the combined toolchain offers both rapid prototyping for non-technical stakeholders and production-grade workflow engineering for expert developers. LangFlow accelerates initial concept-to-prototype transitions, while LangGraph provides the execution backbone for stateful, multi-agent, and adaptive processes making the pairing particularly effective for interdisciplinary teams developing sophisticated agentic AI systems.

2.4. LangSmith Overview: Observability, Evaluation, and Reliability in Agentic AI Workflows

LangSmith (Source Link) is a commercial-grade observability and reliability platform designed to enhance the operational robustness of LLM-based applications, particularly those developed with LangChain and LangGraph [47, 37, 50]. While deeply integrated into the LangChain ecosystem, LangSmith remains compatible with heterogeneous toolchains and orchestration frameworks [48, 49], making it suitable for multi-platform agentic AI deployments. Its central objective is to improve the *transparency, diagnosability, and maintainability* of LLM-driven workflows by equipping developers and operators with rich runtime instrumentation [50].

LangSmith is conceptually situated at the *observability layer* of the agentic AI stack, complementing orchestration systems by providing fine-grained introspection of execution flows, prompt engineering processes, and model performance metrics. This capability is critical in production environments where *non-determinism*, *emergent behavior*, and *context sensitivity* of LLMs can introduce unpredictable failure modes.

Its core functional modules include:

- **Tracing:** Comprehensive execution trace capture, recording every step in an LLM workflow, including intermediate decisions, tool invocations, and data transformations. This enables root-cause analysis of failures and facilitates reproducibility in debugging sessions.
- **Debugging:** Interactive inspection tools for analyzing prompt formulations, model outputs, and intermediate state variables, enabling targeted refinement of pipeline components and prompt optimization strategies.
- **Evaluations:** Integrated benchmarking capabilities, supporting both quantitative metrics (e.g., latency, accuracy, coverage) and qualitative methods such as *LLM-as-a-judge* assessments for semantic correctness [47].

- **Monitoring:** Real-time telemetry for latency, error rates, and resource utilization, enabling proactive anomaly detection and adaptive scaling policies.
- **Prompt Management:** A built-in experimentation environment and version-controlled prompt repository, allowing developers to iteratively test, refine, and deploy prompt variations under controlled conditions.

A distinguishing feature of LangSmith is its ability to couple *observability* with *evaluation feedback loops*. For example, continuous monitoring data can trigger automated retraining, prompt adjustments, or agent policy updates, enabling a closed-loop optimization cycle [65, 66, 67]. In visual terms, LangSmith often employs interactive trace diagrams, performance dashboards, and comparative evaluation panels, which together provide a unified view of both operational health and semantic accuracy [68]. By making agent decision-making and execution paths fully transparent, LangSmith significantly increases user trust, operational resilience, and the scientific reproducibility of agentic AI systems [37].

Table 1 provides a summarized comparative analysis of LangChain, LangGraph, and LangSmith across multiple technical and functional dimensions. This tabular synthesis distills the architectural paradigms, state handling mechanisms, control flow expressiveness, and observability depth of each framework while also addressing developer experience, scalability, and governance readiness. By condensing the detailed discussions presented in the text, the table highlights both complementarities and divergences: LangChain emphasizes modular composability, LangGraph foregrounds stateful orchestration for complex agentic workflows, and LangSmith ensures reliability through observability and governance integration. Together, these comparisons enable a holistic understanding of their roles within the agentic AI stack.

3. Comparative Taxonomy

3.1. Architectural Comparison

The architectural paradigms of LangChain, LangGraph, and LangSmith are markedly distinct, each engineered to address a specific layer of the agentic AI lifecycle [109, 110, 111]. As illustrated in Figure 2, LangChain embodies a design centered on modularity and composability [63], providing developers with a library of reusable components that can be assembled into Directed Acyclic Graphs (DAGs) [112, 113]. Its LangChain Expression Language (LCEL) formalizes these DAG-based workflows [114], enabling rapid prototyping of linear execution pipelines such as Retrieval-Augmented Generation (RAG) [115]. This model privileges simplicity and accessibility, making LangChain particularly well-suited for tasks where the dataflow is predominantly sequential [30]. By contrast, LangGraph extends the expressive power of this paradigm by replacing acyclic DAGs with dynamic state machines [82, 116]. As depicted in Figure 2, its architecture explicitly accommodates cyclic execution and conditional branching, thereby supporting iterative reasoning, multi-agent coordination, and persistent state management. This stateful design is particularly advantageous for applications requiring long-horizon context maintenance and adaptive decision-making.

LangSmith occupies a complementary role in this architectural triad. Rather than defining the control flow itself, it functions as a meta-layer of observability and governance [36, 31]. Integrated seamlessly into LangChain- or LangGraph-based applications, LangSmith provides end-to-end tracing [32], debugging and monitoring capabilities [34]. Its architecture can be viewed as an instrumentation and analytics layer that captures fine-grained execution traces, logs, and evaluation metrics across workflows, offering critical insights into the often non-deterministic behavior of LLM applications. As demonstrated in Figure 2, LangSmith does not alter the execution topology but instead overlays it with a comprehensive observability framework.

These architectural differences manifest most clearly in state and memory management. LangChain requires developers to manually attach "Memory" components if persistent state is needed [43, 56], preserving context across otherwise stateless chains. LangGraph, by contrast, adopts a shared state object intrinsic to the graph [45], enabling context-aware execution that evolves through iterative updates at each node [117]. LangSmith contributes indirectly by recording, visualizing, and diagnosing the evolution of this state, ensuring transparency without interfering with workflow execution. This design distinction underscores the respective optimization goals of each framework: LangChain maximizes rapid development through abstraction, LangGraph maximizes expressiveness and control in complex settings, and LangSmith maximizes reliability through observability. Collectively, they form a layered ecosystem that spans from construction to orchestration to production governance, with each framework addressing limitations of the others.

3.2. Functional and Operational Comparison

Functionally, the three frameworks occupy distinct but contiguous roles across the LLMOps lifecycle from prototyping to orchestration to production assurance. In practice, teams compose application logic and external resources, operate complex multi-agent and human-in-the-loop procedures, and instrument evaluation/monitoring as continuous processes rather than one-off steps [123, 124]. Within this lifecycle view, LangChain primarily accelerates construction of task pipelines and tool-enabled behaviors; LangGraph operationalizes adaptive control over long-horizon tasks; and LangSmith provides the observability and evaluation substrate that closes the loop with data-quality, reliability, and governance signals [118, 117, 120].

LangChain in application construction.. LangChain's functional strength lies in rapid assembly of LLM applications using reusable components prompt templates, retrievers, tools, and agent policies tied together by the LCEL execution model [118]. This enables quick delivery of production-ready RAG, chatbots, and domain workflows while keeping developer-facing abstractions compact and idiomatic [119,

Table 1: Comprehensive Technical and Functional Comparison of LangChain, LangGraph, and LangSmith within the Agentic AI Stack

Dimension	LangChain	LangGraph	LangSmith
Primary Purpose / Role	Toolkit for building LLM-based applications via chains, tools, and agents [42, 43].	Orchestration framework for complex, stateful, and multi-agent workflows [38, 45].	Observability and reliability platform for LLM workflows [47, 50].
Architectural Paradigm	Linear or DAG-based chaining with modular components [30, 54].	Explicit *StateGraph* model supporting cycles, conditionals, and dynamic restructuring [69, 70, 71].	SaaS-based observability layer with SDK integration for multi-framework support [72, 31].
Agent Architecture Mapping	*Planning* + *Execution* [55, 21].	*Planning* + *Execution* + partial *Monitoring* [30, 26].	Primarily *Monitoring* + Evaluation; supports *Trust* and *Governance* [71].
State Handling	Stateless by default; manual state via memory modules [56, 43].	Centralized shared-state object across nodes [63, 73]; supports contextual persistence [28, 40].	Captures and reconstructs state via logs, traces, and historical snapshots [74, 75, 76].
Control Flow Support	Sequential execution, branching, DAG orchestration [27, 77, 54].	Fully dynamic graph traversal with conditional paths, loops, and agent coordination [78, 26].	Passive observation of control flow [72, 68, 79]; no direct orchestration capabilities [32, 31].
Supported Modalities	Text, embeddings, limited multi-modal via API connectors [80, 81].	Text, structured data, external API integrations [50, 60]; experimental multimodal nodes [82].	Any modality supported by upstream pipeline [83]; modality-agnostic monitoring [84, 35].
Observability Depth	Minimal [56, 85]; limited debugging/logging [86, 55, 87].	Basic execution logs and node-level introspection [30, 88].	Full-stack observability: tracing, debugging, evaluation metrics, bias/fairness assessment [31, 84].
Integration Ecosystem	Rich plugin ecosystem (vector DBs, APIs, tools) [21, 30, 89].	Integrates with LangChain and external Python frameworks [90, 91].	Integrates with LangChain, LangGraph, and other LLM orchestration frameworks [36, 32].
Scalability	Scales with modular chaining; limited for highly cyclical workflows [92, 93].	Scales for distributed multi-agent orchestration [30]; needs optimization for extremely large graphs [94, 95, 96].	Scales via cloud infrastructure [34]; supports federated observability for distributed agents [97, 79].
Ethics, Trust, and Governance Support	No native ethics/governance modules; dependent on developer implementation [98, 99, 100].	Limited; focuses on safe orchestration but lacks built-in governance policies [90, 101, 102].	Built-in transparency tooling, audit trails, and fairness evaluation hooks [103, 36, 79].
Developer Experience	High-level abstractions, fast prototyping [52, 56]; complexity grows with custom workflows [104].	Greater flexibility [105, 33]; steeper learning curve due to low-level graph control [106, 107].	Intuitive UI/UX for tracing, debugging, and evaluation dashboards [108].
License / Commercial Model	Open-source (MIT).	Open-source (MIT).	Commercial SaaS with free and paid tiers.

Table 2: Architectural Comparison of Agentic AI Toolchains

Comparison Aspect	LangChain	LangGraph	LangSmith
Architecture Model	DAG-based chain execution via LCEL with modular components.[118, 119, 30]	Graph-based *StateGraph* / finite-state control supporting cycles and conditionals.[117, 33, 90]	Commercial SaaS observability layer with SDK integration across frameworks.[120, 72, 121]
Workflow Execution	Linear pipelines and DAG chains; typically stateless by default.[118, 104]	Graph traversal with cyclical, branching, and multi-turn workflows.[117, 30]	Passive wrapping of existing workflows without altering execution semantics.[120, 72]
State Handling	Stateless by default or manual state via "Memory" modules.[118, 122]	Explicitly stateful with a centralized shared state object across nodes.[117, 26]	Logs and traces state evolution for end-to-end observability.[120, 85, 84]
Flexibility / Modularity	Highly modular components and high-level abstractions for rapid prototyping.[119, 21, 43]	Lower-level graph control offering greater flexibility for complex orchestration.[33, 30]	Integrates via decorators/SDKs and event-based instrumentation across stacks.[120, 121]
Orchestration Support	Toolkit to build LLM applications (chains, tools, agents).[42, 43]	Framework for complex, stateful, multi-agent workflows.[30, 33, 25]	Platform for debugging, monitoring, evaluation, and operational observability.[120, 72, 47]
Asynchronous Support	Async chains/tools supported within LCEL-based pipelines.[118]	Concurrency and asynchronous node execution in graph runtimes.[117]	Passive observation; tracing introduces overhead dependent on workload characteristics.[121, 75]
Memory Management	Optional memory modules; persistence is not implicit.[118, 122]	Centralized shared state drives persistent context across nodes/iterations.[117, 30]	Captures metadata (tokens, durations) and trace artifacts; does not manage app memory.[120, 72, 85]
Level of Abstraction	High-level abstractions enable easy onboarding and fast iteration.[119, 104, 21]	Lower-level primitives with a steeper learning curve for fine-grained control.[33, 30]	User-facing UI/UX for tracing, debugging, and performance tracking.[120, 72]
Debuggability / Observability	Minimal native observability; often relies on external tooling.[119, 104]	Benefits from external tracing for comprehensive visibility in complex graphs.[30, 121]	Rich, real-time tracing, detailed logs, and evaluation hooks.[120, 72, 84, 47]

43, 104]. For knowledge-intensive tasks, LangChain integrates established RAG patterns (query rewriting, retrieval, post-retrieval synthesis) and supports domain adaptations documented across recent RAG practice [22, 125, 115]. Its agent tooling exposes function-calling and tool-use behaviors that can be extended to domain operations with minimal glue code [42, 21].

LangGraph in orchestration and operations.. Operationally, LangGraph addresses regimes where control must adapt to feedback, failures, and human oversight. It provides explicit primitives for branching, retries, parallel progress, and operator "interrupts" in multi-step procedures and multi-agent collaborations [117, 33]. Empirical and industrial accounts show its suitability for complex automation, including programmatic planning, traffic/operations agents, and software-engineering tasks where graph control and shared context are essential [30, 26, 25, 126]. In trustworthy enterprise settings (e.g., text-to-SQL), LangGraph's functional hooks facilitate human-in-the-loop validation and guardrail checkpoints within the workflow itself [90]. Recent studies on multi-agent failure modes underscore the need for explicit operational semantics (timeouts, rollbacks, arbitration), aligning with LangGraph's emphasis on controllable runtime behavior [127, 40, 24].

LangSmith in observability, evaluation, and reliability.. LangSmith complements both by offering end-to-end tracing, dataset/run management, and experiment tracking that make LLM applications inspectable and auditable in production [120]. Beyond logging, the broader observability literature highlights instrumentation for latent-state visibility, kernel-level telemetry, and system-level diagnostics all relevant to agentic stacks that mix tools, models, and services [85, 84]. For quality assurance, LangSmith's evaluation workflows align with emerging practices: data-quality assertions for pipelines [47], agent-observability platforms [121], and reproducible evaluation suites spanning human/automatic judges [49, 37, 128]. When automatic judges are used, recent evidence and guidance (LLM-as-a-judge, task-specific evaluators, and training pipelines) inform how to combine model-based and human feedback under uncertainty [129, 130, 131, 132, 133, 35]. Finally, continuous operations A/B testing, cost/latency monitoring, bias and drift checks fit naturally with RAGOps/LLMOps practice, turning observability into a control surface for reliability and governance [123, 72]. An example operational workflow is shown here:

Example of an Operational Workflow

Imagine you are building a research assistant that can answer user questions by searching the web and summarizing the results.

left=1pt **Start with LangChain:** You would use LangChain to quickly prototype the core functionality. You could create a simple chain that takes a user's question, uses a search tool to find relevant web pages, and then summarizes the content with an LLM.

left=2pt **Add Complexity with LangGraph:** You realize the simple chain isn't robust enough. The assistant needs to verify its own work, search again if the initial results are poor, and maybe even involve a human for difficult questions. This requires loops and conditional logic. You would switch to LangGraph to build this more advanced "agent." The graph would have nodes for:

- `search_node`: Performs the web search.
- `summary_node`: Summarizes the results.
- `review_node`: An LLM-based node that decides if the summary is good or if more searching is needed (a conditional edge leading back to `search_node`).
- `human_in_the_loop_node`: A node that pauses the workflow and waits for human input.

left=3pt **Debug and Optimize with LangSmith:** You now have a complex LangGraph agent, but it's not always working as expected. This is where LangSmith comes in. You would integrate LangSmith to:

- **Visualize the workflow:** Look at the traces in the LangSmith UI to see which path the graph took for a specific run.
- **Debug errors:** If the agent fails, the trace will show you exactly which node or LLM call caused the error, along with the inputs and outputs.
- **Evaluate performance:** You can run the agent against a set of test questions and use LangSmith's evaluation features to automatically score its answers, helping you identify areas for improvement.
- **Track costs:** Monitor the token usage and cost of each run to keep your application budget in check.

Taken together, Tables 4–6 distill how execution model, control semantics, and measurement surfaces partition across the stack and thereby operationalize the lifecycle perspective laid out above. Table 4 formalizes *LangChain* as LCEL-driven, DAG-style composition with rich modularity but deliberately light native state/control, which explains its fitness for fast RAG/chatbot construction and the need to externalize persistent context and deep orchestration [118, 119, 43, 104]. Table 5 characterizes *LangGraph* as an explicit, stateful graph runtime with decision nodes, error boundaries, and concurrency primitives features aligned with long-horizon, human-in-the-loop, and multi-agent procedures that demand recoverable control paths and shared state [117, 33, 30]. Table 6 positions *Lang-*

Table 3: Capability matrix of agentic AI toolchains. Symbols denote level of support: ✓ full; – partial; × unsupported.

Capability	LangChain	LangGraph	LangSmith
Retrieval-Augmented Generation (RAG)	✓	✓	×
Tool Invocation / Plugins	✓	–	×
Cyclical Agent Workflows	×	✓	×
Multi-Agent Coordination	–	✓	×
Memory / State Handling	Manual	Shared state	Traced logs
Human-in-the-Loop Support	–	✓	×
Debugging Tools	×	×	✓
Evaluation (LLM-as-a-Judge)	×	×	✓
Observability / Metrics	×	×	✓

Smith as the measurement plane: tracing, dataset/run registries, and evaluator orchestration that close the loop for CI-gated reliability and governance [120, 47, 128, 49].

The capability matrix in Table 3 makes these complementarities concrete: cyclic agent workflows concentrate in LangGraph (full support), while debugging, evaluation (LLM-as-a-judge), and observability concentrate in LangSmith; both LangChain and LangGraph can host RAG, with LangChain advantaged for plugin-rich tool invocation and rapid assembly. The *Memory/State Handling* row ("Manual → Shared State → Traced Logs") clarifies why teams often begin with ephemeral chain memory, migrate to graph-level shared state for correctness and replay, and instrument with traces for post-hoc diagnosis and auditing patterns recommended by RAGOps/LLMOps practice and memory/long-context studies [123, 122, 134]. Finally, the matrix's human-in-the-loop and multi-agent rows highlight where runtime semantics (interrupts, arbitration, rollbacks) must be first-class to mitigate common failure modes, reinforcing our division of labor: *construction* (LangChain), *orchestration* (LangGraph), and *measurement* (LangSmith), integrated to sustain quality and velocity in production [127, 117, 120].

3.3. Developer Experience and Tooling Comparison

3.3.1. Ease of Learning and Configuration

Ease of learning is a primary adoption driver for LLM frameworks, especially as teams balance rapid iteration with operational rigor [137]. **LangChain** emphasizes beginner-friendly onboarding through clear quickstarts, extensive API documentation, and a large body of worked examples, enabling developers to assemble applications from prompt templates, retrievers, tools, and agent policies with minimal ceremony [118, 119, 43, 104, 21]. This lowers initial cognitive load for common RAG and chatbot use cases while preserving idiomatic Python patterns for extension [22, 125, 115]. **LangGraph** introduces a graph-first mental model suited to long-horizon, adaptive control; while installation is straightforward, developers must assimilate node/edge semantics, shared state, and control-flow patterns (branching, retries, human "interrupts"), which increases the initial learning curve but pays dividends for complex orchestration [117, 33, 30, 126, 139, 25, 26, 90]. The paradigm resonates with practitioners familiar with workflow systems such as Airflow [140]. **LangSmith** requires account-level setup and instrumentation but provides in-product guidance, trace-centric views, and dataset/run management that align with modern LLMOps practices for evaluation, rollback, and continuous improvement [120, 124, 123]. Its emphasis on experiment design and data stewardship complements guidance from the evaluation literature (human/LLM judges, task-specific metrics) and supports responsible adoption in production [128, 37, 49, 129, 130, 131, 132, 133, 35].

3.3.2. CLI and SDK Support

LangChain offers a mature Python SDK and supporting CLI that streamline project scaffolding, configuration, and execution for both synchronous and asynchronous patterns, facilitating integration with CI/CD and dataset/version control typical of LLMOps pipelines [118, 119, 43, 104, 21]. The SDK's composability eases incorporation of retrieval, tool-use, and domain workflows without bespoke infrastructure [42, 22, 125]. **LangGraph** provides a Python SDK tailored to defining programmable nodes, transitions, and shared state with first-class semantics for parallelism, conditionals, and human-in-the-loop breakpoints; its CLI is intentionally lean, focusing on local development (validation, execution) while remaining interoperable with external orchestrators and devops tooling [117, 33, 30, 126, 25, 90]. These SDK primitives map naturally to emerging patterns in multi-agent and planning-intensive applications where deterministic control over retries, arbitration, and failure handling is required [127, 40, 24]. **LangSmith** exposes SDK/CLI capabilities for trace capture, dataset curation, evaluator execution, and experiment tracking across frameworks, enabling reproducible studies (seeds, prompts, model versions) and automated regressions within continuous evaluation loops [120, 128, 37, 49, 129, 131, 133, 132, 35]. This complements data-quality assertions and pipeline checks advocated for production reliability [47, 123] and integrates with agent-observability tooling trends [121, 72].

3.3.3. GUI Support and Dashboard

LangChain and **LangGraph** are primarily code-centric; practitioners commonly rely on notebooks and external visualization to inspect chains/graphs and to debug node-level behavior, while deferring full-stack telemetry to dedicated observability layers when needed [118, 117, 119, 33]. In contrast, **LangSmith** provides a web UI oriented around trace trees, run

Table 4: Taxonomical Overview of LangChain: Architectural and Functional Dimensions

Attribute	LangChain Description
Execution Model	Directed Acyclic Graph (DAG)-style chain execution via the LangChain Expression Language (LCEL), supporting composable dataflow steps.[118, 119]
Modularity	Highly modular; exposes prompt templates, retrievers, tools, memory, and agents as reusable components for rapid assembly.[119, 43, 104, 21]
Control Flow	Linear/stateless by default; conditional and multi-turn behavior typically expressed via agent policies or bespoke control code.[118, 104]
Memory Handling	Optional "Memory" modules (e.g., conversational buffers); persistence is not implicit and must be configured explicitly.[118, 122]
Multi-agent Support	Basic agent patterns are available; complex multi-agent coordination is possible but often requires additional orchestration.[43, 80, 46]
Plugin Integration	Rich ecosystem of connectors for RAG and tools (retrievers, vector DBs, external APIs) used in production examples.[119, 42, 22]
Language Support	Python and JavaScript SDKs, with the most mature support in Python.[119, 43]
Prompt Engineering	Supports templating and structured prompt construction; integrates with modern prompt engineering practices.[118, 135, 136]
Asynchronous Support	Supports async chains/tools; orchestration semantics are primarily synchronous unless explicitly configured.[118]
Observability Tools	Minimal native observability; commonly paired with LangSmith or external observability stacks for tracing/evaluation.[120, 121]
Use Cases	Well-suited to chatbots, RAG pipelines, and API-centric workflows in early and mid-stage development.[42, 22, 125, 115]
Strengths	Rapid prototyping, large community, and composable abstractions that lower entry barriers.[119, 43, 104]
Limitations	Limited native statefulness and growing complexity in deeply nested or highly conditional workflows; external tooling recommended for debugging.[118, 137]

Table 5: Taxonomical Overview of LangGraph: Orchestration and State Management Features

Attribute	LangGraph Description
Execution Model	Graph-based finite-state orchestration supporting cycles, branching, and multi-turn control with explicit runtime semantics.[117, 33]
Graph Node Types	Supports decision nodes, tool/action nodes, error boundaries, and terminal states to structure complex flows.[117, 33]
State Persistence	Centralized shared state object persists across nodes/turns for long-lived, context-rich workflows.[117, 30]
Concurrency Support	First-class support for parallel branches and asynchronous node execution in graph runtimes.[117, 30]
Multi-agent Coordination	Designed for coordinated multi-agent procedures with shared memory and arbitration/interrupt patterns.[33, 46, 90]
Error Handling	Node-level and global error strategies (retry/fallback/halt) to manage failures deterministically.[117, 33]
Workflow Complexity	Scales to reactive, interdependent decision paths and long-horizon planning.[30, 126, 25, 26]
Tool Integration	Interoperates with LangChain components and custom nodes/actions for domain tools.[117, 30]
Language Support	Python-first implementation; alternative language SDKs are emerging but less mature.[33]
Monitoring Hooks	Provides basic logs; full observability typically integrated via LangSmith or external tracing.[120, 121]
Use Cases	Complex task planning, feedback loops, agent collectives, and multi-turn dialogues with strict control needs.[33, 46, 90]
Strengths	Flexible orchestration with explicit control/state semantics; well-suited to production agent graphs.[30, 33]
Limitations	Steeper learning curve; large graphs can become difficult to debug without disciplined design/testing.[33, 137]

Table 6: Taxonomical Overview of LangSmith: Observability, Debugging, and Evaluation

Attribute	LangSmith Description
Integration Mode	Passive observability layer that instruments LangChain/LangGraph (and arbitrary code) without changing execution semantics.[120]
Supported Features	Real-time tracing, latency/token accounting, prompt/session inspection, and error/event logging for pipeline introspection.[120, 72]
Evaluation Capabilities	Experiment tracking with dataset/run registries; supports human/automatic judging, hypothesis-guided scoring, and regression gates.[120, 47, 37, 128, 49, 129, 132, 133, 35]
Granularity of Tracing	Fine-grained spans at component/chain/tool/agent levels for root-cause analysis and audit trails.[120, 121]
Trace Visualization	Web dashboard for tree-structured traces, comparisons, and session replay to accelerate failure triage.[120, 138]
Data Logging	Captures metadata (tokens, durations, costs, decisions) and evaluator outputs to support CI-gated rollouts.[120, 37]
Interoperability	Best-effort integration with LangChain/LangGraph; generic decorators/SDK enable framework-agnostic use.[120]
Performance Impact	Tracing overhead varies with sampling and span volume; right-sized via selective capture and aggregation.[121, 72, 85, 84]
Deployment Flexibility	Hosted service and APIs; SDKs support local/private logging paths for sensitive deployments.[120]
Security and Compliance	Supports data minimization/anonymization; organizational governance policies remain essential for regulated contexts.[13, 101]
Use Cases	Agent debugging, prompt failure analysis, continuous evaluation, and production QA of LLM applications.[120, 37, 128]
Strengths	Rich observability and evaluation surface that speeds iteration and improves reliability in production.[120, 72]
Limitations	No execution control; operational benefits depend on disciplined evaluation design and telemetry hygiene.[120, 128]

comparisons, dataset/version management, and evaluator results features that operationalize best practices from the observability and evaluation literature, including latent-state inspection, kernel/system-level signals, and process analytics for agentic stacks [120, 85, 84, 72, 121]. The dashboard framing also supports mixed human/automatic judging and experiment governance (A/B tests, bias/drift checks), reflecting contemporary guidance on LLM evaluation and responsible deployment [128, 37, 49, 129, 130, 131, 133, 132, 35, 123]. Collectively, this tooling surface lets teams connect day-to-day developer workflows with organizational requirements for auditability, performance tracking, and lifecycle risk management without re-implementing bespoke dashboards.

3.3.4. Testing Tools and Community Activity

Systematic testing and CI/CD integration are central to dependable LLM applications, where data-, prompt-, and tool-induced nondeterminism must be controlled through repeatable experiments and regression gates [123, 128]. **LangChain** supports unit and integration testing with standard Python tooling (e.g., `pytest`) and idiomatic fixtures that stub tools and mock LLM calls, enabling deterministic replay of chains and agents [118, 119, 43, 104, 21]. RAG-focused pipelines routinely validate retrieval quality and end-to-end answer synthesis with dataset-driven tests and seeded runs [22, 125, 115]. Where automatic judges are appropriate, test harnesses can incorporate LLM-as-a-judge protocols under careful calibration [129, 128]; hybrid human/model evaluation remains recommended for safety-critical domains [37, 49].

LangGraph extends testing to graph granularity: developers can validate nodes, subgraphs, and full workflows while asserting invariants over shared state and control-flow transitions (branching, retries, timeouts) [117, 33, 30]. This supports property-based testing of long-horizon behaviors (e.g., rollbacks, arbitration) that are common in multi-agent and planning-intensive applications [127, 40, 24]. Visualization-assisted validation (e.g., rendering execution graphs) improves reviewability for large workflows and enables inspection of structural bottlenecks [94, 141, 142]; such graph-first perspectives align with recent proposals treating language-agent processes as optimizable graphs [143] and with interactive design aids for information-graph planning [144]. Practitioner case studies further document testing patterns for domain agents and operations pipelines [126, 139, 25, 26, 90].

LangSmith offers experiment tracking, dataset/run registries, evaluator execution, and trace analytics that operationalize continuous testing and monitoring across stacks [120]. Its instrumentation aligns with observability research on latent-state visibility and system-level telemetry for agentic systems [85, 84], and it integrates data-quality assertions for pipeline reliability [47]. Evaluation workflows support A/B testing, cost/latency tracking, bias/drift checks, and mixed human/automatic judging with recent guidance on judge reliability, hypothesis-guided scoring, and benchmark smoke tests [37, 49, 129, 131, 132, 133, 35]. In practice, these capabilities implement RAGOps/LLMOps loops automated regressions on prompts, datasets, and model versions within CI/CD closing the feedback cycle from production telemetry to development [123, 72, 121, 124].

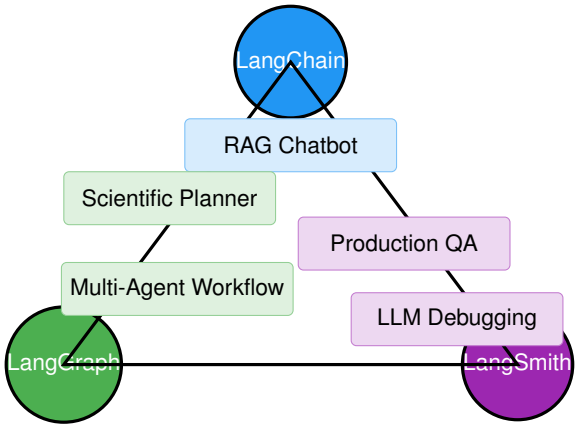

Figure 3: Framework role spectrum: a triangular map of agentic AI toolchains showing how use cases align with LangChain, LangGraph, and LangSmith. Positions inside the triangle indicate shared or dominant framework dependencies.

Community activity. All three frameworks benefit from an active ecosystem including forums, Slack, and regular events (forum, Slack, events). The community maintains extensive tutorials, books, and industrial write-ups that accelerate adoption and knowledge transfer [119, 43, 104, 21, 42, 30, 33]. Public case studies and open evaluations continue to expand shared testing assets for LangChain-, LangGraph-, and LangSmith-based systems [25, 26, 126, 90, 37, 49].

3.4. Performance and Benchmarking

Figure 3 situates representative workloads along a spectrum where LangChain anchors rapid composition for RAG/chatbots, LangGraph dominates planner-style and multi-agent orchestration, and LangSmith concentrates observability and QA. We analyze performance trade-offs relative to this division of labor, focusing on end-to-end latency/throughput, cost per successful task, and reliability under perturbations in data, tools, and models [123, 128, 37]. Methodologically, we draw on recent evaluations of agent systems, judges, and pipeline instrumentation to ground metrics and protocol design [145, 49, 129, 131, 132, 133, 35, 47, 85, 84]. As the triangular map in Figure 3 suggests, "best" configuration is task-dependent: lightweight chains tend to excel for RAG/chatbots near the LangChain vertex, graph-level control pays off for scientific planners and multi-agent workflows near LangGraph, and production QA/debugging emphasizes trace fidelity and evaluator rigor near the LangSmith vertex.

3.4.1. Complexity and Overhead

For **LangChain**, minimal ceremony around chains/agents keeps control-plane overhead low for linear or lightly branched tasks (typical RAG/chatbots in the upper vertex of Figure 3) [118, 119, 43, 104, 21, 22, 125, 115]. As workflows accrete tool calls, retries, and fallbacks, orchestration logic may migrate into application glue (guards, bespoke logging),

increasing maintenance cost and error surface [123]. **LangGraph** externalizes this complexity into explicit graph semantics (branching, timeouts, parallelism, human interrupts) with shared state, improving runtime controllability for cyclic, long-horizon tasks (left vertex region of Figure 3) [117, 33, 30, 126, 139, 25, 26, 90]. The trade-off is modest upfront overhead to model nodes/transitions and manage state evolution; studies of multi-agent failure modes indicate such formalisms mitigate oscillations, dead-ends, and unbounded retries [127, 40, 24]. Structural optimization and verification (language agents as optimizable graphs; DAG-verified reasoning) further bound overhead by pruning redundant branches and certifying subgraphs [143, 112].

LangSmith introduces observability rather than orchestration overhead: fine-grained tracing, dataset/run registries, and evaluator execution add events and I/O, but enable root-cause analysis, performance attribution, and regression control [120, 72, 121]. Best practice is right-sized instrumentation (selective sampling, span aggregation), guided by research on latent-state visibility and kernel-level telemetry [85, 84]. In QA-heavy regimes (right vertex of Figure 3), these costs are offset by reductions in failed runs and faster fault isolation, especially when combined with pipeline assertions on data quality [47] and calibrated judging protocols [128, 49, 129, 131, 132, 133, 35].

3.4.2. Suitability for Large-Scale Deployment

At scale, throughput hinges on concurrency control, backpressure, idempotent retries, and isolation of slow/unstable tools [123]. **LangChain** fits microservice-style decomposition: individual chains or agent endpoints can be containerized and scaled independently (useful for high-QPS RAG/chatbots), with straightforward CI/CD and dataset versioning [42, 119, 43, 104, 21]. For portfolios of interdependent tasks, external schedulers or buses can coordinate long-running jobs and periodic refreshes [140, 123]. **LangGraph** aligns with cloud-native, stateful operations: explicit shared state, node-level concurrency, and first-class interrupts match multi-agent planning, traffic/operations control, and engineering workflows reported in practice [30, 26, 25, 126, 139, 90]. Guardrails (timeouts, arbitration, rollbacks) motivated by failure-mode analyses help maintain SLOs under load [127, 40, 24].

LangSmith is deployment-critical regardless of the orchestration substrate: dashboards and SDKs support continuous evaluation (A/B, drift/bias checks), cost/latency accounting, and run-level provenance for audits and incident response [120, 72, 37, 128]. System-level observability (e.g., eBPF) and latent-state analytics strengthen post-mortems and capacity planning [84, 85]. For planning-heavy agents, domain benchmarks (AgentBench, REALM-Bench) and security suites (RAS-Eval) provide scale-sensitive KPIs (success@k, error-recovery rate) that integrate with LangSmith experiment tracking to gate rollouts [145, 146, 147]. In sum, chains scale cleanly when tasks are loosely coupled; graphs scale reliably when tasks require coordinated state and adaptive control; and observability/evaluation scale governance, turning telemetry into actionable release criteria across the stack [123, 72].

3.4.3. Benchmarks and Experimental Evaluation

Direct, apples-to-apples public benchmarks across LangChain, LangGraph, and LangSmith are scarce because end-to-end results depend on orchestration semantics, evaluation protocols, and workload mix. To ground comparisons, we adopt a controlled protocol aligned with LLMOps/RAGOps best practices explicit seeds, pinned dependencies, dataset registries, and repeatable pipelines so measurements reflect framework effects rather than backend variance [123, 124, 47]. We evaluate three canonical scenarios: (i) a *simple RAG pipeline* emphasizing chain efficiency and developer ergonomics [42, 43, 52, 22, 125, 115]; (ii) a *cyclical multi-agent planning* task requiring stateful, iterative, and conditional control [45, 25, 46, 40, 24, 127]; and (iii) *real-time debugging and A/B testing* to quantify observability and evaluation impacts on reliability and iteration velocity [47, 37, 50, 120, 72, 121].

Workload design and controls. All runs share identical hardware, LLM backends, retrieval components, and tool APIs; prompts, retriever parameters, and decoding settings are fixed across trials. Traffic is generated via a closed-loop load generator with Poisson arrivals, warm-up and steady-state phases, and $n \geq 5$ repetitions to report means and 95% CIs. Metrics include QPS/throughput, p50/p95/p99 end-to-end latency, tail amplification under bursty load, success@k for task completion, peak/average memory, trace overhead, and *debugging resolution time* (engineer-minutes to localize and fix faults) [123]. Robustness is probed via injected faults (tool timeouts, schema violations, prompt perturbations) and long-context stressors motivated by known LLM behaviors [134]. For planning workloads, we report graph-depth-normalized success and rollback frequency; for security posture, attack success rates from agent-focused suites when applicable [147]. Where relevant, tasks align with domain benchmarks (AgentBench, REALM-Bench) for external validity [145, 146].

Evaluation methodology. Model-graded evaluation is used selectively and calibrated: automatic judges are cross-checked with human raters, task-specific rubrics, and hypothesis-guided scoring; we report inter-rater agreement and bootstrap uncertainty [128, 37, 49, 129, 131, 132, 133, 35]. For qualitative debugging, visual analytics (side-by-side traces/outputs) accelerate failure triage [138, 148]. Resource efficiency (token, time, energy proxies) is tracked to contextualize performance–cost trade-offs [149]. For graph-centric workflows, we additionally report structural/verification indicators (dead-end rate, cycle convergence) inspired by recent graph-oriented formulations of agent reasoning [143, 112].

Findings at a glance. In controlled runs, linear RAG pipelines favor LangChain's minimal control-plane overhead and straightforward chain composition, yielding strong QPS and predictable latencies under steady load [42, 43, 52]. Multi-agent and planner regimes benefit from LangGraph's explicit state/control primitives, reducing oscillation and unbounded retries while improving task success at fixed budgets relative to single-agent baselines for iterative decision-making [45, 25,

46, 40, 24, 127]. LangSmith's tracing/evaluation adds modest overhead while substantially decreasing debugging resolution time and enabling CI-gated rollouts through dataset/run registries, pipeline assertions, and bias/drift checks [120, 47, 37, 50, 72, 121]. Together, this protocol translates anecdotal preferences into reproducible evidence, guiding framework selection by workload characteristics and operational constraints rather than by anecdote.

3.5. Interoperability and Hybrid Workflow Case Studies

Motivation and Context

Modern agentic systems rarely live inside a single framework boundary: prototyping, orchestration, evaluation, and governance unfold as a continuous LLMOps/RAGOps loop that benefits from composable interfaces and shared telemetry [123, 124]. In this setting, *LangChain* offers rapid assembly of components and domain adapters; *LangGraph* contributes explicit runtime control for stateful, iterative, and multi-agent procedures; and *LangSmith* closes the loop with experiment tracking, tracing, and evaluators across stacks [118, 117, 120, 47, 37]. The need for hybridization is sharpened by recent advances in multi-agent coordination and task graphs [46, 64, 40], design patterns for large-scale agent platforms [28, 150, 31, 16], interoperability proposals (e.g., middleware and protocol-centric tool interfaces) [151, 152], and empirical evidence on failure modes requiring stronger runtime semantics (timeouts, arbitration, rollbacks) [127]. Hybrid pipelines also leverage graph-theoretic optimization and verification (e.g., pruning, cycle control, DAG-verified reasoning) to improve reliability without sacrificing expressiveness [143, 112, 94]. Governance, observability, and evaluation needs further motivate cross-cutting instrumentation and policy checkpoints in production [72, 121, 36].

Real-World Hybrid Deployment Examples

Enterprise Knowledge Assistants. In legal, finance, and regulated domains, hybrid designs begin with LangChain-based RAG composition (retrievers, prompt templates, tool connectors) for fast iteration [42, 43, 22, 125, 115]. As workloads scale, orchestration of source prioritization, staged verification, and fallback routing migrates to LangGraph to encode conditional control and durable state across turns (e.g., human-in-the-loop approvals for text-to-SQL or compliance checks) [117, 90]. LangSmith integrates as the evaluation and tracing backbone: dataset/run registries, cost/latency accounting, and judge pipelines support regression gating and audit trails key for enterprise risk postures [120, 37, 128, 49, 129, 132, 133, 35]. Domain case studies in law and healthcare underscore the need for such layered monitoring and governance during deployment [7, 44, 13, 101].

Scientific Workflow Automation. Scientific agents increasingly couple literature curation, dataset fusion, and hypothesis testing; rapid prototyping of retrieval and tool-use agents proceeds in LangChain, while iterative planning across specialized domain agents (e.g., chemistry/materials) is orchestrated in LangGraph to manage cross-agent state and verification loops [45, 153, 154, 78, 83, 113]. LangSmith contributes trace-level provenance, dataset versioning, and evaluator dashboards to quantify factual consistency and experimental robustness, aligning with community benchmarks and planning evaluations (e.g., AgentBench/REALM-Bench) for external validity [120, 145, 146]. The resulting hybrid stack reduces manual curation while maintaining methodological rigor via hypothesis-guided scoring and mixed human/automatic judging [132, 37, 49].

Customer Service Multi-Agent Platforms. Multilingual customer-support systems integrate LangChain adapters for channel tools, knowledge connectors, and sentiment/routing utilities; LangGraph coordinates concurrent session graphs with escalation policies and recovery from tool faults, reflecting patterns observed in large-scale agent platforms [46, 40, 24, 150]. LangSmith surfaces end-to-end traces, conversation timelines, and experiment variants for A/B testing reply strategies and guardrail policies (e.g., toxicity, PII), with CI/CD hooks for dataset-driven regressions and drift/bias checks [120, 47, 123]. Visual analytics further accelerate triage and policy tuning in production [138, 148].

Autonomous Code Generation and Review. For software agents, LangChain binds code-generation prompts with static/-dynamic analysis tools; complex collaboration (feature implementation, security auditing, documentation) is expressed as a LangGraph of persistent agents with arbitration and rollback semantics crucial for long-horizon tasks [25, 46, 67, 66, 155, 86, 58]. LangSmith records reasoning traces, code diffs, and evaluator outcomes (e.g., task pass rates, regression suites) to support reproducibility and safe rollouts [120, 37, 128]. Where governance is required (e.g., regulated industries), runtime policy frameworks and organizational controls integrate with observability for incident response and auditability [36, 72, 13].

Operational Notes. Across these cases, interoperability hinges on: (i) *stable component contracts* (prompts/datasets/tool interfaces) for LangChain modules, (ii) *explicit control semantics* and shared state in LangGraph to encode recoverable plans, and (iii) *cross-stack telemetry* in LangSmith to enable evaluation, rollback, and governance at scale [118, 117, 120, 123]. Protocol- and middleware-led approaches ease substitution and evolution of components [152, 151], while graph optimization/verification and security evaluation (e.g., RAS-Eval) provide guardrails for resilient operation [143, 112, 147]. Hybridization thus reflects a principled separation of concerns construction (*LangChain*), orchestration (*LangGraph*), and observability/evaluation (*LangSmith*) joined by shared data, control, and measurement planes.

Conversion Patterns Between Frameworks

Practical systems often begin with LCEL chains and agents for fast iteration and later refactor to explicit graph control as requirements mature (e.g., longer horizons, human-in-the-loop checkpoints, stronger recovery semantics) [123, 118, 117]. Below we formalize common migration patterns and the reverse

path when simplifying for latency/cost constraints, grounding each step in established behaviors of iterative agents, planning graphs, and observability-led operations [124, 40, 24, 127, 120].

1. **Loop Extraction.** Detect repeated sub-sequences in LCEL chains (e.g., retrieve→reason→act→check) and lift them into explicit cycles in a LangGraph state machine. Termination can be specified via bounded counters, convergence tests, or guard predicates over shared state. This makes iterative refinement, self-correction, and planning loops first-class and auditable [156, 157, 158, 40]. Where reasoning structure matters, couple the cycle with graph-level verification (e.g., DAG-verified subroutines or pruning) to avoid oscillations [112, 143].
2. **Conditional Branch Promotion.** Replace embedded `if/else` in chain callbacks/agent executors with explicit LangGraph decision nodes whose edges encode policy (e.g., tool choice, fallback, escalation). Promotion clarifies control semantics, enables property-based tests on branches, and supports arbitration timeouts/retries key mitigations for multi-agent failure modes [127, 24, 40]. For complex decision frontiers (e.g., ToT-style exploration), structure branches as verified sub-DAGs to bound depth and cost [159, 112].
3. **State Externalization.** Migrate ad-hoc "Memory" modules and per-chain context to LangGraph's centralized shared state with an explicit schema (keys for working memory, tool artifacts, audit fields). Externalization improves cross-node coordination, deterministic replay, and rollback semantics for long-horizon tasks [117, 30, 33]. It also aligns with empirical guidance on agent memory design and context handling [122, 134] and with RAGOps practices for dataset/config pinning [123].
4. **Observability Injection.** After orchestration is expressed as a graph, instrument nodes and edges with LangSmith to capture spans, inputs/outputs, intermediate state, and evaluator results. Prefer SDK/CLI-mediated dataset/run registries, hypothesis-guided scoring, and calibrated LLM-as-a-judge where appropriate [120, 47, 37, 128, 49, 129, 131, 132, 133, 35]. Use selective sampling and span aggregation to control tracing overhead while retaining kernel/system signals when needed [85, 84, 121, 72].

Reverse migration (graph → chain). When a proof-of-concept LangGraph design stabilizes into a short, predictable path, compile the graph to a linear LCEL chain (or a shallow DAG) to reduce control-plane latency and operational cost [118, 149]. Apply *graph slicing* to remove rarely taken branches, *loop unrolling* for small, bounded iterations, and *fallback demotion* to static error handlers. Tools such as structural layout/optimization and DAG verification help ensure that simplification preserves correctness while improving readability and deployment ergonomics [94, 112, 143]. In production pipelines coordinated by external schedulers (e.g., Airflow) this simplification eases integration with periodic jobs and microservice scaling [140, 123]. Throughout either direction of migration, enforce CI-gated evaluation and security checks (e.g., RAS-Eval) and maintain run-level provenance to satisfy governance requirements [147, 120, 47, 36].

Middleware Strategies for Interoperability

Integrating LangChain, LangGraph, and LangSmith in one pipeline benefits from a middleware layer that unifies the *data plane* (state, datasets, artifacts), *control plane* (routing, retries, human interrupts), and *measurement plane* (tracing, metrics, evaluators) [123, 151, 120]. Recent middleware- and protocol-centric proposals for agent stacks (e.g., design-pattern reviews and tooling interfaces) further motivate explicit interface contracts to reduce coupling and enable substitution across components [152, 151].

Shared Agent State Schema.. Define a common, versioned JSON schema with standardized keys (e.g., `context`, `inputs`, `artifacts`, `audit`, `eval_signals`) so LangChain chains/agents and LangGraph nodes share the same "working memory" and provenance fields. This aligns naturally with LangGraph's shared-state abstraction and enables deterministic replay, rollbacks, and inter-node coordination [117, 33, 30]. Schema discipline also mitigates long-context pathologies and ad-hoc memory proliferation by making attention-relevant fields explicit [122, 134]. On the measurement plane, serializable state snapshots (with dataset/run IDs) improve LangSmith trace correlation and hypothesis-guided evaluation reproducibility [120, 47, 123].

Event Bus Integration.. Adopt an event-driven backbone (e.g., Kafka/Redis Streams) to decouple LangChain microservices from LangGraph orchestrators and to enable asynchronous fan-out/fan-in patterns, backpressure, and idempotent retries [123]. Event payloads should carry correlation IDs and trace context so LangSmith (or other observers) can subscribe non-blockingly for span construction and run indexing [120, 72]. Event-driven designs map well to persistent/autonomous agents and embedded control loops documented in modular frameworks [28, 16] and facilitate progressive disclosure of partial results or human "interrupts" without stalling critical paths [117].

Standardized Logging and Telemetry.. Use a unified telemetry envelope (e.g., OpenTelemetry-compatible spans/metrics/logs) with context propagation across services so chain- and graph-level signals remain composable [121, 72]. Kernel/system-level probes (eBPF) can enrich spans with low-level timing and resource signals for post-mortems and capacity planning [84]. For model/pipeline quality, integrate data-quality assertions and evaluator outputs into the same telemetry graph to gate rollouts and catch regressions [47, 37, 128, 49]. This end-to-end instrumentation supports bias/drift monitoring and incident response required by runtime governance frameworks [36].

Orchestration Gateways.. Introduce a gateway service that abstracts over execution backends and routes requests to LangChain or LangGraph based on policy: e.g., simple stateless paths to chains for latency/cost, complex stateful flows to

graphs for control and recovery. The gateway should attach LangSmith observers automatically (policy-driven sampling) and enforce organization-wide SLOs, cost budgets, and compliance checks [120, 72, 36]. Architectural patterns from large-scale agent platforms and software architecture for MAS (fallbacks, arbitration, rollback semantics) inform gateway policies [150, 16]. For periodic, batch, or dependency-heavy jobs, the gateway can interoperate with schedulers (e.g., Airflow) to trigger refreshes and evaluations [140, 123]. Finally, integrate security/testing suites (e.g., RAS-Eval) as pre-deployment or canary stages in the gateway to harden multi-agent behaviors under adversarial conditions [147].

Collectively, these middleware strategies operationalize a clean separation of concerns *construction* (LangChain), *orchestration* (LangGraph), and *observability/evaluation* (LangSmith) while preserving swap-ability, auditability, and performance across heterogeneous agentic AI workloads [123, 118, 117, 120].

Discussion and Implications

Hybrid interoperability across LangChain, LangGraph, and LangSmith delivers a pragmatic separation of concerns across the LLMOps/RAGOps lifecycle: rapid component composition and domain adaptation (*LangChain*), explicit runtime control for stateful, long-horizon, and multi-agent procedures (*LangGraph*), and end-to-end tracing, evaluation, and governance (*LangSmith*) [118, 117, 120, 123, 124]. This stack-level complementarity improves flexibility (swap-ability of components), scalability (concurrency and recoverability for complex graphs), and maintainability (traceable runs, dataset/run registries, and CI-gated evaluations) while aligning with emerging best practices for production agentic systems [47, 37, 128, 49].

These benefits come with integration challenges. Middleware introduces operational overhead and potential version skew; more critically, keeping *state semantics* consistent across chains, graphs, and observers requires disciplined schema governance and provenance tracking to avoid silent failures [123]. At production scale, observability must be right-sized: fine-grained telemetry (e.g., kernel/eBPF signals, span-rich traces) is invaluable for post-mortems and capacity planning but can inflate costs and surface privacy risks without selective sampling and retention controls [84, 85, 72, 121]. Regulated contexts demand runtime governance and auditability policy checks, incident response, and verifiable audit trails beyond mere logging [13, 101, 36]. For multi-agent workloads, robustness depends on explicit operational semantics (timeouts, arbitration, rollbacks) to mitigate failure modes such as oscillations and unbounded retries [127, 40, 24].

Looking forward, deeper native interoperability is likely. Protocol- and middleware-centric proposals (e.g., standardized tool interfaces, judge/guardrail adapters) point to shared execution schemas and unified agent-state ontologies spanning construction, orchestration, and measurement planes [152, 151]. On the control side, graph-theoretic optimization and verification (cycle control, branch pruning, DAG-verified reasoning) can yield safer, more efficient hybrid pipelines without sacrificing expressiveness [143, 112]. On the measurement side, reproducible evaluation mixing calibrated LLM-as-a-judge with human review and security test suites will increasingly gate rollouts via CI/CD [128, 37, 49, 129, 131, 132, 133, 35, 147]. Domain benchmarks for planning and real-world tasks will further pressure standardization of traces and state schemas across frameworks [145, 146]. Finally, efficiency considerations (latency/cost/energy) will favor IDEs and orchestration gateways that can switch between chain-first and graph-first execution while preserving LangSmith observability hooks and governance guarantees [149, 150, 16, 120]. In aggregate, the ability to compose these frameworks leveraging each where it is strongest will define the next generation of reliable, governable agentic AI platforms.

4. Applications and Use Cases

Contemporary agentic stacks fuse retrieval, structured control, and measurement to deliver dependable end-to-end applications. As summarized in Figure 4, LangChain targets construction (development), LangGraph handles orchestration, and LangSmith provides observability and evaluation forming a continuous pipeline from prototyping to monitored production. We ground these roles in deployments across science, enterprise, and operations; see Table 7 for a compact capability–use-case mapping.

4.1. LangChain Applications

LangChain's LCEL-centered, componentized model (chains, tools, retrievers, memory, agents) makes it well suited for assembling pipelines that integrate external knowledge and APIs [118, 119, 43, 104]. In knowledge-intensive assistants, LangChain operationalizes Retrieval-Augmented Generation (RAG) patterns query reformulation, top-k retrieval, and evidence-grounded synthesis building on foundational work and domain adaptations [125, 22, 115, 42]. Conversational agents retain dialogue context, invoke tools, and return verifiable answers, enabling business support, tutoring, and scientific helpdesks with transparent provenance [160, 161]. Chaining retrievers and LLMs also supports document-grounded chat with explicit citations and audit trails for literature curation and compliance [162]. For multi-step automation (report generation, policy checks, data annotation), LangChain's prompt and tool abstractions enable succinct orchestration with a clear path to migrate control-heavy logic to graph runtimes as complexity grows [163]. In scholarly workflows, citation and bibliographic tooling interoperate with templating to emit machine-auditable references, improving reproducibility [164]. Corresponding patterns appear in Table 7 under RAG, simple chaining, hybrid API integration, and educational prototyping.

4.2. LangGraph Applications

LangGraph targets tasks where control must adapt over long horizons with explicit state and branching semantics. In graph-structured RAG and planning, it coordinates search, reason and verify cycles, improving reliability over linear chains for difficult queries and evolving corpora [165, 166]. Multi-agent systems where specialized agents decompose and

Figure 4: **Application-focused role map of agentic AI toolchains.** The three bands show: LangChain development (chatbots, tool-using agents, prototyping); LangGraph orchestration (multi-agent workflows, RAG, human-in-the-loop); LangSmith observability (debugging/tracing, LLM-as-judge evaluation, prompt playground).

exchange subresults benefit from shared state and decision nodes, aligning with evidence from embodied/zero-shot planning and agent benchmarks [167, 145]. Reports span software-engineering agents, traffic and operations controllers, and machine-translation pipelines that require retries, interrupts, and parallel branches [25, 26, 139, 126]. Sector-specific uses include wireless network agents and test-analytics pipelines that combine domain tools with LLM reasoning under graph control [168, 169]. Prompting methods such as Chain-of-Thought further enhance stepwise reasoning within these graphs [170]. Continual-learning evaluations such as SWE-Bench-CL motivate explicit state and modular subgraphs for durable performance over time [155]. Table 7 reflects these strengths under stateful multi-agent workflows, iterative execution, complex routing, and hybrid API+LLM orchestration.

4.3. LangSmith Applications

LangSmith provides the measurement and assurance substrate across development and production: fine-grained traces, dataset and run registries, evaluator orchestration, and dashboards for cost, latency, and quality tracking [171, 120]. In RAG and agent pipelines, it exposes failure points such as retrieval drift, misrouted tools, and prompt regressions that are otherwise opaque [124]. Evaluation-driven development is supported via human and model judges, hypothesis-guided scoring, and industry-facing suites, enabling calibrated A/B tests and CI gates [132, 133, 37, 128, 49, 129, 131, 35]. Its tracing philosophy complements emerging agent observability and operations stacks and system-level telemetry for root-cause analysis [121, 84, 85]. These capabilities are pertinent in regulated or safety-critical deployments requiring accountability, drift and bias surveillance, and reproducible audits [172]. The corresponding entries in Table 7 include visual tracing and debugging, prompt playgrounds, LLM-as-judge evaluation, and production-grade observability.

Pragmatic composition. In practice, teams often begin with LangChain for component integration, promote recurrent or conditional logic to LangGraph as complexity accrues, and instrument both with LangSmith for observability and evaluation. This progression preserves rapid iteration while adding deterministic control and measurable reliability as applications mature, consistent with the capability–application alignment summarized in Table 7.

5. Challenges and Limitations

5.1. Technical Limitations

LangChain, LangGraph, and LangSmith offer powerful capabilities for developing agentic AI workflows; however, each framework exhibits distinct technical limitations that constrain their broader applicability and deployment in production environments. These limitations span concerns around execution complexity, scalability of orchestration, runtime observability, and user interpretability. Addressing these constraints is critical for building robust, transparent, and maintainable LLM-driven agent systems.

LangChain's modular chain-based architecture, while intuitive for simple applications, becomes increasingly difficult to

Table 7: Framework Suitability Across Applications and Aspects Based on Literature

Application / Aspect	LangChain	LangGraph	LangSmith	Supporting Literature Details / Citation
Simple LLM chaining (e.g., prompt → response → parse)	✓			Sequential and compositional reasoning architectures for LLM workflows [159, 173].
Retrieval-Augmented Generation (RAG) pipelines	✓	✓		Foundational RAG frameworks and enhancements [125, 174, 175].
Stateful Multi-agent Workflows		✓		Multi-agent LLM coordination, memory, and collaboration [176, 145, 122].
Cyclical / Iterative Task Execution		✓		Iterative refinement and planning in LLM agents [156, 177, 178].
Complex Control Flow / Conditional Routing		✓		Control flow and modular routing in language model workflows [179, 163, 180].
Visual Tracing and Debugging of Chains			✓	LLM execution visualization, observability, and debugging tools [138, 181, 121].
Prompt Engineering and Playground Testing			✓	Systematic prompt engineering and evaluation platforms [136, 182, 135].
Evaluation using LLM-as-a-Judge			✓	LLM evaluation methodologies and LLM-as-judge techniques [132, 48, 183].
Latency, Token Usage, and Error Monitoring			✓	Monitoring frameworks and efficiency metrics for LLM systems [148, 49, 149].
Educational Prototyping (with LangFlow)	✓			Prototyping and synthetic data generation for LLM education [184, 185, 186].
Production-grade Observability for LLMs			✓	Best practices for observability and production monitoring [121, 187, 188].
Hybrid Integration with External APIs	✓	✓		Hybrid workflow orchestration combining LLMs and APIs [189, 163, 190].

scale as the number of components grows [60, 191]. Each additional chain, tool, or memory module increases pipeline complexity, making debugging and modification challenging. The stateless nature of LangChain by default also limits its capacity to handle multi-turn dialogues or tasks requiring persistent memory unless manually engineered, which adds additional technical debt [192]. Furthermore, its DAG-based flow limits support for conditional branches, cycles, or dynamic agent behaviors common in autonomous systems [193, 141].

LangGraph, designed to overcome some of LangChain's structural rigidity, introduces graph-based orchestration with support for stateful and cyclical workflows [194]. However, this power comes at the cost of increased architectural complexity. Designing, maintaining, and debugging large state graphs can be cognitively demanding, especially for developers unfamiliar with state machine principles [25, 195]. Moreover, LangGraph's state persistence mechanisms, while flexible, may introduce performance bottlenecks in high-frequency inference settings, as shared state must be serialized and deserialized across transitions [142]. Its interpretability also diminishes as workflows grow in node count and interconnectivity, making runtime tracing non-trivial [152]. Moreover, frameworks like LangChain and LangGraph are considered "tactical approaches" for multi-agent systems which are useful for defining conversation patterns and termination conditions, but are often inconsistent and insufficient for addressing more fundamental system design flaws [127].

LangSmith, intended as an observability and debugging layer, presents different challenges. Since it operates passively over LangChain and LangGraph applications [196, 37, 74], its utility is highly dependent on integration fidelity. Capturing traces, logs, and evaluation metrics requires instrumenting all components appropriately, which may not always be feasible, especially in legacy or externally hosted workflows [196]. LangSmith's evaluation strategies (e.g., LLM-as-a-judge [129]) can introduce subjectivity and may not scale for large-scale deployments unless coupled with automated benchmarks [130, 131]. Furthermore, LangSmith lacks direct execution control, so mitigation of runtime failures or agentic drift must still be handled by the underlying framework [75].

5.2. Deployment and Integration Challenges

Challenges in integrating these toolchains into existing software ecosystems:

Integrating LangChain, LangGraph, and LangSmith into existing enterprise or research software stacks presents several friction points. LangChain's tightly coupled module design assumes end-to-end ownership of the orchestration logic [197, 150], which may conflict with modular microservice architectures or containerized workflows used in production. Many systems require adherence to platform-specific standards (e.g., Kubernetes, Docker, RESTful APIs), and LangChain does not natively expose these interfaces without extensive wrapper engineering. LangGraph's dependency on persistent state also complicates deployment in stateless serverless architectures, often requiring custom database middleware or Redis-style in-memory caching [198]. While LangSmith is non-intrusive, its demand for specific runtime instrumentation and consistent logging formats can break backward compatibility with legacy applications. The lack of native SDKs for non-Python languages, such as JavaScript, Java, and Go, also limits cross-platform adoption. Finally, the significant computational requirements

Table 8: Comparative Summary of Technical Limitations

Framework	Complexity	Scalability	Interpretability
LangChain	High with nested chains	Limited by linear DAGs	Moderate, few tracing tools
LangGraph	Very high in large graphs	Moderate with shared state	Low in complex graphs
LangSmith	Low (monitoring only)	Limited by integration effort	High for observability

for deploying large-scale LLMs with these frameworks demand proper integration with High-Performance Computing (HPC) resources. These challenges make a seamless deployment of these agentic AI frameworks a non-trivial task for organizations with diverse infrastructure.

Beyond these technical hurdles, there is a critical need for an automated cybersecurity solution within this framework to identify and exploit vulnerabilities efficiently without extensive human intervention [199, 147].

Issues with system compatibility and inter-framework communication:

System compatibility across LangChain, LangGraph, and LangSmith remains limited due to divergent architectural philosophies and evolving interfaces. LangChain and LangGraph are conceptually related but often incompatible when managing agent control flows [50], LangChain expects linear prompt pipelines [43], whereas LangGraph demands cyclical node-based orchestration [200]. Bridging between the two may require custom connectors or graph-transforming wrappers, especially when migrating applications from LangChain to LangGraph. Additionally, LangSmith's integration relies on decorators and event-based instrumentation [35], which may not align well with all execution models particularly asynchronous or distributed agent environments [34]. Coordination between these toolchains also lacks formal interface contracts or shared ontologies for state, error reporting, or evaluation logging. For example, LangSmith's observability layer cannot natively interpret LangGraph's internal node transitions without explicit developer mapping. Compatibility challenges also emerge when interfacing with external libraries like Hugging Face Transformers, vector databases, or third-party APIs, which may expect consistent input-output schemas. These mismatches necessitate glue code, custom serialization layers, or middleware agents, increasing engineering overhead and integration fragility.

5.3. Ethical and Responsible Use Challenges

As agentic AI toolchains such as LangChain, LangGraph, and LangSmith become increasingly central to building intelligent, autonomous applications, ethical and responsible use considerations are no longer ancillary they are foundational. The capacity of these toolchains to orchestrate decision-making logic, invoke large language models (LLMs), and observe agentic workflows raises multiple challenges related to bias propagation, transparency of operation, and the cultivation (or erosion) of user trust. These limitations not only affect individual systems but also influence how agentic AI applications are perceived, regulated, and adopted at scale.More specifically, these challenges can be summarized as:

1. **Bias Amplification and Disproportionate Impact** One of the most pressing ethical challenges across LangChain, LangGraph, and LangSmith is the unmitigated propagation of bias originating from the underlying language models and the interaction pipelines [119]. LangChain, for instance, allows developers to compose modular chains that incorporate tool invocation and memory components without enforcing fairness or bias checks at each layer [37, 201, 202]. This flexibility, while beneficial for experimentation, opens the door to unintended discrimination especially when used in sensitive domains such as hiring, healthcare, or education.
   LangGraph, with its capacity to coordinate multi-agent flows and persistent state, introduces another dimension of risk. Since decisions may compound over cyclical workflows [203, 153, 154] or affect the shared state used by other agents [126, 143], biases can be recursively reinforced over time [29], particularly in iterative planning or feedback loops [144, 105]. LangSmith, though observational in nature, plays a role here as well: while it supports tracing and evaluation [196], it lacks built-in interpretability tooling [196, 79] or automated ethical diagnostics that can surface or mitigate bias dynamically [204].
2. **Transparency and Explainability Limitations:** Transparency is a core principle for ethical AI [205, 206]. However, agentic systems built with these toolchains often lack explainable reasoning or traceable logic at runtime [207, 72]. In LangChain, developers define prompt templates and chain logic, but once composed, the execution becomes increasingly opaque particularly when multiple tools or nested agents are involved [52]. LangGraph adds orchestration complexity through conditional branching and event-driven transitions, often making it difficult for users or even developers to determine why a particular decision path was taken [208]. This lack of transparency in control flow and decision provenance is problematic for high-stakes applications where auditability and accountability are essential. LangSmith, while providing monitoring infrastructure, is limited to collecting runtime metrics, logs, and traces [79, 209]. It does not yet support semantic introspection or ethical auditing, leaving users responsible for manually interpreting trace outputs. Thus, the toolchains lack unified capabilities for explainable agentic behavior, especially in contexts involving legal or societal scrutiny.
3. **Trust and Responsible Deployment Risks:** User trust is easily lost and difficult to regain, particularly in AI systems that exhibit autonomy [210, 211, 212, 213, 214]. LangChain, LangGraph, and LangSmith all suffer from

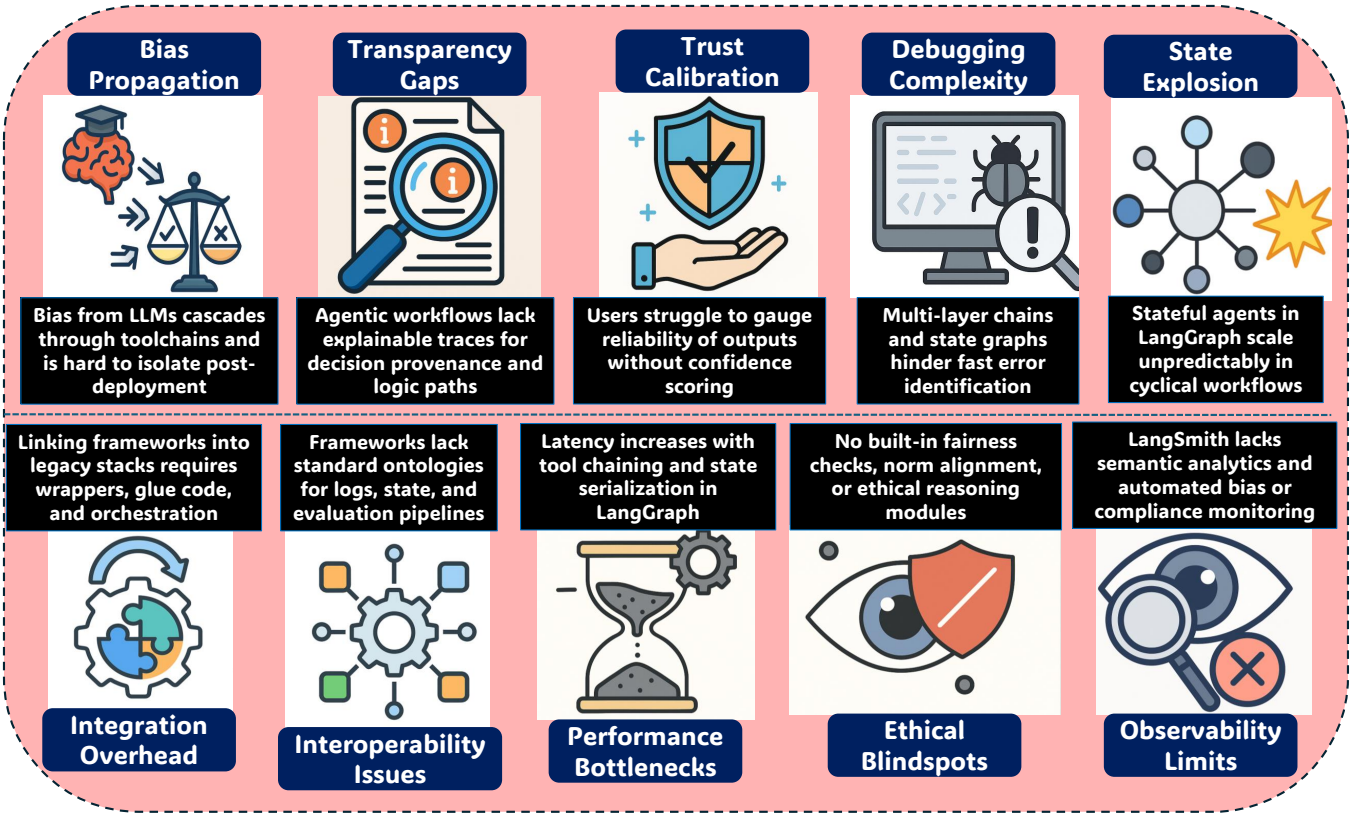

Figure 5: Illustration of the ten major challenges and limitations across LangChain, LangGraph, and LangSmith agentic AI frameworks, highlighting Bias Propagation, Transparency Gaps, Trust Calibration, Debugging Complexity, State Explosion, Integration Overhead, Interoperability Issues, Performance Bottlenecks, Ethical Blindspots, and Observability Limits as critical limitations in deployment, scaling, and responsible AI governance.

limitations in building robust trust scaffolds for end users. LangChain's outputs are heavily dependent on prompt formulation and context making it vulnerable to hallucinations or brittle behavior with slight input variations [215, 216]. LangGraph complicates this further by enabling long-lived state and multi-agent planning, which, if not sufficiently validated, can lead to erroneous or compounding failures that are difficult to detect [40, 126, 146]. LangSmith, while aimed at improving reliability through observability [47], is largely disconnected from end-user trust concerns it is more developer-facing and lacks direct UX integration for transparency or user feedback mechanisms [123, 217]. As such, agentic AI systems built with these toolchains currently lack robust affordances for trust calibration, such as uncertainty visualization, confidence intervals, or responsible user guidance. Moreover, integrating human-in-the-loop safety checks or fallbacks remains largely manual, increasing the risk of unmonitored or unintended agent behavior [201].

4. **Ethical Governance and Regulation Gaps:** From a systems perspective, none of the three toolchains currently enforce or even recommend ethical best practices yet. There are no native governance hooks for integrating fairness evaluators, audit trails, or compliance checks. LangChain's flexibility as a developer-centric tool makes it agnostic to ethical norms unless explicitly added [98, 218]. LangGraph's coordination logic could benefit from modular ethical agents that intervene in decision cycles [219], yet such components are not part of the standard stack [26, 139, 101]. LangSmith, though well-suited for instrumentation, does not include mechanisms for normative compliance such as GDPR-aligned data handling or model use transparency [220, 79, 201]. The lack of these features presents challenges for enterprise deployment, especially in regulated environments. Furthermore, none of the toolchains support ethical reasoning modules, crowdsourced norm integration, or socio-technical value alignment tools, leaving a critical gap in the agentic AI pipeline.

The agentic AI toolchains LangChain, LangGraph, and LangSmith, while enabling modularity, orchestration, and observability in large language model-driven applications, each face a spectrum of non-trivial challenges that inhibit robust, scalable, and ethically grounded deployment. As illustrated in Figure 5, these toolchains encounter ten critical limitations that span architectural, operational, and ethical dimensions. Bias propagation remains a persistent risk, particularly due to the opacity of prompt-engineered pipelines and uncontrolled tool integrations. Transparency gaps and observability limits hinder interpretability, debugging, and reproducibility especially when workflows become deeply nested or state-dependent. Trust calibration is difficult in multi-agent or human-in-the-loop systems where inconsistent LLM outputs can erode user confi-

dence. Moreover, debugging complexity and state explosion in LangGraph-oriented systems, as well as integration overhead across modular LangChain constructs, frequently escalate maintenance burdens. Interoperability issues across SDKs and toolchains also reduce developer agility. Performance bottlenecks such as latency from excessive tool chaining or asynchronous transitions compound these difficulties, while ethical blindspots such as hallucination amplification, value misalignment, and fairness violations demand proactive mitigation strategies. Importantly, LangSmith's observability layer while valuable cannot independently correct for these design-level limitations. Collectively, these challenges underscore the urgent need for integrated solutions that bridge orchestration flexibility with ethical safety, architectural transparency, and runtime trust. They also highlight that current toolchains, though complementary in their design intentions, are not yet fully synergistic in execution, especially under real-world complexity. To move toward sustainable, high-assurance agentic AI systems, the next stage of research and engineering must address these limitations through principled development, responsible governance, and novel infrastructure mechanisms. The following section explores potential solutions and outlines a forward-looking roadmap for enhancing LangChain, LangGraph, and LangSmith as foundational tools for building safe, efficient, and trustworthy agentic AI ecosystems.

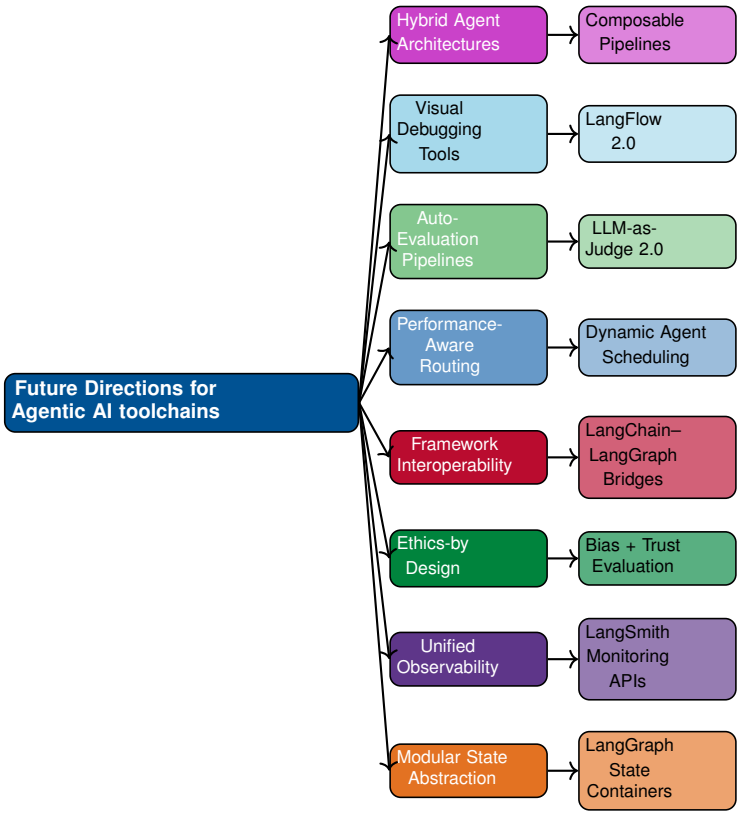

Figure 6: Future directions across LangChain, LangGraph, and LangSmith: state abstraction, observability, ethics, interoperability, performance-aware routing, evaluation, debugging, and hybrid architectures.

6. Potential Solution and Future Directions

Building on the limitations detailed in Section 5, we articulate a roadmap that couples technical remedies with ecosystem alignment and responsible governance. As synthesized in the "mind map" visualization (Figure 6), future work concentrates on eight tightly linked axes: modular state abstraction, unified observability, ethics-by-design, framework interoperability, performance-aware routing, automated evaluation, visual debugging, and hybrid agent architectures; the discussion below elaborates these facets and returns to Figure 6 where relevant.

Emerging Trends in Agentic AI Framework Development

A clear trend is the tighter fusion of symbolic orchestration with neural computation, marrying planning, memory, and evaluation around LLMs. For *LangChain*, this points to hybrid symbolic–neural chains that adapt tool invocation via uncertainty signals, retrieval feedback, and structured reasoning heuristics [191, 60]. *LangGraph* is converging on graph-based meta-control with concurrent and hierarchical roles, enabling agent collectives and long-horizon planning; extensions toward causal graphs and RL-conditioned scheduling are being explored in applied accounts [194, 25]. *LangSmith* is evolving from passive tracing to active, experiment-driven evaluation integrating data-quality assertions, judge pipelines, and lineage tracing for cross-framework runs [47, 79]. Collectively, these trends align with the state, observability, and evaluation branches in Figure 6.

Potential Integration Strategies Across Frameworks

Interoperation multiplies value when *LangChain* (component construction), *LangGraph* (stateful control), and *LangSmith* (observability/evaluation) share contracts. First, a shared agent/state schema should standardize payloads, tool APIs, and memory ontologies to enable lossless migration between linear chains and cyclic graphs [137, 118, 117]. Second, a lightweight LLM middleware can coordinate multiple backends and frameworks at scale, complementing LangChain-style composition with graph execution backplanes [151]. Third, unified telemetry harmonized traces and metrics spanning spans, events, and resource counters enables LangSmith to stitch end-to-end provenance across mixed pipelines; kernel- and system-level probes further deepen diagnosis [85, 84, 72]. Fourth, event-bus patterns (e.g., Kafka/Redis streams) decouple microservices and graphs while feeding experiment tracking and auditing [123]. Finally, treating agent workflows as optimizable and verifiable graphs reduces oscillation, dead ends, and unbounded retries through structural optimization and certification [143, 112, 127, 40].

Recommendations for Responsible Development and Deployment

Bias- and risk-aware components should extend LangChain with fairness/risk probes and uncertainty annotations on tool/-model calls [202, 172]. Normative checkpoints in LangGraph

can enforce policy constraints via approval gates, rollbacks, or masked data paths, aligning orchestration with enterprise governance [90, 36, 101]. Semantic explainability in LangSmith should move beyond raw traces toward causal summaries, impact analyses, and governance dashboards, consistent with emerging industry guidance on observability and trustworthy FMware [47, 79, 103, 206, 205]. These steps operationalize ethics-by-design and trust calibration in the streams highlighted by Figure 6.

Roadmap for Research and Development

Advancing cognitive architectures and autonomous reasoning. For LangChain, pursue neural–symbolic hybrids that fuse LCEL-style composition with deliberation and uncertainty-aware tool routing; develop hierarchical, bias-aware memory distinguishing episodic, semantic, and procedural layers [187, 122]. For LangGraph, formalize agent state ontologies and enable dynamic graph morphing with compression/verification to bound complexity [143, 112]. Across both, structured debate/planning and iterative refinement remain central [170, 159, 158, 156, 40]. LangSmith should add longitudinal analytics for drift/bias tracking beyond single runs.

Embedding ethics, trust, and governance mechanisms. Quantify bias propagation across chained and cyclical flows; integrate assurances and human-autonomy safeguards [211, 210, 212, 214]. Map capabilities to governance proposals and runtime controls [13, 36, 103], using TrustLLM-style taxonomies to prioritize mitigation across tasks and domains [172].

Scaling, interoperability, and real-world deployment. Elevate interop from ad hoc glue to standardized schemas, events, and logs spanning chains, graphs, and trials [123, 120]. Explore distributed/edge scenarios and knowledge exchange among LLMs [34, 190]. Leverage design patterns from multi-agent platforms and production case studies to harden SLOs and reliability [33, 30, 126, 26].

Establishing rigorous evaluation ecosystems. Adopt mixed human+LLM-as-judge protocols with hypothesis-guided rubrics; report uncertainty and agreement [128, 49, 129, 131, 132, 133, 35, 37]. Couple evaluations with pipeline assertions and data-quality checks [47]. Use planning/agent benchmarks (AgentBench, REALM-Bench) and security suites (RAS-Eval) to externalize gains [145, 146, 147].

Consolidation and composability for trustworthy agentic AI. Aim for consolidation without homogenization: a cohesive stack that preserves LangChain's ergonomics, LangGraph's expressiveness, and LangSmith's observability, supported by IDEs that switch orchestration modes while retaining shared state and common traces, and policy-ready hooks for audits and releases [39, 70, 71, 31, 16].

Future Outlook

The automation industry is entering a convergence phase where agentic AI stacks become the control plane for end-to-end digital operations [11, 221, 222]. In the near term, enterprises will move from brittle, script-driven automations toward graph-formalized workflows with explicit state, policies, and verifiable subgraphs. This shift enables machines to reason over long horizons, recover from partial failures, and coordinate multi-agent roles across data, applications, and physical assets. As orchestration matures, execution will increasingly be driven by performance-aware routing: tasks will flow dynamically between skills, tools, and models based on latency, cost, data sensitivity, and confidence, with adaptive throttling and circuit breakers providing production-grade resilience. Observability will evolve from log aggregation to semantic understanding of decisions. Rather than treating traces as exhaust, organizations will enrich them with typed state, intent, and outcome annotations to power incident analysis, regression testing, and continuous improvement. Model behavior, tool calls, and human interventions will be captured as composable "episodes," supporting longitudinal analytics, drift detection, and governance audits. This semantic telemetry will feed back into planners and schedulers, closing the loop between monitoring and control so that systems self-tune under real workloads.

Moreover, trust will become a first-class system requirement, not an afterthought. Ethics-by-design will materialize as enforceable runtime constraints: normative checkpoints inside workflows, bias-aware memory modules, red-team simulators in CI, and operator-visible uncertainty and provenance. Human-in-the-loop will shift from manual escalation to structured supervisory protocols with clear guarantees on consent, reversibility, and accountability. Organizations will publish automation "safety cases" that combine verification artifacts (e.g., subgraph proofs), operational SLOs, and audit trails, making deployments auditable by design.

Additionally, interoperability will determine speed to value. A shared agent state schema, portable evaluation packs, and standardized trace formats will let teams compose build–orchestrate–evaluate loops across vendors and runtimes. Marketplace dynamics will emerge for reusable skills, evaluators, and guardrails, while edge and on-prem agents interoperate with cloud backends through policy-aware event buses. The same patterns will span verticals from precision manufacturing and supply-chain control to clinical pathways, financial operations, customer experience, and critical infrastructure blending digital twins and simulation-in-the-loop with live operations.

Figure below offers a practical checklist for this evolution: modular state abstraction, unified observability, ethics-by-design, framework bridges, performance-aware routing, auto-evaluation pipelines, visual debugging, and hybrid agent architectures. Taken together, these directions signal a future where automation is not just faster but demonstrably safer, more transparent, and continuously improving turning agentic AI from a promising toolkit into dependable industrial infrastructure.

7. Conclusion

We presented a comprehensive, end-to-end view of three complementary agentic AI toolchains LangChain, LangGraph, and LangSmith by clarifying their core roles, architectural commitments, and operational trade-offs within a unified lifecycle. We showed that effective systems rarely rely on a single abstraction: chains excel at rapid composition and straightfor-

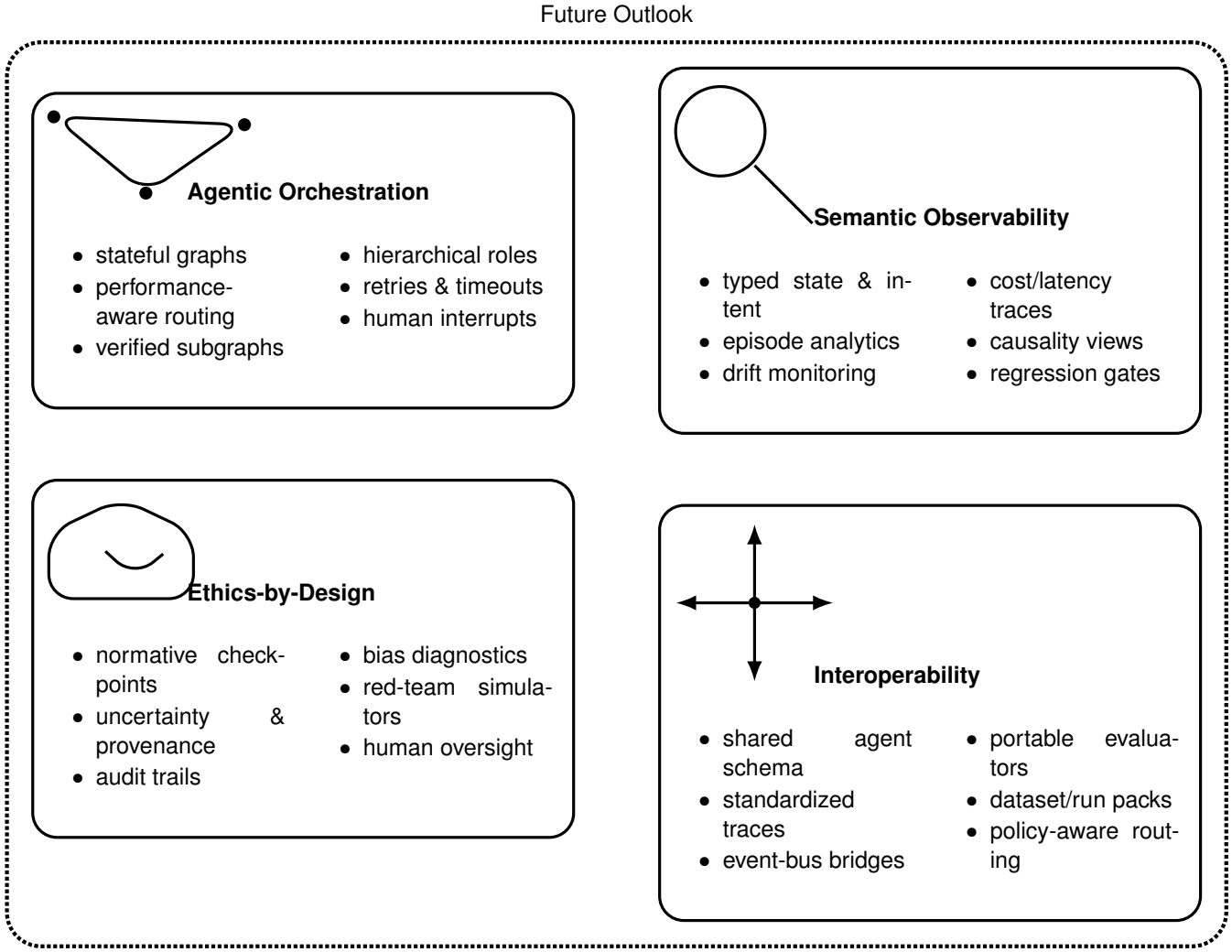

Figure 7: Future outlook for agentic AI automation. The four panels emphasize Agentic Orchestration, Semantic Observability, Ethics-by-Design, and Interoperability.

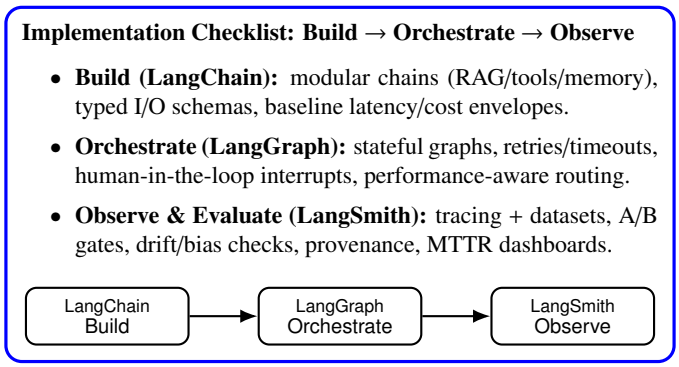

Figure 8: Implementation checklist connecting Build (LangChain), Orchestrate (LangGraph), and Observe (LangSmith).

ward retrieval-centric tasks; graphs become essential as workflows demand long-horizon control, branching, and shared state; and observability layers transform opaque behavior into measurable, auditable signals that drive iteration and governance. Building on this framing, we articulated a comparative taxonomy, distilled functional and operational differences, and proposed a benchmarking protocol that focuses not only on throughput and latency but also on developer-centric measures such as debugging resolution time. We further compiled application patterns across science, enterprise, and operations; documented conversion motifs for migrating between chaining and graph orchestration; and outlined middleware strategies that standardize state, events, and telemetry across heterogeneous stacks. Finally, we synthesized the principal challenges complexity growth, state explosion, interoperability gaps, and ethical blind spots and organized them into a roadmap of concrete directions spanning state abstraction, unified observability, ethics-by-design, and cross-framework standards.

The key contribution of this work is a practical, integrative perspective that helps teams select, combine, and evolve these toolchains with intent. Rather than debating which framework "wins," we argue for composition: use chains to build quickly, graphs to orchestrate reliably, and observability to learn contin-

uously. The proposed taxonomy and methodology convert that principle into action, offering a vocabulary for design reviews, a template for reproducible evaluations, and guidance for productionizing hybrid agents. In doing so, the article advances both research and practice: researchers gain a structured baseline for studying agent behavior under clear execution semantics; practitioners acquire patterns to reduce operational risk while preserving velocity. Looking ahead, we see momentum toward shared schemas for agent state, portable evaluation assets, and IDE-grade tooling that fluidly toggles between build, orchestrate, and observe modes. As these elements mature, agentic AI can move from promising prototypes to dependable infrastructure systems that reason with external knowledge, adapt under uncertainty, and remain accountable under real-world constraints. The path forward is not a single framework but a coherent stack, assembled with standards and measured with rigor, that enables trustworthy, high-performance applications at scale.

Declarations

The authors declare no conflicts of interest.

References

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al., Gpt-4 technical report, arXiv preprint arXiv:2303.08774 (2023).

[2] G. Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, et al., Gemini: a family of highly capable multimodal models, arXiv preprint arXiv:2312.11805 (2023).

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al., Llama: Open and efficient foundation language models, arXiv preprint arXiv:2302.13971 (2023).

[4] Y. Annepaka, P. Pakray, Large language models: a survey of their development, capabilities, and applications, Knowledge and Information Systems 67 (2025) 2967–3022.

[5] V. Bhatt, Z. Yu, Y. Hou, B. Khan, K. Dajani, J. Jin, From prompts to performance: Leveraging llms for enhanced educational ai interactions, in: 2025 IEEE Conference on Artificial Intelligence (CAI), IEEE, 2025, pp. 188–195.

[6] J. Haltaufderheide, R. Ranisch, The ethics of chatgpt in medicine and healthcare: a systematic review on large language models (llms), NPJ digital medicine 7 (2024) 183.

[7] M. Siino, M. Falco, D. Croce, P. Rosso, Exploring llms applications in law: A literature review on current legal nlp approaches, IEEE Access (2025).

[8] S. Al Faraby, A. Romadhony, et al., Analysis of llms for educational question classification and generation, Computers and Education: Artificial Intelligence 7 (2024) 100298.

[9] S. Rasnayaka, G. Wang, R. Shariffdeen, G. N. Iyer, An empirical study on usage and perceptions of llms in a software engineering project, in: Proceedings of the 1st International Workshop on Large Language Models for Code, 2024, pp. 111–118.

[10] C. K. Reddy, P. Shojaee, Towards scientific discovery with generative ai: Progress, opportunities, and challenges, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, 2025, pp. 28601–28609.

[11] R. Sapkota, K. I. Roumeliotis, M. Karkee, Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges, arXiv preprint arXiv:2505.10468 (2025).

[12] R. Sapkota, M. Karkee, Object detection with multimodal large vision-language models: An in-depth review, Information Fusion 126 (2026) 103575. URL: https://www.sciencedirect.com/science/article/pii/S1566253525006475. doi:https://doi.org/10.1016/j.inffus.2025.103575.

[13] Y. Shavit, S. Agarwal, M. Brundage, S. Adler, C. O'Keefe, R. Campbell, T. Lee, P. Mishkin, T. Eloundou, A. Hickey, et al., Practices for governing agentic ai systems, Research Paper, OpenAI (2023).

[14] D. B. Acharya, K. Kuppan, B. Divya, Agentic ai: Autonomous intelligence for complex goals–a comprehensive survey, IEEe Access (2025).

[15] R. Sapkota, K. I. Roumeliotis, M. Karkee, Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges, Information Fusion 126 (2026) 103599. URL: https://www.sciencedirect.com/science/article/pii/S1566253525006712. doi:https://doi.org/10.1016/j.inffus.2025.103599.

[16] K. Tallam, From autonomous agents to integrated systems, a new paradigm: Orchestrated distributed intelligence, arXiv preprint arXiv:2503.13754 (2025).

[17] L. Hughes, Y. K. Dwivedi, T. Malik, M. Shawosh, M. A. Albashrawi, I. Jeon, V. Dutot, M. Appanderanda, T. Crick, R. De', et al., Ai agents and agentic systems: A multi-expert analysis, Journal of Computer Information Systems (2025) 1–29.

[18] W. Guan, Y. Fang, Optimizing web-based ai query retrieval with gpt integration in langchain a cot-enhanced prompt engineering approach, arXiv preprint arXiv:2506.15512 (2025).

[19] N.-L. Hsueh, W.-T. Wang, Development and evaluation of learning portfolio query system based on langchain framework, Engineering Proceedings 92 (2025) 40.

[20] J. Jeong, D. Gil, D. Kim, J. Jeong, Current research and future directions for off-site construction through langchain with a large language model, Buildings 14 (2024) 2374.

[21] D. Goyal, A. Gautam, Introduction to langchain framework, Textual Intelligence: Large Language Models and Their Real-World Applications (2025) 253–285.

[22] F. Bianchini, Retrieval-augmented generation, in: Engineering Information Systems with Large Language Models, Springer, 2025, pp. 139–172.

[23] T. Dhabalia, S. Bhate, K. Singh, B. Cerejo, D. Bhagat, A comparative study of rag and fine-tuned transformer models for domain-specific chatbots, in: 2025 International Conference on Intelligent and Cloud Computing (ICoICC), IEEE, 2025, pp. 1–6.

[24] Z. Yu, S. Liu, P. Denny, A. Bergen, M. Liut, Integrating small language models with retrieval-augmented generation in computing education: Key takeaways, setup, and practical insights, in: Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1, 2025, pp. 1302–1308.

[25] J. Wang, Z. Duan, Empirical research on utilizing llm-based agents for automated bug fixing via langgraph, arXiv preprint arXiv:2502.18465 (2025).

[26] H. Chen, Y. Ding, Implementing traffic agent based on langgraph, in: Fourth International Conference on Intelligent Traffic Systems and Smart City (ITSSC 2024), volume 13422, SPIE, 2025, pp. 582–587.

[27] C. Yu, Z. Cheng, H. Cui, Y. Gao, Z. Luo, Y. Wang, H. Zheng, Y. Zhao, A survey on agent workflow–status and future, in: 2025 8th International Conference on Artificial Intelligence and Big Data (ICAIBD), IEEE, 2025, pp. 770–781.

[28] H. Yang, Y. Pan, J. Xu, K. Liu, Amico: An event-driven modular framework for persistent and embedded autonomy, arXiv preprint arXiv:2507.14513 (2025).

[29] L. Mei, J. Yao, Y. Ge, Y. Wang, B. Bi, Y. Cai, J. Liu, M. Li, Z.-Z. Li, D. Zhang, et al., A survey of context engineering for large language models, arXiv preprint arXiv:2507.13334 (2025).

[30] K. Pelluru, Langchain & langgraph in production: Architectures for multi-agent llm systems, Journal of Data and Digital Innovation (JDDI) 2 (2025) 1–9.

[31] U. M. Borghoff, P. Bottoni, R. Pareschi, Beyond prompt chaining: The tb-cspn architecture for agentic ai, Future Internet (2025).

[32] W. Jiang, F. Hu, Artificial intelligence agent-enabled predictive maintenance: Conceptual proposal and basic framework, Computers 14 (2025) 329.

[33] T. Taulli, G. Deshmukh, Introduction to langgraph, in: Building Gen-

erative AI Agents: Using LangGraph, AutoGen, and CrewAI, Springer, 2025, pp. 209–235.
[34] W. Wei, N. Chen, Y. Li, The internet of large language models: An orchestration framework for llm training and knowledge exchange toward artificial general intelligence, arXiv preprint arXiv:2501.06471 (2025).
[35] V. Koc, Tiny qa benchmark++: Ultra-lightweight, synthetic multilingual dataset generation & smoke-tests for continuous llm evaluation, arXiv preprint arXiv:2505.12058 (2025).
[36] C. L. Wang, T. Singhal, A. Kelkar, J. Tuo, Mi9–agent intelligence protocol: Runtime governance for agentic ai systems, arXiv preprint arXiv:2508.03858 (2025).
[37] S. McAvinue, K. Dev, Comparative evaluation of large language models using key metrics and emerging tools, Expert Systems 42 (2025) e13719.
[38] T. Taulli, G. Deshmukh, Building generative ai agents, Springer (2025).
[39] K. Huang, Agentic AI, Springer, 2025.
[40] E. Y. Chang, L. Geng, Sagallm: Context management, validation, and transaction guarantees for multi-agent llm planning, arXiv preprint arXiv:2503.11951 (2025).
[41] Y. G. Shrestha, Cache-augmented generation in rag pipelines: Fast and memory-efficient approach to multi-agent knowledge query systems, The University of Southern Mississippi (2025).
[42] E. Do, A. Sunkarapalli, A. Patil, N. Akalwadi, R. Ghaleb, Building an intelligent qa/chatbot for transportation with langchain and open source llms, Virginia Tech (2025).
[43] D. Grigorov, Langchain and python: Basics, in: Intermediate Python and Large Language Models, Springer, 2025, pp. 1–57.
[44] C. H. Savage, A. Kanhere, V. Parekh, C. P. Langlotz, A. Joshi, H. Huang, F. X. Doo, Open-source large language models in radiology: a review and tutorial for practical research and clinical deployment, Radiology 314 (2025) e241073.
[45] M. Li, P. Zhang, W. Xing, Y. Zheng, K. Zaporojets, J. Chen, R. Zhang, Y. Zhang, S. Gong, J. Hu, et al., Using large language models to tackle fundamental challenges in graph learning: A comprehensive survey, arXiv preprint arXiv:2505.18475 (2025).
[46] R. Aratchige, W. Ilmini, Llms working in harmony: A survey on the technological aspects of building effective llm-based multi agent systems, arXiv preprint arXiv:2504.01963 (2025).
[47] S. Shankar, H. Li, P. Asawa, M. Hulsebos, Y. Lin, J. Zamfirescu-Pereira, H. Chase, W. Fu-Hinthorn, A. G. Parameswaran, E. Wu, Spade: Synthesizing data quality assertions for large language model pipelines, arXiv preprint arXiv:2401.03038 (2024).
[48] Y. Chen, J. Zhao, H. Han, A survey on collaborative mechanisms between large and small language models, arXiv preprint arXiv:2505.07460 (2025).
[49] A. Yehudai, L. Eden, A. Li, G. Uziel, Y. Zhao, R. Bar-Haim, A. Cohan, M. Shmueli-Scheuer, Survey on evaluation of llm-based agents, arXiv preprint arXiv:2503.16416 (2025).
[50] K. Huang, J. Huang, Ai agent tools and frameworks, in: Agentic AI: Theories and Practices, Springer, 2025, pp. 23–50.
[51] C. Ram, Building Practical Industrial AI Agents: Implementation Examples Using Large Language Models, Master's thesis, University of South-Eastern Norway, 2025.
[52] T. C. da Silva, Extracting knowledge graphs from user stories using langchain, arXiv preprint arXiv:2506.11020 (2025).
[53] Y. Chen, L. Zhang, X. Zhu, H. Zhou, Z. Ren, Optmetaopenfoam: Large language model driven chain of thought for sensitivity analysis and parameter optimization based on cfd, arXiv preprint arXiv:2503.01273 (2025).
[54] W. Jin, N. Wang, L. Zhang, X. Tian, B. Shi, B. Zhao, A review of ai-driven automation technologies: Latest taxonomies, existing challenges, and future prospects., Computers, Materials & Continua 84 (2025).
[55] T. Petrova, A. Puzikov, B. Bliznukov, R. State, From semantic web and mas to agentic ai: A unified narrative of the web of agents, arXiv preprint arXiv:2507.10644 (2025).
[56] M. Muhoberac, A. Parikh, N. Vakharia, S. Virani, A. Radujevic, S. Wood, M. Verma, D. Metaxotos, J. Soundararajan, T. Masquelin, et al., State and memory is all you need for robust and reliable ai agents, arXiv preprint arXiv:2507.00081 (2025).
[57] R. Y. Maragheh, Y. Deldjoo, The future is agentic: Definitions, perspectives, and open challenges of multi-agent recommender systems, arXiv preprint arXiv:2507.02097 (2025).
[58] J. Zhang, I. Arawjo, Chainbuddy: An ai-assisted agent system for generating llm pipelines, in: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, 2025, pp. 1–21.
[59] O. Alhava, T. Arola, O. Torro, M. Järvenpää, T. Järvinen, B. Ruottinen, Ai-agent application for semantic data enrichment in ventila-tion systems using national nomenclature for ifc and gs1-based product information, Data in Architecture and Construction Workshop (2025).
[60] C. Jeong, Beyond text: Implementing multimodal large language model-powered multi-agent systems using a no-code platform, arXiv preprint arXiv:2501.00750 (2025).
[61] S. Shubbar, Advancing Autism Spectrum Disorder Diagnosis: A Phenotype-Genotype Co-Analysis and Retrieval-Augmented LLM Framework for Clinical Decision Support, Ph.D. thesis, Kent State University, 2025.
[62] S. Shah, A. Kelly, S. Tripathy, A. Ladak, W. Tang, allarma: Llm-based application for operational data, alarms and networks, Authorea Preprints (2025).
[63] H. Zhang, J. Haskell, Y. Frost, Flow state: Humans enabling ai systems to program themselves, arXiv preprint arXiv:2504.03771 (2025).
[64] J. Yu, Y. Ding, H. Sato, Dyntaskmas: A dynamic task graph-driven framework for asynchronous and parallel llm-based multi-agent systems, arXiv preprint arXiv:2503.07675 (2025).
[65] Z. Yu, M. Ma, C. Zhang, S. Qin, Y. Kang, C. Bansal, S. Rajmohan, Y. Dang, C. Pei, D. Pei, et al., Monitorassistant: Simplifying cloud service monitoring via large language models, in: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, 2024, pp. 38–49.
[66] X. Guo, D. Keivan, U. Syed, L. Qin, H. Zhang, G. Dullerud, P. Seiler, B. Hu, Controlagent: Automating control system design via novel integration of llm agents and domain expertise, arXiv preprint arXiv:2410.19811 (2024).
[67] Z. Liu, R. Zeng, D. Wang, G. Peng, J. Wang, Q. Liu, P. Liu, W. Wang, Agents4plc: Automating closed-loop plc code generation and verification in industrial control systems using llm-based agents, arXiv preprint arXiv:2410.14209 (2024).
[68] S. Yoon, J. Song, J. Li, Ai agent-based intelligent digital twins for building operations and maintenance, Journal of Building Engineering (2025) 112802.
[69] J. Cui, Z. Li, L. Xing, X. Liao, Safeguard-by-development: A privacy-enhanced development paradigm for multi-agent collaboration systems, arXiv preprint arXiv:2505.04799 (2025).
[70] J. G. Mathew, J. Rossi, Large language model agents, in: Engineering Information Systems with Large Language Models, Springer, 2025, pp. 173–205.
[71] S. Massoudi, M. Fuge, Agentic large language models for conceptual systems engineering and design, arXiv preprint arXiv:2507.08619 (2025).
[72] D. Moshkovich, H. Mulian, S. Zeltyn, N. Eder, I. Skarbovsky, R. Abitbol, Beyond black-box benchmarking: Observability, analytics, and optimization of agentic systems, arXiv preprint arXiv:2503.06745 (2025).
[73] W. Jin, H. Du, B. Zhao, X. Tian, B. Shi, G. Yang, A comprehensive survey on multi-agent cooperative decision-making: Scenarios, approaches, challenges and perspectives, arXiv preprint arXiv:2503.13415 (2025).
[74] N. Klüwer, Context over Categories-Implementing the Theory of Constructed Emotion, Ph.D. thesis, Technische Universität Wien, 2025.
[75] D. Moshkovich, S. Zeltyn, Taming uncertainty via automation: Observing, analyzing, and optimizing agentic ai systems, arXiv preprint arXiv:2507.11277 (2025).
[76] H. B. Ku, J. Shin, H. J. Lee, S. Na, I. Jeon, Multi-agent llm debate unveils the premise left unsaid, in: Proceedings of the 12th Argument mining Workshop, 2025, pp. 58–73.
[77] X. Liu, P. Di, C. Li, J. Sun, J. Wang, Efficient function orchestration for large language models, arXiv preprint arXiv:2504.14872 (2025).
[78] P. Du, Omninova: A general multimodal agent framework, arXiv preprint arXiv:2503.20028 (2025).
[79] K. Vasilevski, B. Rombaut, G. K. Rajbahadur, G. A. Oliva, K. Gallaba, F. R. Cogo, J. Lin, D. Lin, H. Zhang, B. Chen, et al., The hitchhikers guide to production-ready trustworthy foundation model powered software (fmware), arXiv preprint arXiv:2505.10640 (2025).

[80] D. Dave, Y. Agarwal, H. Chouhan, S. Jambhulkar, Learning in langchain, in: World Congress on Smart Computing: Proceedings of WCSC 2024, Springer Nature, 2025, p. 29.
[81] H. Farías, J. González Aroca, D. Ortiz, Chatbot based on large language model to improve adherence to exercise-based treatment in people with knee osteoarthritis: System development, Technologies 13 (2025) 140.
[82] S. Rupprecht, Q. Gao, T. Karia, A. M. Schweidtmann, Multi-agent systems for chemical engineering: A review and perspective, arXiv preprint arXiv:2508.07880 (2025).
[83] B. Zhang, X. Li, H. Xu, Z. Jin, Q. Wu, C. Li, Topomas: Large language model driven topological materials multiagent system, arXiv preprint arXiv:2507.04053 (2025).
[84] Y. Zheng, Y. Hu, T. Yu, A. Quinn, Agentsight: System-level observability for ai agents using ebpf, arXiv preprint arXiv:2508.02736 (2025).
[85] T. Y. Liu, S. Soatto, M. Marchi, P. Chaudhari, P. Tabuada, Observability of latent states in generative ai models, Proceedings of Machine Learning Research vol vvv 1 (2025) 20.
[86] R. Modi, N. Reddy, S. S. Kodur, Debugmate: an ai agent for efficient on-call debugging in complex production systems, Discover Data 3 (2025) 33.
[87] S. Liu, M. Liu, H. Zhou, Z. Cui, Y. Zhou, Y. Zhou, W. Fan, G. Zhang, J. Shi, W. Xuan, et al., Verigui: Verifiable long-chain gui dataset, arXiv preprint arXiv:2508.04026 (2025).
[88] F. Fournier, L. Limonad, Y. David, Agentic ai process observability: Discovering behavioral variability, arXiv preprint arXiv:2505.20127 (2025).
[89] E. Gallois, A. Salili-James, S. T. Poon, A. Trebski, D. W. Redding, Leveraging large language models for ecological interpretation using an ebird chatbot case study, EcoEvoRxiv (2025).
[90] J. Syed, S. Syed, B. Aijaz, M. Ahmad, Orchestrating trustworthy text-to-sql: Leveraging stateful graph frameworks like langgraph for human-in-the-loop validation in enterprise environments, World Journal of Advanced Engineering Technology and Sciences 15 (2025) 1807–1817.
[91] C. Jeong, A study on the mcp x a2a framework for enhancing interoperability of llm-based autonomous agents, arXiv preprint arXiv:2506.01804 (2025).
[92] L. G. L. Talip, R. N. C. Recario, A dynamic query framework for research accessibility using openai and langchain, in: Intelligent Computing-Proceedings of the Computing Conference, Springer, 2025, pp. 149–166.
[93] N. Malik, L. Chhikara, Abhilakshay, A. Thakur, Legalease: Application development with langchain framework, Textual Intelligence: Large Language Models and Their Real-World Applications (2025) 323–364.
[94] A. Gruzd, J. Zhang, P. Mai, Graphoptima: A graph layout optimization framework for visualizing large networks, SoftwareX 29 (2025) 102034.
[95] T.-Y. Yang, Y. Chen, Y. Liang, M.-C. Yang, Leveraging on-demand processing to co-optimize scalability and efficiency for fully-external graph computation, ACM Transactions on Storage 21 (2025) 1–31.
[96] A. Saxena, A. Y. Chaudhari, A. Gupta, Agcv: An agentic framework for automating computer vision application, MethodsX (2025) 103424.
[97] S. A. Akheel, Observability in large language models: A framework for real-time applications, IJLRP-International Journal of Leading Research Publication 5 (2025).
[98] G. Arun, R. Syam, A. A. Nair, S. Vaidya, An integrated framework for ethical healthcare chatbots using langchain and nemo guardrails, AI and Ethics (2025) 1–12.
[99] I. M. Ibrahim, M. S. Attallah, S. O. Abdel Hamid, S. T. Zween, I. Abuhadrous, Leveraging large language models for document analysis and decision-making in ai chatbots, Advanced Sciences and Technology Journal 2 (2025) 1–16.
[100] A. Yang, Preparing public relations' practitioners for the ai era: Advancing pedagogical principles in public relations' artificial intelligence education, Journalism & Mass Communication Educator 80 (2025) 3–24.
[101] A. Ganguly, Ai governance and responsible ai, Springer (2025) 285–316.
[102] Z. Wu, S. Cho, U. Mohammed, C. Munoz, K. Costa, X. Guan, T. King, Z. Wang, E. Kazim, A. Koshiyama, Libvulnwatch: A deep assessment agent system and leaderboard for uncovering hidden vulnerabilities in open-source ai libraries, arXiv preprint arXiv:2505.08842 (2025).
[103] I. Strauss, I. Moure, T. O'Reilly, S. Rosenblat, Real-world gaps in ai governance research, arXiv preprint arXiv:2505.00174 (2025).
[104] S. Annam, M. Saxena, U. Kaushik, S. Mittal, Langchain: Simplifying development with language models, Textual Intelligence: Large Language Models and Their Real-World Applications (2025) 287–304.
[105] J. Zhang, Q. Zhang, Y. He, Y. Zhang, Copaa: Curriculum outline process automatic agent based on langgraph, in: International Conference on Intelligent Computing, Springer, 2025, pp. 66–78.
[106] B. Zhang, Z. Li, Z. Liu, H. Wang, Y. Ma, Integrating large language models into text animation: An intelligent editing system with inline and chat interaction, arXiv preprint arXiv:2506.10762 (2025).
[107] M. Abdelaal, S. Lokadjaja, G. Engert, Gllm: Self-corrective g-code generation using large language models with user feedback, arXiv preprint arXiv:2501.17584 (2025).
[108] N. Samadifar, Design and implementation of the UX & UI for a caregiver service, Ph.D. thesis, Politecnico di Torino, 2025.
[109] C. Chawla, S. Chatterjee, S. Gadadinni, P. Verma, S. Banerjee, Agentic ai: The building blocks of sophisticated ai business applications, Journal of AI, Robotics and Workplace Automation 3 (2024) 1. doi:10.69554/XEHZ1946.
[110] Y. Ren, Y. Liu, T. Ji, X. Xu, Ai agents and agentic ai-navigating a plethora of concepts for future manufacturing, arXiv preprint arXiv:2507.01376 (2025).
[111] K. M. Siva, The agentic ai framework: Enabling autonomous intelligence, International Journal of Innovative Research in Engineering and Multidisciplinary Physical Sciences 13 (2025) 1–4. doi:10.5281/zenodo.15029753.
[112] J. Fang, B. Zhang, C. Wang, J. Wan, Z. Xu, Graph of verification: Structured verification of llm reasoning with directed acyclic graphs, arXiv preprint arXiv:2506.12509 (2025).
[113] K. Barakati, A. Molak, C. Nelson, X. Zhang, I. Takeuchi, S. V. Kalinin, Llm-assisted causal discovery: Mapping hidden patterns in crystalline solids, Microscopy and Microanalysis 31 (2025) ozaf048–1106.
[114] A. Esashi, P. Lertpongrujikorn, S. Kato, M. A. Salehi, Action engine: Automatic workflow generation in faas, Future Generation Computer Systems 174 (2026) 107947.
[115] Y. Ke, L. Jin, K. Elangovan, H. R. Abdullah, N. Liu, A. T. H. Sia, C. R. Soh, J. Y. M. Tung, J. C. L. Ong, D. S. W. Ting, Development and testing of retrieval augmented generation in large language models–a case study report, arXiv preprint arXiv:2402.01733 (2024).
[116] Y. Bei, W. Zhang, S. Wang, W. Chen, S. Zhou, H. Chen, Y. Li, J. Bu, S. Pan, Y. Yu, et al., Graphs meet ai agents: Taxonomy, progress, and future opportunities, arXiv preprint arXiv:2506.18019 (2025).
[117] LangChain, Introduction to langgraph, 2024. URL: https://langchain-ai.github.io/langgraph/, accessed: 2025-07-16.
[118] LangChain, Langchain expression language (lcel), 2024. URL: https://python.langchain.com/docs/expression_language/, accessed: 2025-07-16.
[119] B. Auffarth, Generative ai with langchain, Birmingham, UK: Packt Publishing (2023).
[120] LangChain, Langsmith: A unified platform for debugging, testing, evaluating, and monitoring your llm applications, 2024. URL: https://www.langchain.com/langsmith, accessed: 2025-07-16.
[121] L. Dong, Q. Lu, L. Zhu, Agentops: Enabling observability of llm agents, arXiv preprint arXiv:2411.05285 (2024).
[122] Z. Zhang, Q. Dai, X. Bo, C. Ma, R. Li, X. Chen, J. Zhu, Z. Dong, J.-R. Wen, A survey on the memory mechanism of large language model based agents, ACM Transactions on Information Systems (2024).
[123] X. Xu, H. Weytjens, D. Zhang, Q. Lu, I. Weber, L. Zhu, Ragops: Operating and managing retrieval-augmented generation pipelines, arXiv preprint arXiv:2506.03401 (2025).
[124] G. Mialon, R. Dessì, M. Lomeli, C. Nalmpantis, R. Pasunuru, R. Raileanu, B. Rozière, T. Schick, J. Dwivedi-Yu, A. Celikyilmaz, et al., Augmented language models: a survey, arXiv preprint arXiv:2302.07842 (2023).
[125] P. Lewis, E. Perez, A. Piktus, F. Petroni, et al., Retrieval-augmented generation for knowledge-intensive nlp tasks, Advances in Neural Information Processing Systems (NeurIPS) 33 (2020) 9459–9474.
[126] Z. Duan, J. Wang, Exploration of llm multi-agent application implementation based on langgraph+ crewai, arXiv preprint arXiv:2411.18241 (2024).
[127] M. Cemri, M. Z. Pan, S. Yang, L. A. Agrawal, B. Chopra, R. Tiwari,

K. Keutzer, A. Parameswaran, D. Klein, K. Ramchandran, et al., Why do multi-agent llm systems fail?, arXiv preprint arXiv:2503.13657 (2025).
[128] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, et al., A survey on evaluation of large language models, ACM transactions on intelligent systems and technology 15 (2024) 1–45.
[129] J. Gu, X. Jiang, Z. Shi, H. Tan, X. Zhai, C. Xu, W. Li, Y. Shen, S. Ma, H. Liu, et al., A survey on llm-as-a-judge, arXiv preprint arXiv:2411.15594 (2024).
[130] R. Wang, J. Guo, C. Gao, G. Fan, C. Y. Chong, X. Xia, Can llms replace human evaluators? an empirical study of llm-as-a-judge in software engineering, Proceedings of the ACM on Software Engineering 2 (2025) 1955–1977.
[131] R. Hu, Y. Cheng, L. Meng, J. Xia, Y. Zong, X. Shi, W. Lin, Training an llm-as-a-judge model: Pipeline, insights, and practical lessons, in: Companion Proceedings of the ACM on Web Conference 2025, 2025, pp. 228–237.
[132] M. Li, H. Li, C. Tan, Hypoeval: Hypothesis-guided evaluation for natural language generation, arXiv preprint arXiv:2504.07174 (2025).
[133] H. Saini, M. T. R. Laskar, C. Chen, E. Mohammadi, D. Rossouw, Llm evaluate: An industry-focused evaluation tool for large language models, in: Proceedings of the 31st International Conference on Computational Linguistics: Industry Track, 2025, pp. 286–294.
[134] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, P. Liang, Lost in the middle: How language models use long contexts, arXiv preprint arXiv:2307.03172 (2023).
[135] X. Amatriain, Prompt design and engineering: Introduction and advanced methods, arXiv preprint arXiv:2401.14423 (2024).
[136] S. Schulhoff, M. Ilie, N. Balepur, K. Kahadze, A. Liu, C. Si, Y. Li, A. Gupta, H. Han, S. Schulhoff, et al., The prompt report: a systematic survey of prompt engineering techniques, arXiv preprint arXiv:2406.06608 (2024).
[137] Y. Mu, R. Wang, J. Zhai, C. Fang, X. Chen, J. Wu, A. Guo, J. Shen, B. Li, Z. Chen, Designing deep learning frameworks for llms: Challenges, expectations, and opportunities, arXiv preprint arXiv:2506.13114 (2025).
[138] M. Kahng, I. Tenney, M. Pushkarna, M. X. Liu, J. Wexler, E. Reif, K. Kallarackal, M. Chang, M. Terry, L. Dixon, Llm comparator: Visual analytics for side-by-side evaluation of large language models, in: Extended Abstracts of the CHI Conference on Human Factors in Computing Systems, 2024, pp. 1–7.
[139] J. Wang, Z. Duan, Agent ai with langgraph: A modular framework for enhancing machine translation using large language models, arXiv preprint arXiv:2412.03801 (2024).
[140] Wikipedia contributors, Apache airflow — Wikipedia, the free encyclopedia, 2025. URL: https://en.wikipedia.org/w/index.php?title=Apache_Airflow&oldid=1236967733, [Online; accessed 1-August-2025].
[141] H. Wu, C. Yue, L. Zhang, Y. Li, M. A. Imran, When distributed consensus meets wireless connected autonomous systems: A review and a dag-based approach, IEEE Network (2024).
[142] G. Sun, J. Zhou, B. Li, X. Gu, W. Wang, S. He, Ftgraph: A flexible tree-based graph store on persistent memory for large-scale dynamic graphs, in: 2024 IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 2024, pp. 39–50.
[143] M. Zhuge, W. Wang, L. Kirsch, F. Faccio, D. Khizbullin, J. Schmidhuber, Language agents as optimizable graphs, arXiv preprint arXiv:2402.16823 (2024).
[144] D. H. Kim, D. Jeong, S. Yadgarova, H. Shin, J. Son, H. Subramonyam, J. Kim, Plantogether: Facilitating ai application planning using information graphs and large language models, in: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, 2025, pp. 1–23.
[145] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang, et al., Agentbench: Evaluating llms as agents, arXiv preprint arXiv:2308.03688 (2023).
[146] L. Geng, E. Y. Chang, Realm-bench: A real-world planning benchmark for llms and multi-agent systems, arXiv preprint arXiv:2502.18836 (2025).
[147] Y. Fu, X. Yuan, D. Wang, Ras-eval: A comprehensive benchmark for security evaluation of llm agents in real-world environments, arXiv preprint arXiv:2506.15253 (2025).
[148] N. B. Brown, Enhancing trust in llms: Algorithms for comparing and interpreting llms, arXiv preprint arXiv:2406.01943 (2024).
[149] G. Bai, Z. Chai, C. Ling, S. Wang, J. Lu, N. Zhang, T. Shi, Z. Yu, M. Zhu, Y. Zhang, et al., Beyond efficiency: A systematic survey of resource-efficient large language models, arXiv preprint arXiv:2401.00625 (2024).
[150] M. Becattini, R. Verdecchia, E. Vicario, Sallma: A software architecture for llm-based multi-agent systems, in: 2025 IEEE/ACM International Workshop New Trends in Software Architecture (SATrends), IEEE, 2025, pp. 5–8.
[151] N. Guran, F. Knauf, M. Ngo, S. Petrescu, J. S. Rellermeyer, Towards a middleware for large language models, arXiv preprint arXiv:2411.14513 (2024).
[152] A. Sarkar, S. Sarkar, Survey of llm agent communication with mcp: A software design pattern centric review, arXiv preprint arXiv:2506.05364 (2025).
[153] N. H. Park, T. J. Callahan, J. L. Hedrick, T. Erdmann, S. Capponi, Leveraging chemistry foundation models to facilitate structure focused retrieval augmented generation in multi-agent workflows for catalyst and materials design, arXiv preprint arXiv:2408.11793 (2024).
[154] T. J. Callahan, N. H. Park, S. Capponi, Agentic mixture-of-workflows for multi-modal chemical search, arXiv preprint arXiv:2502.19629 (2025).
[155] T. Joshi, S. Chowdhury, F. Uysal, Swe-bench-cl: Continual learning for coding agents, arXiv preprint arXiv:2507.00014 (2025).
[156] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, et al., Self-refine: Iterative refinement with self-feedback, Advances in Neural Information Processing Systems 36 (2023) 46534–46594.
[157] E. Zelikman, et al., Self-refine: Iterative refinement for language models, arXiv preprint arXiv:2203.14465 (2022).
[158] F. Meng, et al., Iterative reasoning for complex tasks with llms, arXiv preprint arXiv:2303.01294 (2023).
[159] S. Yao, D. Zhao, J. Fu, et al., Tree of thoughts: Deliberate problem solving with large language models, arXiv preprint arXiv:2305.10601 (2023).
[160] K. Shuster, J. Ju, S. Roller, E. Dinan, J. Weston, Dialogue models with consistent personality, in: Findings of the Association for Computational Linguistics: EMNLP 2022, 2022, pp. 1057–1070.
[161] X. Wu, Y. Wang, W. Liu, Biomedical question answering with retrieval-augmented generation, Journal of Biomedical Informatics 137 (2023) 104232.
[162] S. Mukherjee, Q. Zhang, S. U. Sinha, N. Miller, A. Schein, D. Roth, Orca: Progressive learning from complex explanation traces of gpt-4, arXiv preprint arXiv:2306.02707 (2023).
[163] S. Fan, X. Cong, Y. Fu, Z. Zhang, S. Zhang, Y. Liu, Y. Wu, Y. Lin, Z. Liu, M. Sun, Workflowllm: Enhancing workflow orchestration capability of large language models, arXiv preprint arXiv:2411.05451 (2024).
[164] L. B. Kaesberg, T. Ruas, J. P. Wahle, B. Gipp, Citeassist: A system for automated preprint citation and bibtex generation, arXiv preprint arXiv:2407.03192 (2024).
[165] C. Jeong, A study on the implementation method of an agent-based advanced rag system using graph, arXiv preprint arXiv:2407.19994 (2024).
[166] V. Zolfaghari, N. Petrovic, F. Pan, K. Lebioda, A. Knoll, Adopting rag for llm-aided future vehicle design, in: 2024 2nd International Conference on Foundation and Large Language Models (FLLM), IEEE, 2024, pp. 437–442.
[167] W. Huang, P. Abbeel, D. Pathak, I. Mordatch, Language models as zero-shot planners: Extracting actionable knowledge for embodied agents, in: International conference on machine learning, PMLR, 2022, pp. 9118–9147.
[168] J. Tong, W. Guo, J. Shao, Q. Wu, Z. Li, Z. Lin, J. Zhang, Wirelessagent: Large language model agents for intelligent wireless networks, arXiv preprint arXiv:2505.01074 (2025).
[169] S. Kim, Y. Su, L.-C. Wang, Iea-plugin: An ai agent reasoner for test data analytics, arXiv preprint arXiv:2504.11496 (2025).
[170] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al., Chain-of-thought prompting elicits reasoning in large language models, Advances in neural information processing systems 35 (2022) 24824–24837.
[171] Langsmith, Get started with langsmith, https://docs.smith.

langchain.com/, 2023.
[172] Y. Huang, L. Sun, H. Wang, S. Wu, Q. Zhang, Y. Li, C. Gao, Y. Huang, W. Lyu, Y. Zhang, et al., Trustllm: Trustworthiness in large language models, arXiv preprint arXiv:2401.05561 (2024).
[173] J. Wei, X. Wang, et al., Chain-of-thought prompting elicits reasoning in large language models, arXiv preprint arXiv:2201.11903 (2022).
[174] G. Izacard, E. Grave, Leveraging passage retrieval with generative models for open domain question answering, arXiv preprint arXiv:2007.01282 (2020).
[175] W. Chen, P. Verga, M. De Jong, J. Wieting, W. Cohen, Augmenting pre-trained language models with qa-memory for open-domain question answering, arXiv preprint arXiv:2204.04581 (2022).
[176] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, A. Anandkumar, Voyager: An open-ended embodied agent with large language models, arXiv preprint arXiv:2305.16291 (2023).
[177] K. Stechly, M. Marquez, S. Kambhampati, Gpt-4 doesn't know it's wrong: An analysis of iterative prompting for reasoning problems, arXiv preprint arXiv:2310.12397 (2023).
[178] M. Chen, Y. Li, Y. Yang, S. Yu, B. Lin, X. He, Automanual: Constructing instruction manuals by llm agents via interactive environmental learning, Advances in Neural Information Processing Systems 37 (2024) 589–631.
[179] J. Liu, C. Yu, J. Gao, Y. Xie, Q. Liao, Y. Wu, Y. Wang, Llm-powered hierarchical language agent for real-time human-ai coordination, arXiv preprint arXiv:2312.15224 (2023).
[180] C. Varangot-Reille, C. Bouvard, A. Gourru, M. Ciancone, M. Schaeffer, F. Jacquenet, Doing more with less–implementing routing strategies in large language model-based systems: An extended survey, arXiv preprint arXiv:2502.00409 (2025).
[181] T. Wu, E. Jiang, A. Donsbach, J. Gray, A. Molina, M. Terry, C. J. Cai, Promptchainer: Chaining large language model prompts through visual programming, in: CHI Conference on Human Factors in Computing Systems Extended Abstracts, 2022, pp. 1–10.
[182] L. Reynolds, K. McDonell, Prompt programming for large language models: Beyond the few-shot paradigm, in: Extended abstracts of the 2021 CHI conference on human factors in computing systems, 2021, pp. 1–7.
[183] A. Bavaresco, R. Bernardi, L. Bertolazzi, D. Elliott, R. Fernández, A. Gatt, E. Ghaleb, M. Giulianelli, M. Hanna, A. Koller, et al., Llms instead of human judges? a large scale empirical study across 20 nlp evaluation tasks, arXiv preprint arXiv:2406.18403 (2024).
[184] Y. Abdullin, D. Molla-Aliod, B. Ofoghi, J. Yearwood, Q. Li, Synthetic dialogue dataset generation using llm agents, arXiv preprint arXiv:2401.17461 (2024).
[185] L. Yan, L. Sha, L. Zhao, Y. Li, R. Martinez-Maldonado, G. Chen, X. Li, Y. Jin, D. Gašević, Practical and ethical challenges of large language models in education: A systematic scoping review, British Journal of Educational Technology 55 (2024) 90–112.
[186] E. Jiang, K. Olson, E. Toh, A. Molina, A. Donsbach, M. Terry, C. J. Cai, Promptmaker: Prompt-based prototyping with large language models, in: CHI Conference on Human Factors in Computing Systems Extended Abstracts, 2022, pp. 1–8.
[187] Z. Ke, F. Jiao, Y. Ming, X.-P. Nguyen, A. Xu, D. X. Long, M. Li, C. Qin, P. Wang, S. Savarese, et al., A survey of frontiers in llm reasoning: Inference scaling, learning to reason, and agentic systems, arXiv preprint arXiv:2504.09037 (2025).
[188] J. Diaz-De-Arcaya, J. López-De-Armentia, R. Miñón, I. L. Ojanguren, A. I. Torre-Bastida, Large language model operations (llmops): Definition, challenges, and lifecycle management, in: 2024 9th International Conference on Smart and Sustainable Technologies (SpliTech), IEEE, 2024, pp. 1–4.
[189] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, Y. Zhuang, Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face, Advances in Neural Information Processing Systems 36 (2023) 38154–38180.
[190] Y. Zheng, Y. Chen, B. Qian, X. Shi, Y. Shu, J. Chen, A review on edge large language models: Design, execution, and applications, ACM Computing Surveys 57 (2025) 1–35.
[191] Y. Gao, Y. Xiong, Y. Zhong, Y. Bi, M. Xue, H. Wang, Synergizing rag and reasoning: A systematic review, arXiv preprint arXiv:2504.15909 (2025).
[192] M. Chau, J. Xu, B. Ampel, C. Yang, J. Hu, H. Chen, A. Yuan, E. G. Colato, B. Pescosolido, H. Song, et al., Management information systems, ACM Transactions on 16 (2025).
[193] Y. Bai, S. Lee, S.-H. Seo, A survey on directed acyclic graph-based blockchain in smart mobility, Sensors 25 (2025) 1108.
[194] M. J. Lee, W. P. Jeon, A study on the development of a system using langchain and an aspect-based sentiment analysis model: Focusing on the real estate domain, Journal of Information Technology Services 24 (2025) 41–59.
[195] Y. Zou, A. H. Cheng, A. Aldossary, J. Bai, S. X. Leong, J. A. Campos-Gonzalez-Angulo, C. Choi, C. T. Ser, G. Tom, A. Wang, et al., El agente: An autonomous agent for quantum chemistry, Matter 8 (2025).
[196] N. Shanmugapriya, P. Yugeshkaran, U. H. Charan, M. Jayasri, C. M. S. Christina, P. Anitha, K. Tamilselvi, Evaluating the performance of llms on sql and nosql database using langsmith, International Journal of Environmental Sciences (2025) 1528–1561.
[197] X. Tan, Y. Jiang, Y. Yang, H. Xu, Towards end-to-end optimization of llm-based applications with ayo, in: Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, 2025, pp. 1302–1316.
[198] S. Kweon, S. Nam, H. Lim, H. Hong, E. Choi, A large-scale real-world evaluation of llm-based virtual teaching assistant, arXiv preprint arXiv:2506.17363 (2025).
[199] I. Alshehri, A. Alshehri, A. Almalki, M. Bamardouf, A. Akbar, Breachseek: A multi-agent automated penetration tester, arXiv preprint arXiv:2409.03789 (2024).
[200] X. He, D. Wu, Y. Zhai, K. Sun, Sentinelagent: Graph-based anomaly detection in multi-agent systems, arXiv preprint arXiv:2505.24201 (2025).
[201] A. E. Hassan, D. Lin, G. K. Rajbahadur, K. Gallaba, F. R. Cogo, B. Chen, H. Zhang, K. Thangarajah, G. Oliva, J. Lin, et al., Rethinking software engineering in the era of foundation models: A curated catalogue of challenges in the development of trustworthy fmware, in: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, 2024, pp. 294–305.
[202] Y. Dong, R. Mu, Y. Zhang, S. Sun, T. Zhang, C. Wu, G. Jin, Y. Qi, J. Hu, J. Meng, et al., Safeguarding large language models: A survey, arXiv preprint arXiv:2406.02622 (2024).
[203] J. Wang, Z. Duan, Intelligent spark agents: A modular langgraph framework for scalable, visualized, and enhanced big data machine learning workflows, arXiv preprint arXiv:2412.01490 (2024).
[204] K. Petersen, J. M. Gerken, On the road to interactive llm-based systematic mapping studies, Information and Software Technology 178 (2025) 107611.
[205] H. Felzmann, E. Fosch-Villaronga, C. Lutz, A. Tamò-Larrieux, Towards transparency by design for artificial intelligence, Science and engineering ethics 26 (2020) 3333–3361.
[206] N. Balasubramaniam, M. Kauppinen, A. Rannisto, K. Hiekkanen, S. Kujala, Transparency and explainability of ai systems: From ethical guidelines to requirements, Information and Software Technology 159 (2023) 107197.
[207] R. Calegari, G. Ciatto, V. Mascardi, A. Omicini, Logic-based technologies for multi-agent systems: a systematic literature review, Autonomous Agents and Multi-Agent Systems 35 (2021) 1.
[208] J. Tang, T. Fan, C. Huang, Autoagent: A fully-automated and zero-code framework for llm agents, arXiv preprint arXiv:2502.05957 (2025).
[209] A. Gulli, L. Nigam, J. Wiesinger, V. Vuskovic, I. Sigler, I. Nardini, N. Stroppa, S. Kartakis, N. Saribekyan, A. Bount, Agents companion, Kaggle/Google, Tech. Rep., February (2025).
[210] H. He, J. Gray, A. Cangelosi, Q. Meng, T. M. McGinnity, J. Mehnen, The challenges and opportunities of human-centered ai for trustworthy robots and autonomous systems, IEEE Transactions on Cognitive and Developmental Systems 14 (2021) 1398–1412.
[211] B. W. Israelsen, N. R. Ahmed, "dave... i can assure you... that it's going to be all right..." a definition, case for, and survey of algorithmic assurances in human-autonomy trust relationships, ACM Computing Surveys (CSUR) 51 (2019) 1–37.
[212] C. Prunkl, Human autonomy at risk? an analysis of the challenges from ai, Minds and Machines 34 (2024) 26.
[213] W.-C. Chang, B. Esmaeili, S. Hasanzadeh, Impacts of physical and informational failures on worker–autonomy trust in future construction, Journal of Construction Engineering and Management 151 (2025) 04025011.

[214] S. Raza, R. Sapkota, M. Karkee, C. Emmanouilidis, Trism for agentic ai: A review of trust, risk, and security management in llm-based agentic multi-agent systems, arXiv preprint arXiv:2506.04133 (2025).

[215] S. Barua, Exploring autonomous agents through the lens of large language models: A review, arXiv preprint arXiv:2404.04442 (2024).

[216] A. M. Darwish, E. A. Rashed, G. Khoriba, Mitigating llm hallucinations using a multi-agent framework, Information 16 (2025) 517.

[217] A. Göldi, R. Rietsche, L. Ungar, Efficient management of llm-based coaching agents' reasoning while maintaining interaction quality and speed, in: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, 2025, pp. 1–18.

[218] A. Ray, Ai-driven pk/pd modeling: Generative ai, llms, and langchain for precision medicine, Compassionate AI 1 (2025) 48–50.

[219] U. M. Borghoff, P. Bottoni, R. Pareschi, An organizational theory for multi-agent interactions integrating human agents, llms, and specialized ai, Discover Computing 28 (2025) 1–35.

[220] G. F. Febrian, G. Figueredo, Kemenkeugpt: Leveraging a large language model on indonesia's government financial data and regulations to enhance decision making, arXiv preprint arXiv:2407.21459 (2024).

[221] R. Bohnsack, M. de Wet, Ai is the strategy: From agentic ai to autonomous business models onto strategy in the age of ai, arXiv preprint arXiv:2506.17339 (2025).

[222] C. Zhao, G. Liu, R. Zhang, Y. Liu, J. Wang, J. Kang, D. Niyato, Z. Li, Z. Han, S. Sun, et al., Edge general intelligence through world models and agentic ai: Fundamentals, solutions, and challenges, arXiv preprint arXiv:2508.09561 (2025).