

Beyond Prompting: Decoupling Cognition from Execution in LLM-based Agents through the ORCA Framework

Guillermo Fernandez

Independent Researcher – ORCA Project

gfernandf+as@gmail.com

<https://github.com/gfernandf/agent-skills>

Abstract

Recent advances in large language model (LLM) agents have largely relied on prompt-centric designs, where complex tasks are executed through monolithic, single-shot or loosely structured prompting strategies. While effective in some settings, this approach results in implicit and opaque forms of cognition that hinder composability, observability, reuse, and governance.

In this work, we argue for a principled separation between cognition and execution in LLM-based agents. We introduce **ORCA (Open Cognitive Runtime for Agents)**, a conceptual and architectural framework that models cognition as the composition of explicit semantic units, termed *capabilities*, orchestrated through declarative workflows, or *skills*, and executed via a decoupled runtime layer. This separation enables structured reasoning processes, explicit control over execution, and dynamic binding to heterogeneous implementations, including LLM-based and non-LLM tools.

We formalize the core primitives of the ORCA framework and describe its architectural realization, including a capability registry, a workflow execution engine, and a flexible binding mechanism. We further introduce a governance model that incorporates trust levels, validation gates, and execution traceability, enabling safer and more interpretable agent behavior.

Through comparative experiments, we analyze the trade-offs between prompt-based approaches and structured cognitive execution, showing that while ORCA introduces additional computational overhead, it significantly improves composability, transparency, and controllability. We discuss the implications of this trade-off and outline scenarios where structured cognitive runtimes provide clear advantages.

Overall, this work positions cognitive runtime layers as a promising abstraction for structuring and governing agent cognition, particularly in settings where control, transparency, and composability are required.

1 Introduction

In industrial operations, even highly experienced operators are not expected to rely solely on their intelligence to execute complex procedures. Consider the isolation of a piece of equipment in an oil and gas facility: although an operator may understand the underlying engineering, fluid dynamics, system topology, and safety principles, executing the task by reasoning from first principles in real time would be error-prone and unacceptable. Instead, operators follow well-defined procedures that encapsulate prior knowledge, validated practices, and safety constraints. Expertise is expressed not by reinventing the process, but by correctly selecting, adapting, and executing these procedures in context.

This distinction between intelligence and structured execution is fundamental in engineered systems, yet remains underexplored in current LLM-based agent designs. Despite rapid advances in

large language models (LLMs), many agentic systems continue to rely on prompt-centric approaches, where complex tasks are executed through monolithic or loosely structured prompting strategies. In such settings, the model is implicitly expected to reconstruct the necessary reasoning and execution steps within a single interaction, effectively re-deriving the procedure each time a task is performed, often leveraging techniques such as chain-of-thought prompting [1].

While this paradigm can yield plausible results, it introduces important limitations. First, cognition remains implicit and opaque, making it difficult to inspect, validate, or reuse intermediate reasoning steps. Second, the lack of explicit structure hinders composability, as complex behaviors cannot be easily decomposed into reusable components. Third, control over execution is limited, constraining the ability to enforce constraints, integrate heterogeneous tools, or guarantee consistent behavior across runs. These limitations become increasingly critical as LLM-based agents are deployed in real-world environments, where reliability, traceability, and governance are required.

Recent approaches have attempted to address some of these challenges through tool augmentation and structured interaction patterns, such as interleaving reasoning and acting [2] or enabling models to invoke external tools [3]. While these methods introduce elements of structure, they typically remain centered around prompt-driven execution, where the boundaries between reasoning, control flow, and tool invocation are blurred and largely implicit. As a result, they provide limited support for explicit composition, formal reasoning about execution, or systematic governance of agent behavior. Related efforts toward modular reasoning architectures have also been proposed [4], though they often stop short of defining a general cognitive runtime abstraction.

In this work, we argue that cognition in LLM-based agents should be explicitly structured rather than implicitly inferred. We introduce **ORCA (Open Cognitive Runtime for Agents)**, a conceptual and architectural framework that separates cognition from execution and models it as the composition of explicit semantic units, termed *capabilities*, orchestrated through declarative workflows, or *skills*, and executed via a decoupled runtime layer. This separation enables structured reasoning processes, explicit control over execution, and dynamic binding to heterogeneous implementations, including both LLM-based and non-LLM tools.

Within ORCA, capabilities define stable semantic interfaces for cognitive operations, independent of their underlying implementation. Skills represent explicit workflows that compose these capabilities into higher-level behaviors, while the runtime layer manages execution, state, and interaction with external systems. This design allows agents to leverage predefined cognitive structures where available, while retaining the flexibility to orchestrate behavior dynamically in contexts where no predefined workflows exist.

The contributions of this work are threefold. First, we propose a cognitive runtime abstraction that formalizes the separation between cognition and execution in LLM-based agents. Second, we define a set of core primitives — capabilities, skills, bindings, and execution state — that enable composable and governable agent behavior. Third, we present ORCA as a concrete realization of this paradigm and analyze its properties, including its implications for composability, transparency, and control.

Finally, we evaluate the proposed approach against prompt-based baselines, highlighting the trade-offs between structured cognitive execution and monolithic prompting. Our results show that while the introduction of explicit structure may incur additional computational overhead, it provides significant benefits in terms of interpretability, reuse, and governance.

2 Limitations of Prompt-based Agent Design

Despite their flexibility and strong empirical performance, prompt-based approaches to agent design exhibit structural limitations that become increasingly evident as task complexity and reliability requirements grow. These limitations stem from the implicit and monolithic nature of prompt-driven cognition, where reasoning, control flow, and execution are entangled within a single interaction or loosely coordinated sequence of interactions.

2.1 Implicit and Opaque Cognition

In prompt-based systems, the reasoning process is typically embedded within natural language generation, without an explicit representation of intermediate steps or decision structures. While techniques such as step-by-step prompting can encourage models to externalize parts of their reasoning [1], these representations remain informal and lack a well-defined structure. As a result, it is difficult to systematically inspect, validate, or manipulate intermediate cognitive states, limiting both interpretability and debuggability.

2.2 Lack of Composability and Reuse

Prompt-centric designs do not naturally support modular composition. Complex behaviors are often encoded within large prompts or prompt templates, making them difficult to decompose into reusable components. Even when multi-step workflows are implemented, they are typically constructed in an ad hoc manner, with limited standardization of interfaces or semantics between steps. This hinders reuse across tasks and reduces the ability to build higher-level abstractions on top of existing capabilities.

2.3 Entanglement of Reasoning and Control Flow

In many agentic systems, control flow — such as branching, iteration, or tool selection — is implicitly handled within the model’s generated output rather than through explicit execution logic, as seen in approaches that interleave reasoning and acting [2]. This entanglement places the burden of both reasoning and orchestration on the model, making behavior harder to predict and constrain. It also limits the ability to enforce deterministic execution patterns or integrate external control mechanisms in a principled way.

2.4 Limited Execution Control and Constraint Enforcement

Because execution is largely driven by model outputs, enforcing constraints — such as safety rules, business logic, or operational boundaries — becomes challenging. While guardrails and post-processing techniques can mitigate some risks, they are typically reactive and external to the core reasoning process. This limits the ability to ensure consistent and policy-compliant behavior, particularly in environments where strict guarantees are required.

2.5 Weak Support for Observability and Traceability

Prompt-based systems provide limited visibility into how decisions are made. Although logs of prompts and outputs can be recorded, they do not constitute a structured trace of execution. This makes it difficult to reconstruct the sequence of cognitive steps leading to a given outcome, hindering debugging, auditing, and performance analysis. As systems scale, the lack of explicit execution traces becomes a significant operational limitation.

2.6 Redundant Re-computation of Cognitive Processes

Finally, prompt-centric approaches often require the model to reconstruct similar reasoning patterns repeatedly across tasks. Even when prompts are reused, the underlying cognitive process is re-executed from scratch, rather than leveraging persistent or reusable structures. This leads to inefficiencies and variability in outcomes, particularly for tasks that could otherwise be captured as stable, reusable procedures.

Taken together, these limitations highlight a fundamental issue: prompt-based agent design conflates cognition with execution, relying on the model to implicitly manage both. While this approach offers flexibility, it lacks the structural properties required for building reliable, composable, and governable systems.

3 The ORCA Framework

We introduce **ORCA (Open Cognitive Runtime for Agents)** as a conceptual and architectural framework for structuring cognition in LLM-based agents. ORCA is built on the principle that cognition should be explicitly represented, composed, and executed, rather than implicitly inferred within prompt-driven interactions. To achieve this, ORCA separates *what* is computed (cognition) from *how* it is executed (runtime), enabling structured, composable, and governable agent behavior.

At its core, ORCA defines a set of primitives that together form a cognitive runtime model: **capabilities**, **skills**, **bindings**, and **execution state**.

3.1 Capabilities

A *capability* is a semantic unit of cognition that represents a well-defined operation over information. Capabilities define *what* is to be computed, independently of *how* the computation is implemented.

Formally, a capability can be understood as a typed interface:

$$c : X \rightarrow Y$$

where X and Y represent structured input and output spaces. Capabilities encapsulate operations such as transformation, evaluation, extraction, or decision-making, and are defined in terms of their semantics rather than their implementation.

In addition to input and output types, capabilities may expose *behavioral properties* that characterize their execution. These may include whether a capability is deterministic, whether it produces side effects, or whether it is idempotent under repeated execution. While not all capabilities satisfy these properties — particularly those backed by probabilistic models — they provide a useful framework for reasoning about reliability, reproducibility, and system-level guarantees.

Crucially, capabilities are **implementation-agnostic**. The same capability may be realized through different underlying mechanisms, including LLM prompting, deterministic algorithms, external APIs, or hybrid approaches. This abstraction enables portability and interchangeability of implementations without altering the cognitive structure of the system.

3.2 Skills

A *skill* is a declarative workflow that composes multiple capabilities into a higher-level cognitive process. Skills define *how* individual cognitive operations are orchestrated to achieve a specific objective.

A skill can be modeled as a directed computation graph:

$$S = (C, E)$$

where C is a set of capabilities and E defines the dataflow and execution dependencies between them.

Unlike prompt-based workflows, where control flow is implicit, skills make the structure of cognition explicit. They define the sequence of operations, the flow of intermediate data, and the conditions under which different paths are executed. This explicit structure enables reuse, composability, and systematic reasoning about behavior.

3.3 Bindings

A *binding* defines how a capability is mapped to a concrete implementation at execution time. Bindings provide the link between abstract cognitive operations and executable systems.

Given a capability c , a binding selects an implementation:

$$b(c) \rightarrow \text{implementation}$$

Bindings can target heterogeneous execution backends, including LLM-based implementations, external tools and APIs, deterministic functions, or local programmatic logic.

Importantly, bindings can be resolved dynamically, allowing the same cognitive structure to be executed in different environments or with different performance, cost, or trust characteristics.

3.4 Execution State

The *execution state* represents the structured memory of a running skill. It captures inputs, intermediate results, and outputs in a form that is explicitly addressable and manipulable.

Rather than relying on unstructured conversational context, ORCA maintains a structured state composed of named elements, enabling explicit dataflow between capabilities, inspection of intermediate results, and controlled propagation of information. This structured state forms the backbone of observability and traceability within the system.

3.5 Decoupling Cognition from Execution

The combination of capabilities, skills, bindings, and execution state enables a clear separation between cognition and execution. Cognitive structure is defined independently of the mechanisms used to realize it, allowing reasoning processes to be specified declaratively and executed across different backends.

This decoupling introduces several key properties: **composability**, through reusable capabilities and skills; **portability**, via interchangeable bindings; **observability**, through explicit execution state; and **control**, by separating orchestration from underlying implementations.

In contrast to prompt-based approaches, where these concerns are intertwined within model outputs, ORCA provides a structured layer in which they can be independently defined and managed.

At the same time, making cognition explicit introduces new design considerations. In particular, the definition of capabilities and skills raises questions around *semantic constraints*, such as type compatibility, preconditions and postconditions, and invariants that must be preserved across execution steps. While ORCA does not require a fully formal specification of these constraints, it provides a natural foundation for incorporating them, enabling more robust validation, safer composition, and stronger guarantees in agent behavior.

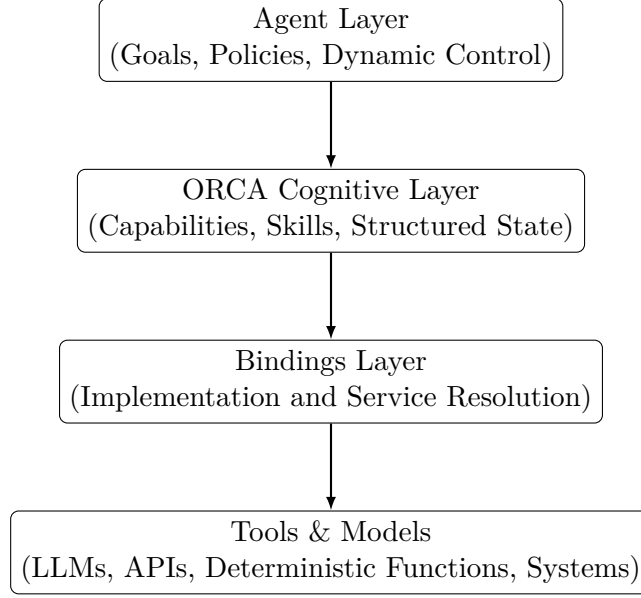


Figure 1: Layered architecture of ORCA within an agent system.

4 Design Principles of ORCA

The ORCA framework is grounded on a set of architectural principles that define how cognition should be represented and executed in structured agent systems. These principles distinguish ORCA from prompt-centric approaches by treating cognition as an explicit, composable layer that is separate from both high-level agent control and low-level execution mechanisms.

ORCA can therefore be understood as a distinct cognitive execution layer within an agent system, positioned between agent-level decision-making and tool-level execution. As illustrated in Figure 1, agents remain responsible for goals, policies, and orchestration, while ORCA manages the structured execution of cognitive processes through capabilities, skills, and explicit state.

These principles collectively define ORCA as a cognitive layer that is distinct from both agent-level decision-making and low-level execution mechanisms, while remaining interoperable with both.

4.1 Separation of Cognition from Execution

Cognitive intent should be defined independently from its execution mechanism. In ORCA, capabilities describe what cognitive function is required, while bindings determine how that function is realized at runtime. This separation enables portability across providers, substitution of implementations without rewriting workflows, and clearer reasoning about system behavior.

4.2 Declarative Cognition

Cognitive processes should be represented declaratively rather than procedurally. Instead of embedding cognition in prompts or imperative control logic, ORCA expresses cognitive operations as explicit semantic units and declarative workflows. This mirrors the advantages of declarative abstractions in other domains, where intent is specified independently of implementation details.

4.3 Late Binding of Intelligence

The resolution of cognitive capabilities should occur at runtime rather than design time. ORCA defers the selection of models, services, or execution strategies until execution, allowing the same skill to be adapted to context, policy, availability, cost, or trust requirements without changing its cognitive structure.

4.4 Minimal Cognitive Granularity

Cognitive operations should be decomposed into the smallest meaningful capabilities, as long as such decomposition improves control, reuse, or observability. This principle encourages representing cognition through atomic reusable units rather than monolithic prompts, while avoiding decomposition for its own sake.

4.5 Limits of Cognitive Decomposition

Fine-grained decomposition is not always beneficial. In practice, decomposition should be avoided when it does not improve traceability, validation, or execution control; when the operation is inherently atomic and does not justify further structure; when strong semantic coupling makes intermediate representations unstable; or when orchestration overhead outweighs the benefits of modularity. These constraints ensure that decomposition remains pragmatic rather than doctrinal.

4.6 Explicit Cognitive State

Intermediate cognitive artifacts should be explicitly represented and persisted across execution steps. ORCA uses structured execution state to expose inputs, intermediate results, and outputs as first-class elements. This enables traceability, debugging, and controlled propagation of information beyond what is possible in stateless prompt execution.

4.7 Observability and Control

Cognitive processes should be inspectable, traceable, and governable. By externalizing workflows and capability invocations, ORCA enables intermediate inspection, execution tracing, validation, and policy enforcement. Observability is therefore not an afterthought but a consequence of the execution model itself.

4.8 Agent Responsibilities

ORCA defines a cognitive runtime layer, but it does not replace the role of the agent. Agents remain responsible for high-level functions such as goal management, dynamic orchestration, policy selection, failure handling, and longer-term adaptation. This layered separation clarifies the role of ORCA within the broader agent architecture, ensuring that cognition, control, and execution remain modular and independently evolvable (Figure 1).

5 Architectural Instantiation of the ORCA Framework

The ORCA framework defines a set of conceptual primitives for structuring cognition. In order to be operationalized, these primitives must be instantiated within an architectural model that supports their definition, composition, and execution. This section outlines a reference architectural

instantiation of ORCA, describing the core system components required to realize a cognitive runtime layer while preserving the abstraction boundaries introduced in Section 3.

At a high level, an ORCA-compliant system is organized into four main components: a **capability registry**, a **skill definition layer**, a **binding and resolution mechanism**, and a **runtime execution engine**. Together, these components enable the definition, discovery, and execution of structured cognitive processes.

5.1 Capability Registry

The capability registry serves as the canonical source of truth for the semantic definition of capabilities. Each capability is defined in terms of its interface, including input and output types, as well as optional behavioral properties such as determinism, side effects, or idempotency.

Importantly, the registry captures the *semantics* of capabilities rather than their implementation. This separation allows capabilities to remain stable even as underlying implementations evolve. It also enables multiple implementations to coexist for a given capability, supporting portability across environments and execution contexts.

The registry may additionally support metadata associated with capabilities, such as descriptions, usage constraints, or compatibility requirements, providing a foundation for validation and discoverability.

5.2 Skill Definition Layer

The skill definition layer provides a declarative representation of workflows that compose capabilities into higher-level behaviors. Skills define the structure of cognition through explicit dataflow and execution dependencies, enabling complex processes to be expressed as compositions of simpler operations.

Within this layer, skills are specified independently of their execution environment. They describe the sequence and dependency of operations, the mapping of inputs and outputs across steps, and, optionally, conditional or branching logic.

This declarative approach allows skills to be versioned, reused, and analyzed independently of runtime concerns, facilitating modular development and systematic composition.

5.3 Binding and Resolution Mechanism

The binding layer is responsible for mapping abstract capabilities to concrete implementations at execution time. This mapping may be defined statically or resolved dynamically based on the execution context, available resources, or policy constraints.

A binding associates a capability with a specific execution strategy, which may include invoking an LLM with a particular prompt template, calling an external API, executing a deterministic function, or combining multiple mechanisms.

The resolution process may incorporate selection policies that consider factors such as performance, cost, reliability, or trust level. This enables the same cognitive structure to be executed differently depending on the operational context, without modifying the skill definitions.

5.4 Runtime Execution Engine

The runtime execution engine is responsible for executing skills by orchestrating the invocation of capabilities according to their defined dependencies. It manages the execution state, evaluates dataflow expressions, and coordinates interactions with bound implementations.

Execution proceeds as a sequence of steps, each corresponding to a capability invocation. At each step, the runtime resolves the appropriate binding, retrieves the required inputs from the execution state, invokes the underlying implementation, and updates the state with the resulting outputs.

This explicit execution model enables fine-grained control over the process, including the ability to monitor progress, inspect intermediate results, and enforce execution policies.

5.5 Execution State and Dataflow

The execution state provides a structured representation of all data involved in the execution of a skill, including inputs, intermediate results, and outputs. Rather than relying on unstructured context accumulation, the state is explicitly organized into named elements that can be referenced and manipulated across steps.

Dataflow between capabilities is defined through mappings that specify how outputs from one step are propagated to subsequent steps. This explicit representation enables validation of dependencies, detection of inconsistencies, and controlled propagation of information. The combination of structured state and explicit dataflow forms the basis for traceability and observability within the system.

6 Reference Implementation: `orca-agent-skills`

To demonstrate the practical viability of the ORCA framework, we implemented `orca-agent-skills` as a reference cognitive runtime system. The reference implementation is publicly available at <https://github.com/gfernandf/agent-skills>. Rather than re-describing the architectural model, this section focuses on the concrete design decisions required to operationalize ORCA abstractions in an executable environment.

6.1 Overview

To ground the ORCA framework in a concrete system, we implemented a full runtime environment that operationalizes its core abstractions. Rather than acting as a thin orchestration layer, the implementation realizes ORCA as a structured execution runtime, where cognition, execution, and integration are explicitly separated.

Figure 2 presents the architecture of this implementation.

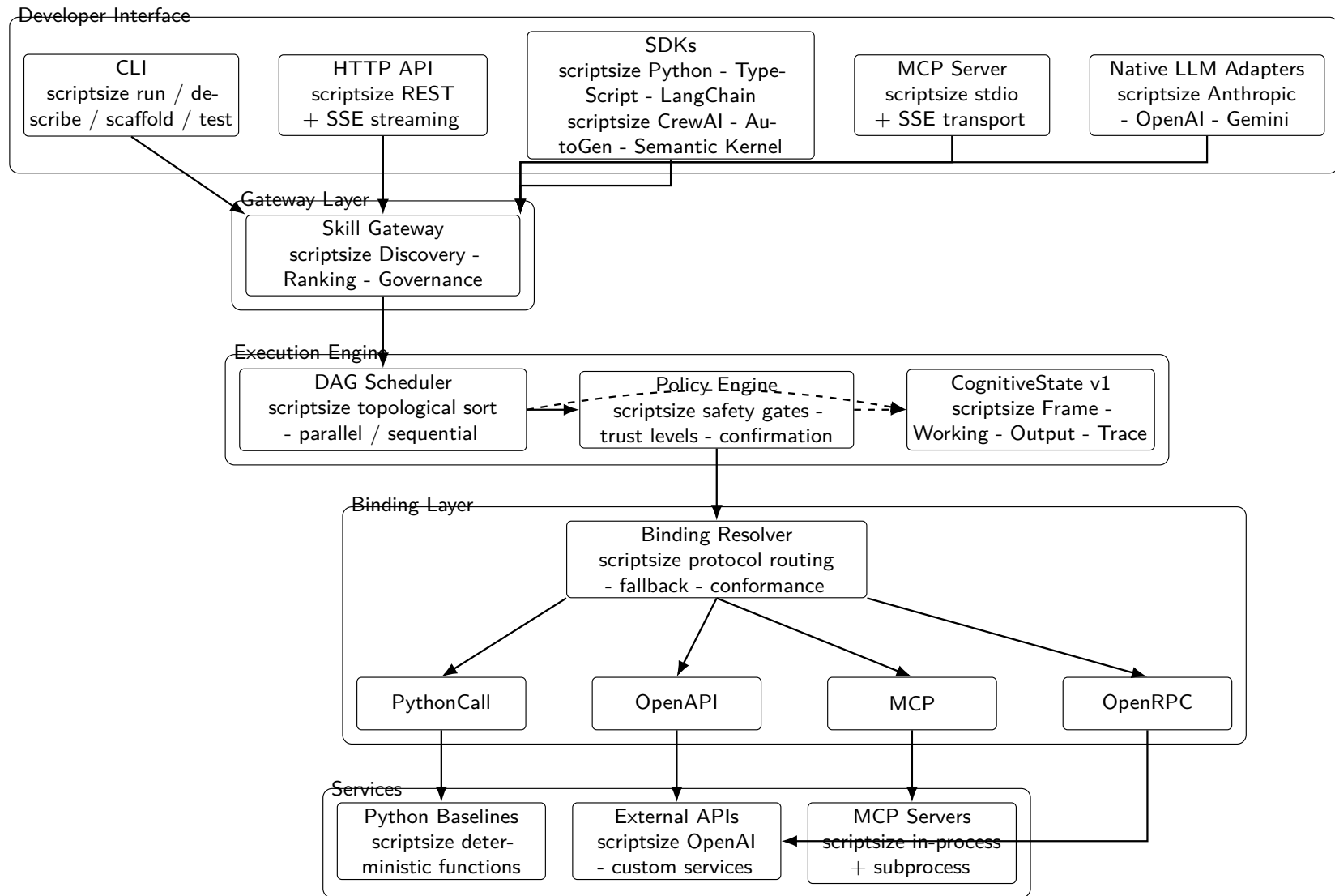


Figure 2: ORCA Runtime Architecture. The ORCA implementation is structured as a layered runtime system that decouples interfaces, orchestration, execution, and service bindings. Skills are resolved through a gateway layer and executed via a DAG-based scheduler, with explicit cognitive state tracking and policy enforcement. Capabilities are dynamically bound to heterogeneous execution backends, including local deterministic functions, external APIs, and MCP-based services.

The system is organized into four primary layers: developer interfaces, a gateway layer, an execution engine, and a binding layer backed by heterogeneous services. This layered separation enforces a strict decoupling between cognitive intent and execution mechanisms.

6.2 Declarative Representation of Cognition

In **orca-agent-skills**, both capabilities and skills are defined using structured, declarative specifications. Skills are expressed as explicit workflows in which each step corresponds to a capability invocation, and data dependencies between steps are defined through named mappings.

This representation makes cognitive structure directly inspectable and executable. Instead of encoding workflows within prompts, the system externalizes them into a formal structure that can be validated, versioned, and reused.

A key design decision is the explicit separation between **inputs**, provided at invocation time, **intermediate variables**, produced and consumed across steps, and **outputs**, representing final results. This structured representation enables deterministic dataflow independent of the underlying implementations of individual capabilities.

6.3 Dataflow and Dependency Resolution

Execution in **orca-agent-skills** is driven by explicit data dependencies between steps. Each step declares its required inputs and produced outputs, allowing the runtime to construct an execution order based on dependency resolution rather than fixed sequencing.

This approach enables automatic ordering of steps, validation of missing or incompatible inputs, and detection of unused or redundant outputs. By treating workflows as dependency graphs rather than linear scripts, the system provides a more robust foundation for composing complex cognitive processes.

6.4 Expression Evaluation and State Access

To support flexible dataflow, the implementation includes an expression evaluation mechanism that allows steps to reference elements of the execution state. These expressions are evaluated in a controlled environment, avoiding unrestricted code execution while enabling dynamic data transformations.

This mechanism allows mapping outputs from one step into inputs of another, conditional parameterization of capability invocations, and lightweight transformation of intermediate results. The use of a constrained expression system balances flexibility with safety, ensuring that workflows remain declarative while still supporting non-trivial data manipulation.

6.5 Capability Model and Naming

Capabilities are the fundamental units of execution in ORCA. In the implementation, capabilities follow a structured naming convention of the form **domain.noun.verb**. This convention serves as more than a syntactic choice: it provides a lightweight semantic taxonomy that improves discoverability, semantic grouping, and predictable reuse across skills.

Capabilities may declare additional properties such as determinism, side effects, and idempotency, which influence execution and governance behavior. This suggests a pragmatic path toward richer semantic relationships between capabilities without requiring a heavy formal ontology.

6.6 Binding Resolution and Overrides

Bindings in `orca-agent-skills` are resolved through a layered mechanism that separates default implementations from environment-specific overrides. Each capability can have one or more associated implementations, and the runtime selects among them based on configuration or execution context.

This design enables substitution of implementations without modifying skills, environment-specific optimization, and experimentation with alternative strategies for the same capability. The ability to override bindings is particularly important for adapting the system across development, testing, and production environments.

6.7 Execution Engine Semantics

The execution engine follows a step-wise evaluation model in which each step resolves its required inputs from the execution state, selects the appropriate binding, invokes the underlying implementation, and updates the state with the resulting outputs.

This explicit execution loop provides fine-grained control over the process and enables the system to monitor execution progress, capture intermediate results, and enforce execution policies. Unlike prompt-based approaches, where execution is implicit within model outputs, this model exposes each cognitive operation as a distinct and observable unit.

6.8 Structured State and Execution Tracing

A key feature of the implementation is the use of a structured execution state that captures all relevant data throughout the execution of a skill. This state is explicitly organized and accessible, enabling inspection of intermediate results and reconstruction of execution paths.

In addition, the system generates execution traces that record the sequence of steps executed, the inputs and outputs of each step, and the selected bindings. These traces provide a detailed view of the cognitive process, enabling debugging, auditing, and performance analysis. They also form a basis for future extensions such as optimization, replay, or verification.

7 Emergent Governance in Structured Cognitive Systems

Structured representations of cognition enable forms of governance that are difficult or impractical to achieve in prompt-based systems. When reasoning and execution are made explicit — through well-defined units, workflows, and state — systems gain the ability to observe, constrain, and reason about their own behavior. This section examines how such governance capabilities emerge naturally from the architectural principles introduced in ORCA.

7.1 Observability and Traceability

In prompt-based systems, cognition is largely embedded within unstructured model outputs, limiting visibility into how decisions are produced. By contrast, structured cognitive systems expose each step of the reasoning process as an explicit operation.

Within ORCA, the execution of a skill produces a traceable sequence of capability invocations, along with their corresponding inputs, outputs, and execution context. This enables inspection of intermediate states, reconstruction of execution paths, and systematic debugging of unexpected outcomes. Such traceability transforms cognition from an opaque process into an observable system, providing a foundation for both analysis and accountability.

7.2 Explicit Control and Constraint Enforcement

When control flow is implicit within model outputs, enforcing constraints requires external intervention, often in the form of post-processing or heuristic guardrails. In structured cognitive systems, control is embedded within the execution model itself.

By externalizing workflows and execution logic, ORCA enables constraints to be applied at multiple levels: at the capability level, through input/output validation; at the workflow level, through explicit control structures; and at the execution level, through policy-driven orchestration. This allows constraints to be enforced proactively rather than reactively, improving consistency and reducing the risk of unintended behavior.

7.3 Trust and Execution Semantics

The explicit representation of cognitive units also enables differentiation between types of execution. Capabilities may exhibit different behavioral properties — such as determinism, side effects, or probabilistic variability — which have implications for how their outputs should be interpreted and trusted.

By making these properties explicit, structured cognitive systems can incorporate notions of *trust* into execution decisions. For example, workflows may treat deterministic operations differently from probabilistic ones, or apply additional validation to outputs generated by non-deterministic components. This provides a basis for reasoning not only about *what* is computed, but also about *how reliable* those computations are.

7.4 Validation and Semantic Constraints

The decomposition of cognition into explicit units enables validation mechanisms that are difficult to implement in prompt-based systems. Because inputs, outputs, and intermediate states are structured, it becomes possible to define and enforce constraints such as type compatibility, preconditions and postconditions, and invariants across execution steps.

While ORCA does not require full formal specification of such constraints, it provides a natural framework for incorporating them. This opens the possibility of systematically verifying aspects of agent behavior, rather than relying solely on empirical evaluation.

7.5 Implications for Reliable Agent Systems

Taken together, these properties suggest that governance is not an external addition to agent systems, but an emergent consequence of how cognition is represented and executed. By structuring cognition explicitly, systems gain the ability to observe, control, and validate their behavior in ways that are not readily achievable through prompt-centric designs.

This has important implications for the deployment of LLM-based agents in real-world environments. As applications move beyond exploratory use cases toward operational settings, requirements such as reliability, traceability, and policy compliance become central. Structured cognitive runtimes provide a pathway to meeting these requirements without constraining the expressive power of underlying models.

8 Comparison with Existing Agent Frameworks

The value of ORCA is best understood not as a replacement for agent systems, but as a different abstraction boundary for building them. Existing frameworks often provide orchestration utilities,

prompt templates, or tool invocation patterns, but they typically leave cognition coupled to execution. ORCA instead introduces an explicit cognitive layer in which capabilities, workflows, and state are first-class runtime concepts.

Feature	ORCA	LangChain-style	ReAct-style
Declarative workflows	Yes	Partial	No
Late binding of implementations	Yes	Partial	No
Explicit cognitive state	Yes	Limited	No
Reusable cognitive capabilities	Yes	Partial	No
Step-level observability	Yes	Partial	Limited

Table 1: Structural comparison between ORCA and representative agent design styles. The comparison is architectural rather than exhaustive, and highlights differences in abstraction rather than implementation completeness.

Frameworks such as LangChain provide useful orchestration utilities and ecosystem integrations, while ReAct-style approaches tightly couple reasoning and acting within prompt-driven loops. ORCA differs in that it externalizes cognition itself into explicit semantic units and runtime-managed workflows. This makes composability, state, and governance architectural concerns rather than prompt-engineering artifacts.

9 Experimental Evaluation

9.1 Objective

The objective of this evaluation is to compare two execution paradigms for LLM-based tasks:

1. **Prompt-based execution**, where a single prompt is used to solve the entire task in one model invocation.
2. **Structured execution under ORCA**, where tasks are decomposed into reusable capabilities orchestrated through a declarative skill and executed via the ORCA runtime.

Rather than optimizing for raw task performance, this evaluation aims to characterize the *system-level trade-offs* introduced by structured cognitive execution.

9.2 Tasks

Two representative task types were selected.

Structured Decision-Making. A constrained reasoning task in which the model must select the best option among several alternatives given explicit evaluation criteria. The ORCA skill is `experiment.structured-decision`, composed of the capabilities `agent.option.generate` \rightarrow `agent.flow.branch`.

Multi-step Text Processing. A compositional task involving sequential transformations over text: entity extraction, summarization, and classification. The ORCA skill is `extttexperiment.text-processing-pipeline`, composed of the capabilities `exttttext.entity.extract` \rightarrow `text.content.summarize` \rightarrow `text.content.classify`.

9.3 Metrics

The evaluation focuses on system-level properties that are directly influenced by the execution model:

- **Latency:** total wall-clock execution time
- **Traceability:** whether intermediate reasoning steps are externally observable
- **Reusability:** whether components can be reused independently across tasks
- **Variability:** Jaccard distance across repeated executions on identical inputs
- **Output validity:** qualitative assessment of whether outputs are structurally valid and usable

We do not include a direct measure of correctness or semantic optimality. In open-ended reasoning and decision-making tasks, defining a consistent and objective notion of correctness requires external ground truth or human evaluation, which is outside the scope of this study.

9.4 Experimental Setup

The benchmark uses 10 inputs per task and two execution strategies per task. Variability is measured on 3 inputs with 3 repetitions each. All experiments were executed locally on a laptop without distributed infrastructure.

The evaluation spans two task families: a structured decision task and a multi-step text-processing task. For each family, we compare a single-shot prompt-based baseline against an ORCA skill that decomposes the task into explicit capability invocations. The model used throughout the benchmark is GPT-4o-mini with a fixed random seed of 42, so the comparison isolates differences introduced by the execution model rather than by model choice.

This setup does not aim to establish state-of-the-art task quality. Instead, it is intended to characterize the operational trade-offs introduced by explicit cognitive structure: latency overhead, reusability, traceability, and the effect of multi-step composition on output variability.

9.5 Results

Dimension	Prompt-based	ORCA Structured
Avg Latency (decision)	4.79s	12.17s
Avg Latency (text processing)	2.93s	7.86s
Traceability	No	Yes (step-level)
Reusability	No	Yes (capability-level)
Variability (text processing)	~0.12	~0.17
Output Validity	High	High

Table 2: Comparison between prompt-based and ORCA structured execution.

9.6 Analysis

The results highlight a clear distinction between prompt-based execution and structured execution under ORCA.

Prompt-based approaches achieve lower latency across both tasks, as the entire problem is resolved within a single model invocation. In contrast, ORCA introduces a consistent overhead due to multi-step decomposition and orchestration, resulting in execution times approximately $2\text{--}3\times$ higher.

However, ORCA enables capabilities that are not present in prompt-based systems. In particular, it provides explicit *traceability*, exposing intermediate steps of the reasoning process, and *reusability*, allowing individual capabilities to be composed across different workflows. These properties emerge directly from the structured execution model.

Both approaches consistently produce valid and usable outputs across tasks. While this evaluation does not attempt to measure semantic optimality, it confirms that structured decomposition does not prevent successful task completion.

Variability analysis shows a modest increase under ORCA, suggesting that multi-step composition may amplify stochastic effects across execution stages. This indicates a trade-off between structural transparency and execution stability.

9.7 Key Findings

The experimental results support the following conclusions:

- **Prompt-based execution optimizes for efficiency:** lower latency and minimal coordination overhead
- **ORCA enables system-level capabilities:** traceability and reusability emerge naturally from structured execution
- **Decomposition incurs predictable overhead:** latency increases due to orchestration across capabilities
- **Structured execution affects stability:** variability may increase due to multi-step composition
- **Task completion remains robust:** both approaches consistently produce valid outputs

Overall, ORCA should not be understood as a performance optimization technique, but as a framework that enables a different class of cognitive systems, prioritizing structure, inspectability, and composability over single-pass efficiency.

10 Discussion and Trade-offs

The ORCA framework introduces a structured approach to cognitive execution, emphasizing explicit decomposition, composability, and control. While this approach enables capabilities that are difficult to achieve in prompt-based systems, it also introduces trade-offs that must be carefully considered.

10.1 Efficiency vs. Control

Prompt-based approaches benefit from minimal overhead, as tasks are executed through a single model invocation. This often results in lower latency and reduced computational cost. In contrast, ORCA decomposes tasks into multiple capability invocations, each potentially requiring separate execution.

This decomposition introduces additional overhead in both latency and resource usage. However, it enables explicit control over execution, allowing intermediate steps to be inspected, validated,

and reused. The trade-off is therefore not between efficiency and inefficiency, but between *implicit efficiency* and *explicit control*. In scenarios where transparency, auditability, or reliability are critical, this trade-off may favor structured execution despite the associated cost.

10.2 Flexibility vs. Structure

Prompt-based systems offer a high degree of flexibility, as arbitrary tasks can be expressed directly through natural language. Structured cognitive systems, by contrast, require tasks to be expressed in terms of capabilities and workflows.

This introduces an upfront cost in defining capabilities and composing skills. However, once defined, these structures enable reuse, consistency, and composability across tasks.

Rather than restricting flexibility, ORCA shifts it from *unstructured prompt design* to *structured composition of cognitive units*. This shift may initially increase friction but provides a more stable foundation for building complex systems.

10.3 Reuse vs. Upfront Investment

A key advantage of structured cognitive systems is the ability to reuse previously defined capabilities and skills. However, this benefit is only realized after an initial investment in defining these components.

In prompt-based systems, each task is typically solved independently, with limited reuse beyond prompt templates. In contrast, ORCA enables the accumulation of reusable cognitive structures over time.

This suggests a temporal trade-off: prompt-based approaches may be more efficient for isolated or exploratory tasks, while structured approaches become advantageous as systems evolve and reuse becomes more prominent.

10.4 Determinism, Variability, and Trust

Prompt-based execution often exhibits variability due to the probabilistic nature of language models. While this variability can be mitigated through prompt engineering or sampling controls, it remains an inherent characteristic of the approach.

By decomposing tasks into explicit units, ORCA allows variability to be localized within specific capabilities. This enables more fine-grained reasoning about where uncertainty arises and how it should be handled.

In addition, the explicit representation of capabilities provides a basis for associating execution semantics — such as determinism, side effects, or reliability — with specific components. This supports more nuanced trust models, where different parts of a workflow may be treated differently depending on their characteristics.

10.5 Human and Agent Roles in Workflow Formation

In practice, structured cognitive systems introduce new questions regarding the role of humans and agents in defining workflows. While ORCA enables explicit definition of skills, it does not require that these workflows be manually authored in all cases.

A practical pattern that emerges is the coexistence of two complementary modes of operation. In one mode, agents leverage existing skills as reusable procedures, enabling efficient and reliable execution. In another mode, agents dynamically compose capabilities to address novel tasks, effectively synthesizing new workflows that may later be reused.

This dual mode reflects patterns observed in human problem-solving, where known procedures are reused when available and new procedures are constructed when necessary. ORCA provides a framework in which both modes can coexist, allowing systems to balance efficiency, flexibility, and control.

10.6 When Structured Cognitive Runtimes Are Appropriate

The benefits of structured cognitive execution are not uniform across all use cases. In exploratory or low-stakes scenarios, the overhead of defining and executing structured workflows may not be justified. In such cases, prompt-based approaches remain a practical and efficient solution.

However, in environments where reliability, traceability, and governance are important — such as enterprise workflows, regulated domains, or safety-critical systems — the advantages of structured execution become more pronounced.

This suggests that structured cognitive runtimes are not a universal replacement for prompt-based systems, but rather a complementary approach that is particularly suited to contexts where control and accountability are required. This distinction aligns with emerging perspectives that emphasize structured reasoning and tool integration over purely prompt-driven intelligence [5–8].

10.7 Limitations and Future Directions

The ORCA framework, as presented, focuses on structuring cognition and enabling explicit execution. However, several challenges remain.

First, the overhead associated with multi-step execution may limit scalability in latency-sensitive applications. Second, the design of capabilities and skills introduces new complexity, particularly in ensuring consistent interfaces and avoiding fragmentation. Third, while structured execution enables governance, it does not eliminate the underlying uncertainty of probabilistic components.

Future work may explore optimization strategies for reducing execution overhead, methods for automated skill generation and refinement, and mechanisms for integrating learning into structured cognitive systems.

11 Conclusion

This paper presented ORCA, a framework for decoupling cognition from execution in LLM-based agents. By representing cognition through explicit capabilities, declarative skills, bindings, and structured execution state, ORCA provides a foundation for building agent systems that are more composable, inspectable, and governable than prompt-centric alternatives.

Our analysis and experimental results suggest that structured cognitive execution does not primarily optimize for raw task efficiency. Instead, it enables system-level properties — such as traceability, reusability, and policy-aware execution — that are increasingly important as LLM-based agents move from exploratory prototypes toward operational systems.

We therefore view cognitive runtime layers not as a replacement for advancing model capabilities, but as a complementary systems abstraction for contexts in which control, transparency, and composability are required.

References

- [1] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.
- [2] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2023.
- [3] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.
- [4] AI21 Labs. MRKL Systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. Technical report, 2022.
- [5] Maciej Besta, Nils Blach, Lukas Gianinazzi, Joanna Gajda, Jan Kubicek, Robert Gerstenberger, Mariusz Nyczyk, and Torsten Hoeffler. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687*, 2023.
- [6] Denny Zhou, Nathanael Sch"arli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.
- [7] LangChain. LangChain. <https://www.langchain.com>, 2023.
- [8] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Chi Wang, and others. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.