



WHITE PAPER

A PRACTICAL APPROACH TO OPTIMIZE MULTI-AGENT SYSTEMS

AUTHORS

Ankur Kumar
Predli AB
ankur@predli.com

Edvin Augustinsson
Chalmers Tekniska
ketabati@student.chalmers.se

Marcus Zethraeus
Predli AB
mz@predli.com

Pradhan Sarathi
Predli AB
pradhan@predli.com

Robert Bridges
AI Sweden
robert.bridges@ai.se

16th December 2025

Contents

1	Introduction	2
2	Multi-Agent Systems	3
2.1	Conceptual Overview	3
2.2	Need for Multi-Agent Systems	3
3	System Architecture & Design Methodology	4
3.1	Design Choices & Rationale	4
3.2	Opinionated Configuration	5
3.3	System Limitations	5
4	Evaluating the System	7
4.1	Overall framework	7
4.1.1	Execution Completeness	7
4.1.2	Planning Capacity	8
4.1.3	Domain Capacity	9
4.1.4	Coordination Capacity	10
4.2	Datasets	10
4.3	Environments	11
4.4	Outcomes & Limitations	12
5	Optimizing the System	13
5.1	Optimization Approach	13
5.2	Search Space Complexity	13
5.3	Optimization Techniques	14
5.4	Bayesian Optimization with Gaussian Processes	15
5.5	GP-UCB Convergence Guarantees	17
5.5.1	Types of Regret	17
5.5.2	Srinivas et al. [SKKS10] Convergence Theorem	19
5.5.3	UCB Convergence Takeaways	20
5.6	BOGP Kernel Functions	20
6	Future work & Conclusions	20

1 Introduction

Multi-agent systems, composed of autonomous AI agents working collaboratively under a hierarchical framework, offer a promising approach to tackling complex, real-world tasks. By leveraging specialization, modularity, and scalable coordination, these systems can achieve outcomes that surpass the capabilities of single agents. However, designing, optimizing, and evaluating such systems present significant challenges, including reliance on heuristics, lack of standardized evaluation, and the inherent complexity of coordinating multiple agents. This whitepaper addresses these challenges by introducing a practical framework for structured optimization and comprehensive benchmarking, grounded in the principles of controlled autonomy and opinionated configuration. Our aim is to provide a clear pathway for enhancing the performance and reliability of multi-agent systems in demanding environments.

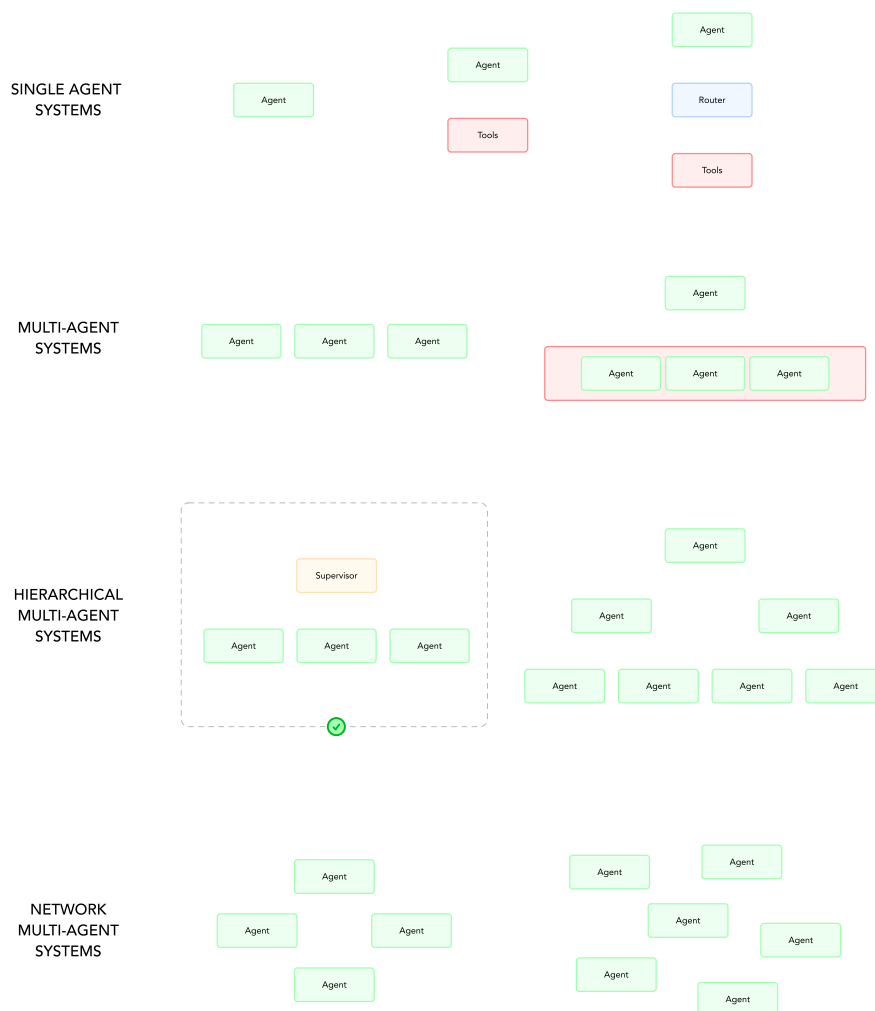


Figure 1: Examples of agentic systems. For the scope of this whitepaper, we will be focusing on hierarchical systems.

2 Multi-Agent Systems

2.1 Conceptual Overview

In its broadest definition, an AI agent can be understood as a Large Language Model (LLM) equipped with specific tools and guided by a predefined set of instructions [WMV24]. Unlike traditional software components that rely heavily on direct user control, agents are designed to operate autonomously. They can analyze their environment, reason about what actions are required next, and execute those actions without continuous human supervision. Their ability to make decisions in pursuit of a goal makes them flexible and adaptable in dynamic contexts.

When multiple such autonomous agents are connected in a structured way, allowing them to communicate, coordinate, and share knowledge, they form what is known as a multi-agent system. In this configuration, each agent contributes its specialized capabilities and their collaboration enables outcomes that are often more sophisticated, efficient, and scalable than what a single agent could achieve alone.

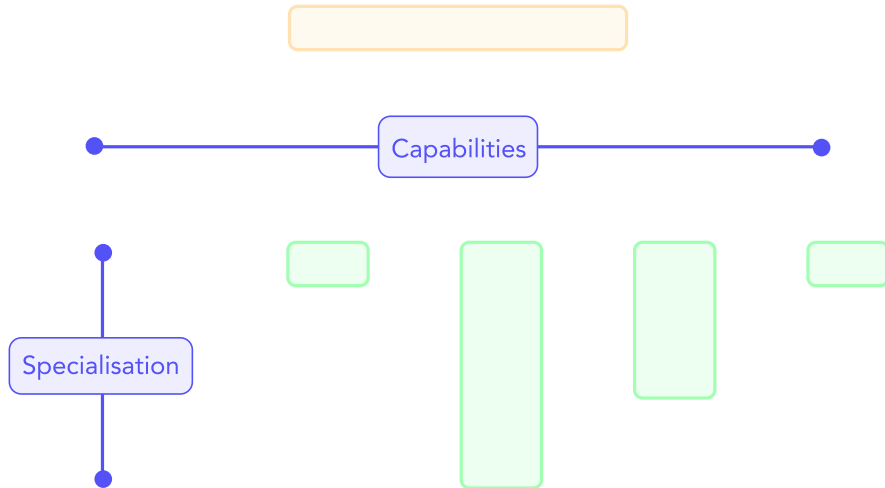


Figure 2: The original version of our system—a supervisor based multi-agent system.

2.2 Need for Multi-Agent Systems

There are a few key reasons why multi-agent systems make sense:

1. A single agent loaded with too many tools becomes harder to control. As the number of tools increases, the system prompt gets cluttered with more instructions, descriptions, and usage rules which can overwhelm the model and reduce its inference and response quality.
2. A single agent is limited by its context window. With no separate mechanism to store or retrieve information, tool outputs and accumulated data quickly consume available context, leading to overflow and degraded performance.
3. Specialization improves results. Different tasks require different reasoning styles or capabilities. By assigning roles to specific agents such as planning, reasoning, retrieval, or execution, the overall system performs more accurately and efficiently.

4. Multi-agent systems scale better. Tasks can be distributed or processed in parallel, which speeds up workflows and makes the system more efficient than a single agent trying to do everything sequentially.
5. They are more modular and flexible. Individual agents can be upgraded, replaced, or extended without redesigning the whole system. This makes maintenance and experimentation easier.
6. The distribution of responsibility reduces cognitive complexity. Each agent handles a focused set of instructions and tools, which generally leads to more consistent and reliable output.

3 System Architecture & Design Methodology

3.1 Design Choices & Rationale

Our multi-agent architecture supports multiple orchestration paradigms, each corresponding to a distinct mode of autonomy. At its core, the system comprises specialized orchestrators—such as supervisors, planners, workers, and, prospectively, delegators—that govern how member agents are selected, coordinated, and executed. These orchestrators enable the same underlying agents to operate under qualitatively different control regimes without altering their internal implementations.

In a fully autonomous configuration (chat mode), orchestration is handled by a supervisor agent equipped with an adaptive planner. This planner may be invoked at any point during execution to incorporate new information, recover from partial failures, or compensate for context limitations. Planning and execution are therefore tightly interleaved, allowing the system to dynamically revise its strategy in response to evolving conditions while maintaining forward progress.

In contrast, the fully controlled configuration (workflows mode) enforces a clear separation between planning and execution. Here, an explicit plan is generated upfront by a planner and surfaced to the user for inspection and iteration. Once finalized, this plan is delegated to a worker agent whose role is deliberately constrained to robust, deterministic execution. This mode prioritizes repeatability, auditability, and predictable behaviour across runs, making it well-suited for production workflows.

We further plan to introduce a delegator orchestrator that occupies an intermediate position between these two extremes. This orchestrator will selectively delegate subtasks while retaining partial oversight, enabling bounded autonomy without fully relinquishing control. The ability to support this spectrum of autonomy is the result of extensive software scaffolding informed by prior experience building production-grade systems. Crucially, member agents are strictly scoped and compositional, allowing them to be reused unchanged across different orchestration paradigms and autonomy modes.

Collectively, these design choices yield a system that supports controlled autonomy: flexible enough to adapt dynamically when required, yet structured enough to guarantee reliability, transparency, and reuse when constraints demand it.

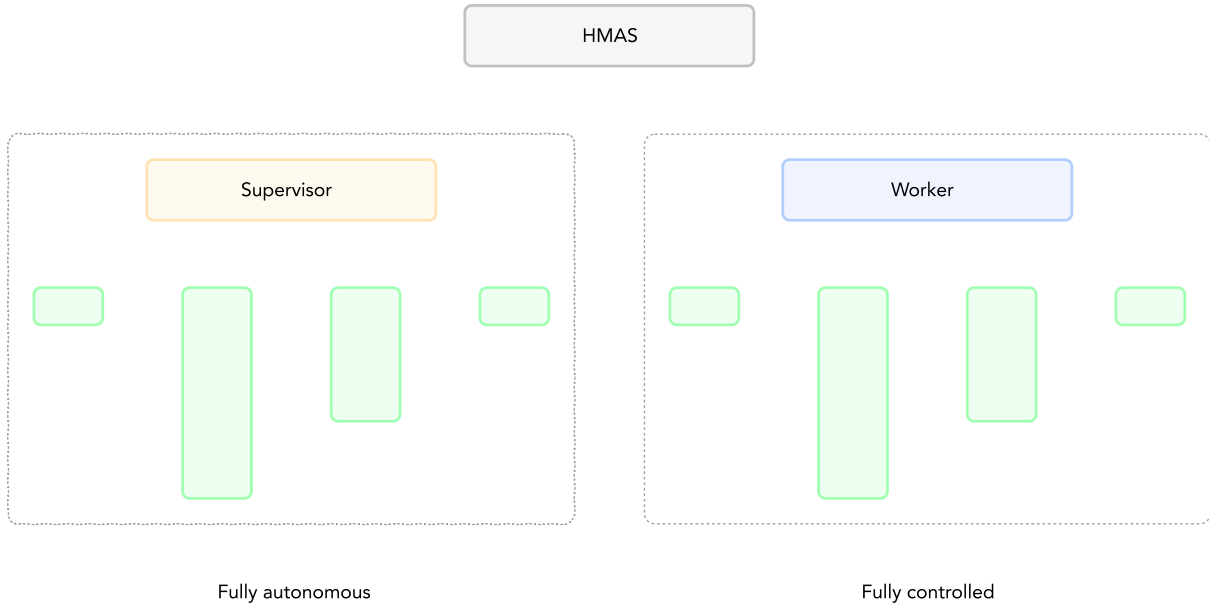


Figure 3: The current version hierarchical multi-agent system.

3.2 Opinionated Configuration

The system is built on a set of opinionated configurations designed with scalability and performance in mind. Most of these decisions fall under the umbrella of context engineering. This includes:

1. Maintaining a persistent storage layer for the agent state. This ensures that agents can retain relevant information across interactions, enabling continuity, reducing redundant computation, and supporting more coherent multi-step reasoning.
2. Determining which messages are sent to which components and controlling the amount of context passed at each step. By carefully selecting what each subsystem receives, the system avoids overloading agents with unnecessary information.
3. Enforcing structured outputs to ensure reliability and predictable behavior.
4. Research on continuously trimming or summarizing older messages to prevent context-window overflow while preserving essential information.
5. Directing messages according to their type (system, user, tool outputs, etc.) so that each component only processes the context appropriate for its function.
6. Using templates or schemas that enforce a uniform structure across agent prompts, which ensures smoother orchestration.

3.3 System Limitations

Despite the advantages offered by multi-agent systems, they come with several inherent limitations that shape both their design and their practical deployment. The most significant challenge is that these systems are largely constructed and refined through heuristics. Much of their behavior emerges from iterative human judgment, examining outputs, adjusting prompts, modifying workflows, and determining through intuition

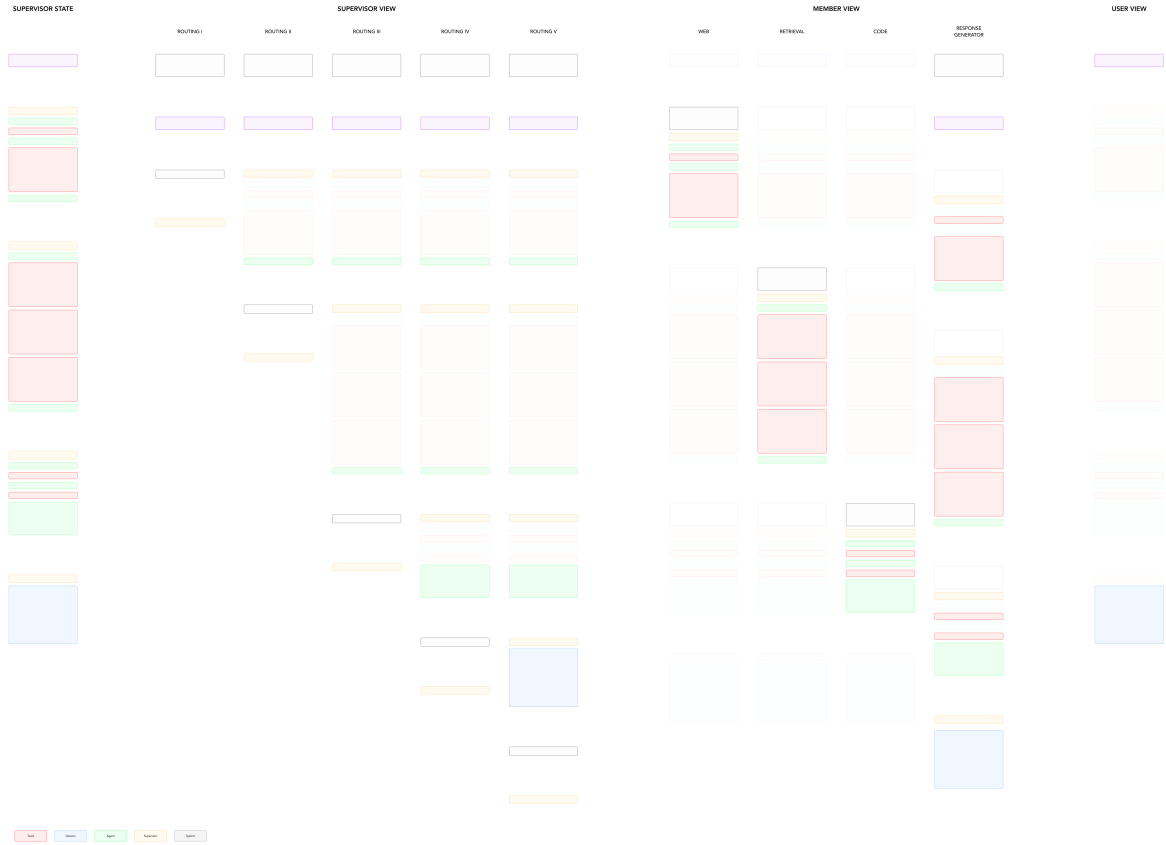


Figure 4: Advanced context engineering to constrain the agent’s worldview.

what feels correct or effective. This leads to a development process that relies more on manual judgment than on a scientific method.

Another limitation is the absence of a universal benchmark for evaluating multi-agent systems. Unlike traditional machine learning models, which can be assessed using standardized metrics and datasets, agentic systems vary widely in their architecture, intent, and operational constraints. Their performance depends not only on the base LLM but also on tool interactions, message routing strategies, memory mechanisms, and coordination policies. Because these factors differ significantly across implementations, no single metric or evaluation framework can meaningfully capture the overall quality or robustness of such a system.

This dependence on subjective heuristics and the lack of consistent evaluation standards makes it difficult to compare different system designs or confidently measure progress. These limitations motivate a deeper exploration into structured optimization techniques and reliable evaluation frameworks for multi-agent systems. The following sections introduce a practical approach to benchmarking and improving such systems in a way that reduces reliance on manual heuristics and provides repeatable and measurable insights.

4 Evaluating the System

4.1 Overall framework

Evaluating multi-agent systems requires a structured framework that captures both end-to-end system performance and the internal capacities of individual agents and their interactions. We organize our evaluation approach along four high-level verticals:

1. Execution completeness
2. Domain capacity
3. Planning capacity
4. Coordination capacity

Together, these dimensions provide coverage across black-box outcomes, agent-level competencies, and system-level dynamics.

4.1.1 Execution Completeness

The first vertical evaluates the system as a black box, focusing on the quality, correctness, and completeness of the final responses. The evaluation pipeline is fully programmatic, encompassing execution, response parsing and evaluation.

Completeness is evaluated using an LLM-based evaluator \mathcal{E} operating under a fixed evaluative role. For each evaluation instance, the system under evaluation \mathcal{S} is executed on a query $q \in \mathcal{Q}$ with context metadata $c \in \mathcal{C}$, producing an execution context $x \in \mathcal{X}$ that captures all intermediate artifacts. Together with an evaluator system prompt $p \in \mathcal{P}$, these form the evaluator input, and the evaluator produces a deterministic structured output

$$\mathcal{E}(q, c, x, p) = y.$$

The output y consists of categorical and numerical components, $y = (y_{\text{cat}}, y_{\text{num}})$, where y_{cat} is a finite set of discrete categorical labels and $y_{\text{num}} = (y_1, \dots, y_K) \in [0, 1]^K$ is a vector of normalized numerical scores, with higher values indicating better performance along the corresponding evaluation dimensions.

The *completeness score* for a single instance is defined as

$$C(q, c, x) = g(y_{\text{cat}}, y_{\text{num}}),$$

where $g : \mathcal{Y} \rightarrow [0, 1]$ is an aggregation function fixed prior to evaluation. The function g may combine numerical scores and apply conditional penalties or gating based on all or a subset of categorical outputs. By conditioning on the execution context x , the resulting score captures both response quality and execution-aware completeness. For a dataset $\mathcal{D} = \{(q_i, c_i, x_i)\}_{i=1}^N$, the evaluator is applied independently to each instance, yielding completeness scores $C_i = C(q_i, c_i, x_i)$. The dataset-level response completeness is then

$$C_{\mathcal{D}} = \frac{1}{N} \sum_{i=1}^N C_i.$$

In addition to this scalar aggregate, the empirical distributions of categorical outputs and numerical scores may be analyzed to provide diagnostic insight into system behaviour and systematic failure modes.

4.1.2 Planning Capacity

The second vertical evaluates the supervisor agent’s ability to decompose tasks, delegate subtasks appropriately, and converge toward a solution. Due to the lack of reliable ground-truth trajectories, we frame this evaluation primarily in terms of robustness rather than optimality.

Our approach analyzes agent trajectories across multiple rephrased variants of the same task, measuring convergence behaviour, number of steps taken, and consistency of delegation patterns. By examining whether similar task variants lead to stable planning strategies and termination states, we assess the resilience and coherence of the planning process without requiring explicit verification of final answers.

Robustness Let \mathcal{S} denote a multi-agent system treated as a black box. For a fixed query–context pair (q, c) , generate N semantic variations $\{q^{(i)}\}_{i=1}^N$. Each evaluation run induces an execution trace that is mapped to a task-sequence string via $\tau : \text{Traces} \rightarrow \mathbf{A}^*$, where \mathbf{A} is the set of agent identifiers, and $\mathbf{A}^* = \{(a_1, \dots, a_n) \mid n \in \mathbb{N}, a_i \in \mathbf{A}\}$ is the set of finite length sequences of agents. The resulting agent-sequence for the i -th variation is

$$s^{(i)} = \tau(\mathcal{S}(q^{(i)}, c)).$$

Let $d(\cdot, \cdot)$ denote the Levenshtein edit distance on \mathbf{A}^* .

Relative Robustness Relative robustness measures the consistency of task sequences across semantic query variations. For two runs i and j , define the normalized pairwise distance

$$\delta_{\text{rel}}(s^{(i)}, s^{(j)}) = \frac{d(s^{(i)}, s^{(j)})}{\max\{|s^{(i)}|, |s^{(j)}|\}}.$$

The relative robustness score is then given by

$$R_{\text{rel}}(q, c; N) = 1 - \frac{2}{N(N-1)} \sum_{1 \leq i < j \leq N} \delta_{\text{rel}}(s^{(i)}, s^{(j)}),$$

with $R_{\text{rel}} \in [0, 1]$, where higher values indicate greater invariance of the agent-level task decomposition under semantic perturbations.

Absolute Robustness Assume a single ground-truth task sequence $s^* \in \mathbf{A}^*$ is defined for the original query q . For each run, define the normalized distance to ground truth as

$$\delta_{\text{abs}}(s^{(i)}, s^*) = \frac{d(s^{(i)}, s^*)}{|s^*|}.$$

The absolute robustness score is defined as

$$R_{\text{abs}}(q, c; N) = 1 - \frac{1}{N} \sum_{i=1}^N \delta_{\text{abs}}(s^{(i)}, s^*),$$

which measures agreement with the intended task sequence, scaled by its reference length.

Extension The same formulation applies recursively at finer granularities by redefining τ to map execution traces to tool-call sequences within individual agents, yielding hierarchical robustness metrics at both the agent-routing and internal execution levels.

4.1.3 Domain Capacity

This evaluation vertical assesses the competence of individual agents against domain-appropriate benchmarks. Each agent type—web, retrieval, database, code, etc.—have distinct operational semantics and output modalities, and therefore requires a tailored evaluation strategy rather than a uniform metric.

Retrieval Agent is responsible for grounding system responses in external knowledge and is evaluated using a combination of internal research artifacts and established external frameworks.

We build on our internal *ARAGOG* framework, which formalizes retrieval evaluation through structured output grading for retrieval-augmented generation pipelines. *ARAGOG* enables controlled comparison of retrieval strategies, including reranking, query expansion, and multi-hop retrieval, while explicitly separating retrieval fidelity from generative fluency [Pre24].

To operationalize large-scale and reproducible evaluation, we leverage the *RAGAS* framework to generate knowledge-graph-based datasets supporting both single-hop and multi-hop question–answer pairs. *RAGAS* metrics are used to measure retrieval relevance, context utilization, and answer faithfulness, enabling fine-grained attribution of errors within retrieval-centric workflows [Exp23].

In addition, we incorporate the *UDA* benchmark, which consists of PDF-derived, real-world unstructured documents paired with expert-annotated question–answer sets. *UDA* provides a realistic stress test for retrieval robustness under noisy and heterogeneous document conditions, complementing more controlled KG-based evaluations [H+24].

Together, *ARAGOG*, *RAGAS*-generated datasets, and *UDA* form a complementary evaluation stack spanning theoretical grounding, synthetic control, and real-world retrieval performance.

Database Agent translates natural language instructions into executable SQL queries over structured schemas. Its evaluation is aligned with widely adopted text-to-SQL benchmarks that provide execution-validated ground truth.

We reference the *Spider* benchmark, which evaluates cross-domain semantic parsing with complex SQL queries and diverse schemas [YZY+18]. To reflect enterprise-scale complexity, we additionally consider *Spider 2.0*, which introduces larger schemas, multi-query tasks, and more realistic database workflows [ZLL+23]. These benchmarks enable objective measurement of query correctness, compositional generalization, and schema adaptation.

Code Agent is evaluated on its ability to generate correct and executable Python programs from natural language specifications. We align this evaluation with established program synthesis benchmarks.

Primary references include *HumanEval*, which measures functional correctness through unit-test-driven evaluation, and *MBPP*, which focuses on basic programming and algorithmic reasoning tasks [CTJ+21, AON+21]. For broader coverage, extended benchmarks such as *APPS* and *DS-1000* are used to assess robustness, library usage, and multi-step reasoning in realistic coding scenarios.

Generated programs are executed against test oracles, allowing direct measurement of correctness, failure modes, and runtime behaviour.

Other Agents including the web agent (search and link retrieval) and MCP-based agents, are currently evaluated using task-specific performance checks rather than full benchmark suites. These agents are treated as extensible components, with evaluation strategies to be formalized as their functional scope stabilizes and suitable public benchmarks mature.

4.1.4 Coordination Capacity

The final vertical focuses on inter-agent coordination, evaluated through message passing and communication structure across agents. This dimension captures how effectively agents exchange context, propagate intermediate results, and avoid redundant or conflicting actions.

4.2 Datasets

Evaluating a multi-agent system requires selecting a dataset that meaningfully reflects the system’s capabilities, particularly its ability to reason, coordinate tools, and operate across diverse modalities. After surveying multiple available benchmarks, the GAIA Benchmark [MFW+24] emerged as the most suitable for this purpose. GAIA is composed of question–answer pairs organized into levels of increasing difficulty, each designed to test progressively more complex forms of reasoning and tool usage.

A defining characteristic of GAIA is its emphasis on grounded real-world tasks. Questions often require interpreting or extracting information from heterogeneous data sources such as PDFs, images (for example, JPG and PNG), slide decks, videos, websites, and code snippets. This aligns closely with the operational demands placed on agentic systems, where success often depends not merely on language understanding but on the ability to select appropriate tools, parse non-text content, and synthesize information across formats.

By centering on tasks that require multimodal understanding and structured tool use, GAIA offers a challenging and relevant benchmark. For the purposes of this evaluation, a subset was selected of Level 1 non-multimodal questions, which aligns with the current capabilities of the system. This subset provides a practical way to assess how well the system handles structured workflows, coordinates agents, and produces accurate, grounded outputs, making it a suitable foundation for evaluating the architecture presented in this work.

Scoring The GAIA benchmark uses exact-match scoring with type-specific normalization. For each question i , the system computes a binary score

$$s_i = \begin{cases} 1 & \text{if } \text{normalize}(\hat{y}_i) = \text{normalize}(y_i), \\ 0 & \text{otherwise,} \end{cases}$$

where \hat{y}_i denotes the predicted answer of the model and y_i denotes the ground-truth answer. The normalization function applies lightweight cleaning appropriate to the answer type (e.g., removing punctuation/whitespace, standardizing numbers, or splitting lists).

The overall accuracy percentage is as follows:

$$\text{Accuracy}(\%) = 100 \times \frac{1}{N} \sum_{i=1}^N s_i.$$

where N is the total number of questions evaluated.

4.3 Environments

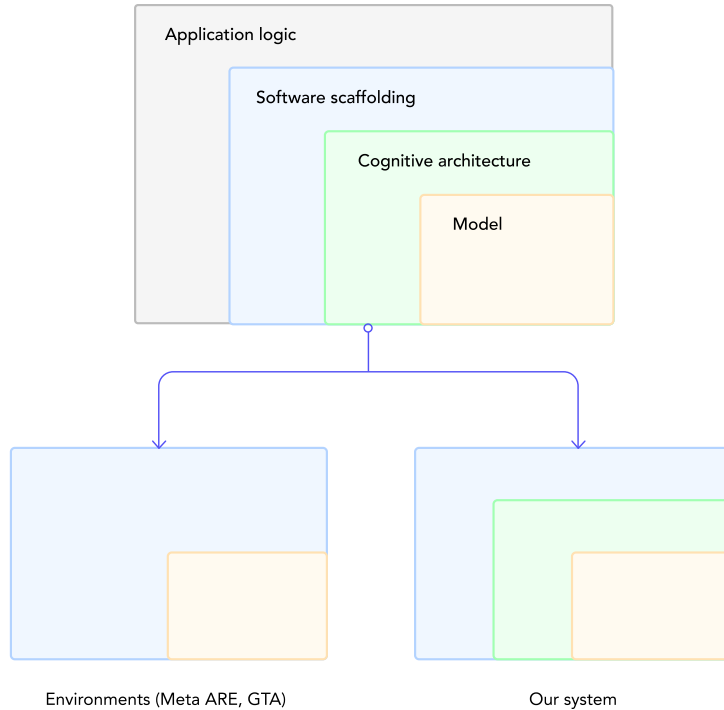


Figure 5: How agent environments relate to our multi-agent system.

While datasets provide the tasks an agent must solve, the environment defines the world in which those tasks are executed. In the context of multi-agent systems, an evaluation environment is not merely a software dependency but a sand-boxed reality that serves as a containerized interface governing the agent’s observation space (what it sees) and action space (what it can do).

For static benchmarks (like typical Q&A datasets), the environment is often trivial. However, for agentic tasks that involve tool use, coding, or web browsing, the environment is critical for three reasons.

- Safety - prevent agents from damaging real systems
- Reproducibility - ensuring that the state of the world is identical for every run
- Determinism - control external variables like network latency or API changes.

Several specialized environments have been developed to standardize this process. [FAB+25] Meta’s Agents Research Environment (ARE) introduces an asynchronous, time-driven simulation for GAIA 2, while others target specific domains such as [XZC+24] OSWorld for operating system control, [ZXZ+24] WebArena for web browsing, and [JYW+24] SWE-bench for software engineering tasks. These frameworks typically provide safe, deterministic sandboxes that isolate agent actions from external variables.

For this evaluation, however, the multi-agent system itself was utilized as the environment. Rather than wrapping the agents in an external simulator, all relevant dataset files and context were directly uploaded and connected to the system’s knowledge base. The benchmarking was conducted programmatically via the system’s API, injecting the GAIA question-answer pairs as standard user requests to evaluate the end-to-end performance in its native operational state.

4.4 Outcomes & Limitations

Running the GAIA evaluation provides direct insight into the strengths and limitations of the current multi-agent architecture. Because each question is paired with a known ground truth answer, the evaluation loop makes it possible to identify where the system succeeds, where it fails, and why. In practice, this feedback highlights which components require revision, such as specific agent behaviors, tool selection policies, or prompt configurations. This creates an iterative improvement cycle: run the evaluation, analyze failure cases, modify the system or prompts, and rerun. At the time of writing, only the Level 1 subset has been executed on our staging instance and some of the improvements we are working as far are as follows:

- Improve agent orchestration and replanning logic, especially when new information is gathered or to avoid infinite loops.
- Enhance task decomposition and supervisor instructions to sub-agents for better clarity and structure.
- Update and structure system prompts and metadata for both member and supervisor agents.
- Improve web and database agent capabilities, including better referencing, context retrieval, and new tools for filtering and fetching data.
- Add few-shot prompting to workflows and supervisor for more robust agent responses.

LLMs as Evaluators

Although LLMs are frequently used as evaluators, our findings indicate that they are fundamentally ill-suited for rigorous multi-agent evaluation. Using a non-deterministic, black-box model to assess outputs generated by similar abstractions leads to unavoidable bias propagation, making it impossible to disentangle evaluator bias from true system performance.

- In practice, we observe strong correlation effects between the model family used in the evaluated system and the model family used as the judge. When both belong to the same lineage, evaluation scores are systematically inflated across metrics, indicating representational alignment rather than objective assessment.
- Additional bias arises from differences in model training, alignment, and deployment. Variations in instruction-following behaviour, safety guardrails, and provider-level scaffolding materially influence evaluation outcomes, causing scores to reflect evaluator-specific artifacts rather than intrinsic system quality.
- Finally, evaluation design choices introduce implementation bias. Structured-output scoring constrains assessment to incomplete metric sets, while option-based prompting exhibits positional bias.

Together, these limitations motivate a shift toward more holistic evaluation methods that open up the possibilities of practically realizing self-optimizing multi-agent systems. We propose one such mathematical formalization in the following section.

5 Optimizing the System

Here we consider the task of finding an optimal system design. We let \mathcal{S} be the search space of systems and $f : \mathcal{S} \rightarrow \mathbb{R}$ an evaluation function, e.g., a linear combination of metrics measuring the systems performance on a fixed benchmark set. Our goal is to identify

$$S^* := \operatorname{argmax}_{S \in \mathcal{S}} f(S) \quad (1)$$

5.1 Optimization Approach

In this Section, we first consider our search space complexity, finding it to be exponential (5.2). Based on this knowledge and the complex, black-box nature of the optimization, we systematically rule out all optimization approaches except Genetic Algorithms (GAs) and Bayesian Optimization (BO) (5.3). We select the latter as the best fit, given that it excels when the input dimensionality is low and the cost of evaluating a proposed system is high. Since a surrogate model (posterior) is required for BO, we will use Gaussian Processes (GP) due to their many successful applications and the wealth of research support available for them. Detailed formulation of the algorithm is given (5.4).

Notably, relatively fast convergence guarantees (sublinear regret) are known for BOGP when using a scheduled Upper Confidence Bound (UCB) acquisition function (5.5). Such a guarantee is highly desirable. Unfortunately, the existing proof relies on a fixed kernel function.

However, because our target system optimization involves unknown, complex relationships between the inputs (models, tools, etc.) and the objective function, we posit that standard kernel functions are unlikely to be sufficient. Adaptive kernels and feature embeddings, as utilized in kernel learning, represent the most promising path forward (5.6).

This leaves us with two options: 1) Proceed with kernel learning and forgo the theoretical convergence guarantee; or 2) Identify and prove an extended BOGP guarantee suitable for adaptive kernels and kernel learning approaches. We plan to pursue both strategies in parallel.

The rest of this section documents the details leading to this plan.

5.2 Search Space Complexity

As an example of the complexity in such a system design, we fix the following hyperparameters for the system:

- M_m : the number of distinct LLM models considered.
- M_t : the number of distinct tools considered.
- C_τ : the maximum allowed number of calls for tool $\tau \in \{1, \dots, M_t\}$

and count the size of our search space. An individual agent A_i is defined by selecting exactly one of the M_m models and any subset of the M_t tools (ranging from the empty set to all tools).

Consequently, the total number of possible unique agents, denoted N_a , is given by:

$$N_a = M_m \times \sum_{k=0}^{M_t} \binom{M_t}{k} = M_m \times 2^{M_t}$$

Our system requires a supervisor agent and is capped at a maximum of M_a total agents. We let $1 \leq n_a \leq M_a$ denote the variable number of agents in a specific system configuration. Assuming the system is defined by an ordered sequence of agents, the total number of possible systems is the sum of the search spaces for each system size from 1 to M_a :

$$\sum_{k=1}^{M_a} N_a^k = N_a + N_a^2 + \dots + N_a^{M_a} = N_a \frac{N_a^{M_a} - 1}{N_a - 1}$$

It follows that the asymptotic complexity of the search space is:

$$\mathcal{O}(N_a^{M_a}) = \mathcal{O}((M_m 2^{M_t})^{M_a}) = \mathcal{O}(M_m^{M_a} 2^{M_t M_a}) \quad (2)$$

Searching over a finite set of sequences (C_τ) and over a finite set of system prompts for the supervisor agent will increase the search space complexity (Eq. 2) multiplicatively. For now consider these fixed.

5.3 Optimization Techniques

To identify viable optimization strategies, first note that a system S is determined by its agent set, where an agent A is specified by its model $m(A)$ and its tool set $t(A) = \{t(A)_1, \dots, t(A)_k\}$ for some $k \in \mathbb{N}_0$. This can be formulated using binary variables: $\{m_{a,i} : i = 1, \dots, M_m\}$ indicating agent a 's use of model i , subject to the constraint $\sum_i m_{a,i} \leq 1$; and $\{t_{a,j} : j = 1, \dots, M_t\}$ indicating agent a 's use of tool j . Notably, such systems may experience complex unknown interactions. For example, tool t_1 might make tool t_2 useless, or Agent A might increase system performance with no tools but decrease performance if given tool t_1 . Thus, we face a black-box, integer programming optimization problem, where the objective function $f(S)$ is non-differentiable, costly to compute, and cannot be factored simply into the variables of S .

Our counting argument shows that even with a modest number of models, tools, and a maximum number of agents, the search space is intractable for brute-force search. As a simple example, suppose we have $M_m = 5$ different LLMs, $M_t = 5$ different tools, and a maximum of $M_a = 5$ agents in the system. The search space size is roughly $\mathcal{O}(5^5 \times 2^{25}) > 10^{10}$. Dynamic Programming and Branch and Bound methods require structure in f that we lack, making them inapplicable. We know of no exact optimization methods that are feasible for this problem.

Moving to heuristic methods, we consider greedy algorithms (e.g., iteratively choosing a random neighboring system, evaluating it, and keeping the new system if it is better). These are not guaranteed to converge, and we lack a strategy to guide the search effectively. Similarly, their stochastic counterparts (e.g., Simulated Annealing, where one probabilistically keeps or rejects a worse system) do have convergence guarantees, but they require an extremely slow cooling schedule ($\mathcal{O}(\log^{-1}(t))$) to hold [GG84]. These are therefore a poor fit.

Genetic Algorithms (GAs) [Hol75] are well suited for black-box optimization and have been used in situations where unknown complex interactions are expected. Notably,

GAs produce the next samples to evaluate very quickly, but they require many iterations to find near-optimal solutions. In particular, GAs have been effective in finding multi-agent LLM system designs [YSC⁺25]. While convergence guarantees have been proven in some cases [Rud94], GAs are known to suffer from premature (sub-optimal) convergence. According to an expert we consulted (Professor Erik Hemberg, MIT), GAs are good for exploration of the search space but often fail to converge efficiently.

Bayesian Optimization (BO) refers to the general iterative algorithm of defining a probabilistic surrogate model of the objective function and using it to identify which system to evaluate next. After each evaluation, the probabilistic surrogate is updated. In short, BO builds a map of the unknown search space to enable informed guessing and accelerate optimization. BO with Gaussian Processes (BOGP) specifies the probabilistic assumptions using Gaussian distributions, utilizing a kernel function to relate system designs to the probability distribution of the objective function, conditioned on all prior observations. There are two main drawbacks of BOGP. First, identifying the candidate for the next evaluation is computationally costly. Updating the distribution requires inverting an $n \times n$ matrix, which is $\mathcal{O}(n^3)$ where n is the number of observed data points, followed by an internal optimization step. Second, BOGP can struggle when optimizing in very high dimensions.

Overall, for black-box situations, GAs and BOGP appear to be the best classical methods. If the evaluation of the objective function is slow (as is our case), research suggests BOGPs are worth the computational cost because they identify the next evaluation point more intelligently [LTRE22]. Conversely, if evaluation is relatively cheap, GAs—which require little time to identify candidates but many iterations of evaluations—can be better [AL21]. Hybrid BOGP–GA algorithms have also been proposed [LTRE22].

Based on these findings, we pursue BOGP.

5.4 Bayesian Optimization with Gaussian Processes

Our goal is the optimization of $f(x)$, representing the evaluation of a multi-agent system as discussed above, where x denotes the design choices; however, we introduce the optimization method here in full generality. We consider the problem of optimizing an unknown reward function $f : D \rightarrow \mathbb{R}$ over a domain $D \subseteq \mathbb{R}^d$ using a Bayesian Optimization (BO) iterative algorithm with a Gaussian Process (GP) model. We assume generally that we can evaluate f with noise, but that evaluations are costly, and typically analytic or gradient-based techniques are inapplicable for the optimization of f .

(Notational note: while we previously used t_j to denote the tools used by an agent, we now let t and t_j to denote the rounds of an optimization algorithm. This follows the literature, and the ambiguity should not cause confusion). In each round $t = 1, 2, \dots, T$, we choose a point $x_t \in D$ and obtain a noisy observation y_t :

$$y_t = f(x_t) + \epsilon_t$$

where the noise terms $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ are independent and identically distributed (i.i.d.) Gaussian noise. The noise variance σ^2 is assumed to be a fixed positive constant. We denote the history of observed input-output pairs up to round $t - 1$ as $\mathcal{D}_{t-1} = \{(x_1, y_1), \dots, (x_{t-1}, y_{t-1})\}$. For convenience, we also define $\vec{x}_{t-1} = [x_1, \dots, x_{t-1}]^\top$ as the vector of observed inputs and $\vec{y}_{t-1} = [y_1, \dots, y_{t-1}]^\top$ as the vector of corresponding observed outputs.

The GP model for f is characterized by its mean function and covariance (kernel) function. We assume a zero prior mean function:

$$\mu_0(x) = \mathbb{E}[F(x)] = 0 \quad \forall x \in D$$

(while, in general, if some prior data is known, a random variable $Y|x$ can have a non-zero prior mean).

The covariance (kernel) function $k : D \times D \rightarrow \mathbb{R}$ is assumed to be positive semi-definite (PSD), symmetric ($k(x, x') = k(x', x)$), and continuous. We assume k is uniformly bounded by a positive constant Λ , such that $k(x, x) \leq \Lambda$ for all $x \in D$, and that $k(x, x) > 0$ (non-degenerate) for all $x \in D$. The covariance of our Gaussian process is given by k ; that is,

$$k(x, x') = \mathbb{E}[(F(x) - \mu_0(x))(F(x') - \mu_0(x')))] = \mathbb{E}[F(x)F(x')]$$

We abuse notation and write $k(\vec{x}, \vec{x}')$ to denote the $|\vec{x}| \times |\vec{x}'|$ matrix where the entry at row i and column j is $k(x_i, x'_j)$ for $x_i \in \vec{x}$ and $x'_j \in \vec{x}'$. (F will be used to denote the random variable corresponding to our GP; e.g., $F(x) \sim N(0, k(x, x))$ while $f(x)$ denotes the true, deterministic value.)

It follows that given the noisy observations $\mathcal{D}_{t-1} = (\vec{x}_{t-1}, \vec{y}_{t-1})$, the posterior distribution over f is again a GP. Specifically, for any set of (new) test inputs $\vec{x}_* = [x_1^*, \dots, x_m^*]^\top$, the corresponding (unobserved) function values $\vec{F}(\vec{x}_*) = [F(x_1^*), \dots, F(x_m^*)]^\top$ follow a multivariate Gaussian distribution conditioned on previous observations:

$$\vec{F}(\vec{x}_*) | \mathcal{D}_{t-1} \sim \mathcal{N}(\mu_{t-1}(\vec{x}_*), \Sigma_{t-1}(\vec{x}_*))$$

where the posterior mean vector $\mu_{t-1}(\vec{x}_*)$ and posterior covariance matrix $\Sigma_{t-1}(\vec{x}_*)$ are given by:

$$\begin{aligned} \mu_{t-1}(\vec{x}_*) &= k(\vec{x}_*, \vec{x}_{t-1})(k(\vec{x}_{t-1}, \vec{x}_{t-1}) + \sigma^2 I)^{-1} \vec{y}_{t-1} \\ \Sigma_{t-1}(\vec{x}_*) &= k(\vec{x}_*, \vec{x}_*) - k(\vec{x}_*, \vec{x}_{t-1})(k(\vec{x}_{t-1}, \vec{x}_{t-1}) + \sigma^2 I)^{-1} k(\vec{x}_{t-1}, \vec{x}_*) \end{aligned} \quad (3)$$

Note that μ_{t-1} depends on both \vec{x}_{t-1} and \vec{y}_{t-1} , while Σ_{t-1} only depends on \vec{x}_{t-1} . Now, for any single test point x , its posterior mean $\mu_{t-1}(x)$ and posterior variance $\sigma_{t-1}^2(x)$ are given by:

$$\begin{aligned} \mu_{t-1}(x) &= k(x, \vec{x}_{t-1})(k(\vec{x}_{t-1}, \vec{x}_{t-1}) + \sigma^2 I)^{-1} \vec{y}_{t-1} \\ \sigma_{t-1}^2(x) &= k(x, x) - k(x, \vec{x}_{t-1})(k(\vec{x}_{t-1}, \vec{x}_{t-1}) + \sigma^2 I)^{-1} k(\vec{x}_{t-1}, x) \end{aligned} \quad (4)$$

Now equipped with GP machinery furnishing a computable posterior, we use a BO process. To do so, we require an *acquisition function* $\alpha_t(x)$, which is used to identify the next data point to evaluate, and may vary with the iteration t . We also require a real, positive definite kernel function k , and stopping criterion \mathcal{S} . The BOGP process in round t is:

1. Compute the sufficient statistics $(\mu_{t-1}, \Sigma_{t-1})$ for the posterior based on all prior observations;
2. Choose the next data point to sample by computing $x_t := \operatorname{argmax}_x \alpha_t(x)$.
3. Evaluate $y_t = f(x_t) + \epsilon_t$, and store (x_t, y_t) .

4. If the stopping criterion is met, return the best previously observed data; else, continue.

Generally, α_t is chosen to be fast to optimize, e.g., admitting gradient or analytic techniques, so steps 1 and 3 are the computationally expensive steps.

5.5 GP-UCB Convergence Guarantees

Convergence guarantees for BOGP are proven under specific settings in Srinivas et al. [SKKS10]. This section gives a concise overview of their main theorem (1) for optimization over a finite domain, as is our situation. We then discuss its limitations and implications for future research.

The optimization guarantee requires the Upper Confidence Bound (UCB) acquisition function,

$$\alpha_t(x) = \mu_{t-1}(x) + \sqrt{\beta_t \sigma_{t-1}^2(x)} . \quad (5)$$

Here $\beta_t > 0$ is a “confidence parameter”, a constant that balances between exploration and exploitation throughout the BOGP process. That is, it encourages exploration by favoring areas of the domain where our posterior variance $\sigma_{t-1}^2(x)$ on the function value is large (high β_t) and exploitation by favoring areas where we have high expected function values $\mu_{t-1}(x)$ (low β_t). We index β_t by time t as the balance will change in each round of the process to ensure convergence.

The main idea of Srinivas et al. is to change the value of the confidence parameter each iteration to guarantee convergence. In order to understand such a schedule and how it ensures convergence, we require notions of *regret*, the quantified error of the process. The next subsection provides a short but formal treatment of regret. The important takeaway is that if a process is “no-regret”, then the best seen value in the process becomes arbitrarily close to the actual optimum (Proposition 2).

5.5.1 Types of Regret

The performance of the optimization algorithm can be quantified asymptotically using the concept of *no regret*.

Definition 1 (Instantaneous, Cumulative, and No Regret). *Let $x^* = \operatorname{argmax}_{x \in D} f(x)$ be a true maximizer of the unknown function f . The **instantaneous regret** r_t at round t is the loss incurred due to not choosing x^* :*

$$r_t = f(x^*) - f(x_t) .$$

The **cumulative regret** R_T after T rounds is the sum of instantaneous regrets:

$$R_T = \sum_{t=1}^T r_t .$$

Previous works seek optimization processes that are **no regret**, meaning

$$\lim_{T \rightarrow \infty} \frac{R_T}{T} = 0$$

No regret does not imply that $r_t \rightarrow 0$, although that is certainly sufficient. Our next proposition shows instead that no regret implies that R_T is sublinear asymptotically.

Proposition 1. *Suppose our process is no-regret ($\lim_{T \rightarrow \infty} R_T/T = 0$), fix any $\delta' > 0$, and let $m(T, \delta') := |\{t \leq T : r_t > \delta'\}|$ = the number of rounds $t \leq T$ such that $r_t > \delta'$. Then $\lim_{T \rightarrow \infty} m(T, \delta')/T = 0$; i.e., $m(T, \delta')$ is sublinear in T for any δ' .*

Proof. For the given δ' we can lower bound the average regret as:

$$\frac{1}{T} \sum_{t=1}^T r_t \geq \frac{1}{T} \sum_{\{t \leq T : r_t > \delta'\}} r_t > \frac{m(T, \delta')\delta'}{T}.$$

Now, since $\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T r_t = 0$, given any $\epsilon > 0$, there exists some N such that for all $T > N$, we have $\frac{1}{T} \sum_{t=1}^T r_t < \epsilon$. Hence, for $T > N$ we have

$$\epsilon > \frac{m(T, \delta')\delta'}{T}.$$

Rearranging gives

$$\frac{m(T, \delta')}{T} < \frac{\epsilon}{\delta'}$$

Since ϵ can be chosen arbitrarily small (for any fixed δ'), this implies that $\lim_{T \rightarrow \infty} m(T, \delta')/T = 0$, meaning $m(T, \delta')$ must grow on an order strictly less than linear in T . \square

Our next example exhibits a process that is no regret, but for which $r_t \not\rightarrow 0$.

Example 1. *Consider a sequence of instantaneous regrets where $r_t = 1$ only when t is a power of 2 (i.e., $t = 2^j$ for $j \in \mathbb{N}$), and $r_t = 0$ for all other t . Clearly $r_t \not\rightarrow 0$. The cumulative regret R_T will be bounded by the number of such non-zero terms up to T , which is $\approx \log_2 T$. Hence, $R_T \leq \log_2 T$. Then, the average regret is:*

$$\frac{R_T}{T} \leq \frac{\log_2 T}{T} \rightarrow 0 \quad \text{as } T \rightarrow \infty$$

So this is indeed a no-regret sequence, despite r_t not converging to zero.

Proposition 2. *Suppose our BO process is no-regret. Let $f_T := \max_{1 \leq t \leq T} f(x_t)$ (the best true function value seen so far). Then for all $\epsilon > 0$, there is an $N(\epsilon)$ such that for all $T > N(\epsilon)$,*

$$\epsilon > \left(f_{\max} - \frac{1}{T} \sum_{t=1}^T f(x_t) \right) \geq f_{\max} - f_T$$

Proof. By the definition of cumulative regret, and average regret:

$$\frac{R_T}{T} = \frac{1}{T} \sum_{t=1}^T (f_{\max} - f(x_t)) = f_{\max} - \frac{1}{T} \sum_{t=1}^T f(x_t)$$

The left inequality follows directly from the definition of a no-regret BO process ($\lim_{T \rightarrow \infty} R_T/T = 0$). For the right inequality, observe that for any $t \in \{1, \dots, T\}$, $f(x_t) \leq \max_{1 \leq k \leq T} f(x_k) = f_T$. Therefore, $\frac{1}{T} \sum_{t=1}^T f(x_t) \leq \frac{1}{T} \sum_{t=1}^T f_T = f_T$. Multiplying by -1 reverses the inequality: $-\frac{1}{T} \sum_{t=1}^T f(x_t) \geq -f_T$. Adding f_{\max} to both sides preserves the inequality:

$$f_{\max} - \frac{1}{T} \sum_{t=1}^T f(x_t) \geq f_{\max} - f_T$$

Combining these proves the proposition. \square

5.5.2 Srinivas et al. [SKKS10] Convergence Theorem

Here we state the main optimization guarantee that applies to our MAS optimization, and then we discuss its proof. To do so, Srinivas et al. define the Information Gain, $I(\mathbf{y}_A; \mathbf{f}_A)$, as the mutual information between the observed noisy outputs $\mathbf{y}_A = \{y_i\}_{x_i \in A}$ and the true function values $\mathbf{f}_A = \{f(x_i)\}_{x_i \in A}$. Explicitly, $I(\mathbf{y}_A; \mathbf{f}_A) := H(\mathbf{y}_A) - H(\mathbf{y}_A | \mathbf{f}_A)$. This quantifies “the reduction in uncertainty about f from revealing y ” (recall $y = f + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma_y^2)$). The main theorem bounds cumulative regret in terms of the maximum information gain, $\gamma_T = \max_{A \subseteq D, |A|=T} I(\mathbf{y}_A; \mathbf{f}_A)$. Srinivas et al. then compute bounds on γ_T for three common kernel functions: Linear, Squared Exponential, and Matérn.

Theorem 1 (Srinivas et al. [SKKS10], Theorem 1). *Suppose our domain D is a finite set. Suppose f is a sample from the GP denoted above with mean zero, using a bounded, symmetric, positive semi-definite kernel k . Fix $\delta \in (0, 1)$ and assume IID sampling noise $\mathcal{N}(0, \sigma^2)$. Let γ_T be the maximum information gain after T rounds. Then, running GP-UCB with the parameter schedule $\beta_t = 2 \log(|D|t^2\pi^2/(6\delta))$ yields a cumulative regret bound of $\mathcal{O}^*(\sqrt{T\gamma_T \log |D|})$ with high probability. Precisely,*

$$\Pr \{ R_T \leq \sqrt{C_1 T \beta_T \gamma_T} \ \forall T \geq 1 \} \geq 1 - \delta$$

where $C_1 = 8/\log(1 + \sigma^{-2})$.

We refer the reader to the paper’s appendix for the detailed proofs (see the URL linked in our reference citation), but we outline the key ideas here with notes on our application. The proof proceeds in four steps:

1. Use a standard Gaussian tail bound combined with a union bound over D and time t to define β_t such that, with probability greater than $1 - \delta$, the true function value falls within the confidence interval for all x and t :

$$|f(x) - \mu_{t-1}(x)| \leq \sqrt{\beta_t \sigma_{t-1}(x)} .$$

2. Use the result from Step 1 and the definition of the UCB selection rule to bound the instantaneous regret:

$$r_t \leq \sqrt{\beta_t \sigma_{t-1}(x_t)} + \sqrt{\beta_t \sigma_{t-1}(x_t)} = 2\sqrt{\beta_t \sigma_{t-1}(x_t)} .$$

3. Apply the Cauchy-Schwarz inequality to the sum of regrets to obtain a bound on cumulative regret in terms of the sum of variances:

$$R_T \leq \sqrt{T \sum_{t=1}^T r_t^2} \leq \sqrt{T \sum_{t=1}^T 4\beta_t \sigma_{t-1}^2(x_t)} .$$

4. Relate the sum of posterior variances to the information gain. It is shown that the information gain can be expressed as $I(\mathbf{y}_T; \mathbf{f}_T) = \frac{1}{2} \sum_{t=1}^T \log(1 + \sigma^{-2} \sigma_{t-1}^2(x_t))$. (We note that this relationship exploited requires a fixed kernel function.) Since $s^2 \leq C \log(1 + \sigma^{-2} s^2)$ for bounded variances, the sum of variances is bounded by a constant multiple of γ_T . Substituting this back into the inequality from Step 3 yields the final result.

5.5.3 UCB Convergence Takeaways

The work of Srinivas et al. provides the mathematical foundation to guarantee BOGP convergence by furnishing a principled and scheduled acquisition function. Notably, the proof in Step 4 depends on a fixed kernel, implying that adaptive kernel methods are not immediately covered by this guarantee. Moreover, applying the bound requires computing or bounding γ_T , which is provided only for specific kernel types (Linear, Squared Exponential, Matérn).

5.6 BOGP Kernel Functions

One must define the kernel function, $k : D \times D \rightarrow \mathbb{R}$, such that any Gram matrix of the form $(k(x_i, x_j))$ generated by a finite subset of D is a real, symmetric positive semi-definite matrix. Intuitively, the kernel function k encodes the similarity between pairs in our domain D , which dictates the covariance structure of the posterior distribution as defined in Eq.s 3 & 4.

The choice of the kernel function is critical for BO, as the process depends almost entirely on the posterior distribution, and hence on the kernel. In other words, using a poorly chosen kernel function limits the posterior to being a poor surrogate model for the objective function.

Traditional kernels, such as linear, exponential, squared exponential (RBF), and Matérn kernels, are commonly used. These provide transparent, interpretable relationships between the inputs $x_i \in D$ and the posterior distribution. Most kernel classes possess hyperparameters that significantly affect the result [RSML19]. To address this, the sub-field of kernel learning has emerged. Adaptive kernel methods use a parameterized class of kernels, $k_\theta(x_i, x_j)$, and learn these parameters θ (e.g., via maximum likelihood estimation) as new data (x_i, y_i) is observed. Empirically, these often outperform fixed kernel approaches and maintain sample efficiency [RSML19]. Other parameters, such as the observation noise variance σ_y , can be learned similarly throughout the process [ORvdW21]. However, traditional kernel functions can be fundamentally too simple to capture complex correlations in the objective function. To overcome this, Deep Kernel Learning (DKL) adaptively learns a feature embedding $x \mapsto v_\phi(x)$ (often using a neural network), so the kernel used is $k_\theta(v_\phi(x_i), v_\phi(x_j))$. As before, the base kernel parameters θ can be learned simultaneously with the embedding parameters ϕ [WHSX16, ORvdW21]. If enough observations (rounds of BO) are available, such methods are often superior to other fixed or adaptive kernel techniques, though they typically require more data to converge [WHSX16].

6 Future work & Conclusions

This paper has outlined a practical approach to the design, optimization, and evaluation of multi-agent systems, addressing foundational challenges in orchestration, coordination, and performance measurement. By explicitly supporting controlled autonomy through opinionated orchestration patterns, and by leveraging systematic optimization techniques such as Bayesian Optimization with Gaussian Processes, we demonstrate how complex multi-agent systems may be made both adaptive and operationally reliable.

The evaluation framework presented here is intentionally incremental. In ongoing and future work, we will extend this framework to achieve comprehensive coverage across all

levels of the GAIA I benchmark, moving beyond initial task tiers to capture increasing degrees of difficulty, compositionality, and multi-modal reasoning. In parallel, we will examine alignment and overlap with other emerging datasets and interactive environments as they mature, and where necessary, synthesize bespoke evaluation datasets to address gaps not covered by existing benchmarks.

Beyond public benchmarks, our system is deployed across multiple client-specific domains, yielding a rich corpus of real-world interaction data. This includes explicit feedback signals on agent preferences, domain competence, and failure modes. We plan to operationalize this data through an automated improvement pipeline, in which agent-level diagnostics and refinements are periodically generated and applied, with significant components of this process driven by our internal coding agent.

Finally, we intend to incorporate reinforcement learning from human feedback (RLHF) to fine-tune the models trained over proprietary information and high-value domains. By combining benchmark-driven evaluation, production usage data, and feedback-informed learning, we aim to establish a continuous optimization loop that incrementally improves agent capability, reliability, and domain alignment over time.

References

- [AL21] Hussain Alibrahim and Simone A Ludwig. Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization. In *2021 IEEE congress on evolutionary computation (CEC)*, pages 1551–1559. IEEE, 2021.
- [AON⁺21] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.
- [CTJ⁺21] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- [Exp23] Exploding Gradients. Ragas: Automated evaluation for retrieval-augmented generation. <https://github.com/explodinggradients/ragas>, 2023. Accessed 2025.
- [FAB⁺25] Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. Are: Scaling up agent environments and evaluations, 2025.
- [GG84] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

- [H⁺24] Qinchuan Hui et al. Uda: A benchmark for unstructured document analysis. <https://github.com/qinchuanhui/UDA-Benchmark>, 2024. Associated paper and dataset.
- [Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.
- [JYW⁺24] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024.
- [LTRE22] Gongjin Lan, Jakub M Tomczak, Diederik M Roijers, and AE Eiben. Time efficiency in optimization with a bayesian-evolutionary algorithm. *Swarm and Evolutionary Computation*, 69:100970, 2022.
- [MFW⁺24] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for general AI assistants. In *The Twelfth International Conference on Learning Representations*, 2024.
- [ORvdW21] Sebastian W Ober, Carl E Rasmussen, and Mark van der Wilk. The promises and pitfalls of deep kernel learning. In *Uncertainty in Artificial Intelligence*, pages 1206–1216. PMLR, 2021.
- [Pre24] Predli. Aragog: Advanced retrieval-augmented generation output grading. *arXiv preprint arXiv:2404.01037*, 2024.
- [RSML19] Ibai Roman, Roberto Santana, Alexander Mendiburu, and Jose A Lozano. An experimental study in adaptive kernel selection for bayesian optimization. *IEEE Access*, 7:184294–184302, 2019.
- [Rud94] Günter Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE transactions on neural networks*, 5(1):96–101, 1994.
- [SKKS10] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: no regret and experimental design. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 1015–1022, 2010. Version including Appendix with proofs: <https://arxiv.org/pdf/0912.3995>.
- [WHSX16] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR, 2016.
- [WMV24] Julia Wiesinger, Patrick Marlow, and Vladimir Vuskovic. Agents. *Whitepaper*. Available online: <https://www.rojo.me/content/files/2025/01/Whitepaper-Agents—Google.pdf> (accessed on 14 December 2024), 2024.
- [XZC⁺24] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024.

- [YSC⁺25] Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. EvoAgent: Towards automatic multi-agent generation via evolutionary algorithms. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6192–6217, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics.
- [YZY⁺18] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanella Roman, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, 2018.
- [ZLL⁺23] Ruiqi Zhong, Tao Li, Fang Liu, et al. Spider 2.0: Evaluating text-to-sql systems in enterprise settings. *arXiv preprint arXiv:2305.16265*, 2023.
- [ZXZ⁺24] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024.