Language Agents as Optimizable Graphs

Mingchen Zhuge[1]★, Wenyi Wang[1]★, Louis Kirsch[2], Francesco Faccio[1,2]
Dmitrii Khizbullin[1]♦, Jürgen Schmidhuber[1,2]

[1]AI Initiative, King Abdullah University of Science and Technology,
[2]The Swiss AI Lab IDSIA, USI, SUPSI

{mingchen.zhuge, wenyi.wang, francesco.faccio,
dmitrii.khizbullin, juergen.schmidhuber}@kaust.edu.sa, louis@idsia.ch

https://gptswarm.org

Abstract

Various human-designed prompt engineering techniques have been proposed to improve problem solvers based on Large Language Models (LLMs), yielding many disparate code bases. We unify these approaches by describing LLM-based agents as computational graphs. The nodes implement functions to process multimodal data or query LLMs, and the edges describe the information flow between operations. Graphs can be recursively combined into larger composite graphs representing hierarchies of inter-agent collaboration (where edges connect operations of different agents). Our novel automatic graph optimizers (1) refine node-level LLM prompts (node optimization) and (2) improve agent orchestration by changing graph connectivity (edge optimization). Experiments demonstrate that our framework can be used to efficiently develop, integrate, and automatically improve various LLM agents. The code can be found here.

1. Introduction

Interest in LLM-powered autonomous problem solvers or agents and their varied applications is continually rising (Wang et al., 2023; Xi et al., 2023). However, much work remains to be done to effectively incorporate these agents into a cohesive society and improve their structure automatically.

Early approaches zero-shot-prompted LLMs or prompted them with few-shot examples (Kojima et al., 2022; Brown et al., 2020). Recent methods prompt LLMs in a structured way, such as chain of thought (COT) (Wei et al., 2022), ReAct (Yao et al., 2022), tree of thought (TOT) (Yao et al., 2023), Reflexion (Shinn et al., 2023), and Graph of Thought (GOT) (Besta et al., 2023), to improve text-based reasoning. Single agent applications such as AutoGPT (Torantulino et al., 2023), BabyAGI (Nakajima, 2023), LangChain (Chase, 2022), and Llama-index (Liu, 2022) utilize LLMs for various functionalities, including tool usage, function calling, and embodied actions. In multi-agent frameworks (Zeng et al., 2022; Zhuge et al., 2023), several LLMs take on different roles (Li et al., 2023; Park et al., 2023; Qian et al., 2023; Wu et al., 2023), to communicate in natural language and collectively solve a given task. This approach often outperforms single agents, exploiting the specialization (Hong et al., 2023) of various LLM agents. Unfortunately, it also leads to increasingly different and disparate code bases that require a lot of human engineering to define prompting schemes and the workflow of agents.

In a "society of mind" (SOM) (Minsky, 1988; Zhuge et al., 2023), higher-level intelligence emerges from the combination of simpler and modular cognitive components. Inspired by SOMs, we describe language agent systems through graph representations. Language agents querying LLMs and utilizing external tools are modeled as computational graphs where each node is dedicated to a specific function, while the edges define a topology of how inputs are processed across nodes, mirroring the prompting schemes in prior studies. A swarm is defined as a composite graph, where each subgraph represents a collaborative agent. This creates a deeper hierarchy of intelligence. Agent graphs combine basic LLM operations (Kennedy, 2006; Nepusz & Vicsek, 2013), and swarm graphs contain subgraphs representing agents. Approaches such as COT (Wei et al., 2022), TOT (Yao et al., 2023), and Self-Consistency (Wang et al., 2022) can be represented by our graphs.

Our graph representation lends itself to optimization via prompting and evolutionary or reinforcement-learning tech-

★Equal Contribution ♦Project Engineer Lead

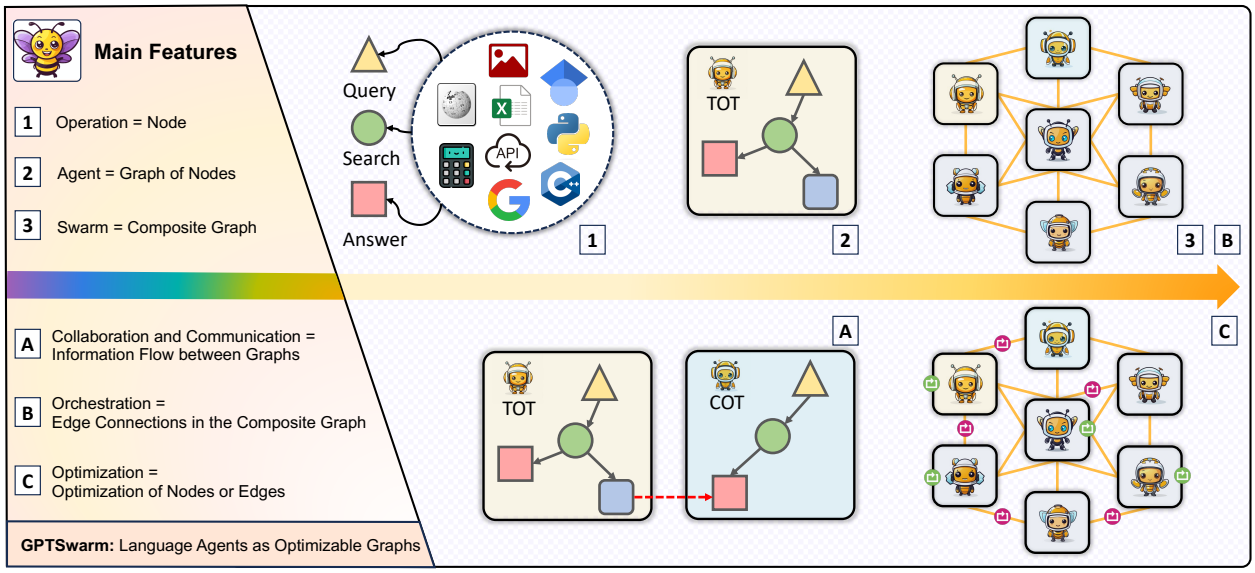


Figure 1. **GPTSwarm is a framework that represents agents as graphs.** In this framework, each node represents an operation (e.g., LLM inference or tool use). An agent is a graph composed of these nodes. An edge between two agent graphs characterizes a communication channel; each agent collaborates with others through different channels. When connected, multiple agents form a composite graph with a certain orchestration topology. This graph representation lends itself to optimization of nodes and edges via prompting and evolutionary or reinforcement learning techniques.

niques, so that agents can improve their communication (or orchestration) patterns. The graph connectivity (adjacency matrices) between agents can self-improve online as a task is being solved or its solution is transferred to another task.

As a proof-of-concept, we demonstrate how suboptimal agent organization can be overcome and how existing prompting techniques, such as Tree of Thought and Reflexion, can be automatically recombined by optimizing edges in a composite graph. Apart from edge optimization, our framework allows each node in the graph to self-improve by adapting its prompts based on previous input and task feedback.

Our contributions can be summarized as follows:

(1) We unify language agent systems by describing them as optimizable computational graphs.

(2) We introduce an open-source framework that allows for constructing arbitrary agent systems by recombining fundamental operations. We describe these engineering-level contributions in Appendix A.

(3) We develop optimization methods for nodes and edges, enabling automatic improvements of agent prompts and inter-agent orchestration.

(4) We validate our framework on various benchmarks including MMLU, Mini CrossWords, HumanEval, and GAIA, with an emphasis on the benefits of automatic graph optimization.

2. GPTSwarm

2.1. Language Agents as Graphs

Taking inspiration from the society of mind (SOM) (Minsky, 1988; Zhuge et al., 2023), we propose to organize intelligence within a modular and hierarchical framework. This framework consists of nodes, graphs, and composite graphs, with each component playing a specific role. A node represents a fundamental operation that includes, but is not limited to, LLM inference, tool use, function calls, and various embodied actions. An agent, conceptualized as a graph, consists of multiple nodes that form a coherent functional entity. A swarm, or composite graph, represents a complex system of agents where the collective capabilities of this system may exceed those of individual agents. Finally, the edges within an agent define its execution topology, while the edges between agents establish collaboration and communication among them.

2.2. Graph Definition

Single language agent as a graph. We model a language agent as a directed computational graph G, defined by a tuple (N, E, F, o), where N is a set of computational nodes, $E \subset N \times N$ is a set of directed edges, $F = \{f_n\}_{n \in N}$ is a set of computational routines and $o \in N$ is an output node. The set of predecessors of node n is denoted by $\text{pre}(n)$. In this paper, we focus on directed acyclic graphs (DAGs). Given an input x, a graph G iteratively executes its nodes according

to their topological order. Each node $n \in N$ receives as input x and the output z_n from its predecessor nodes. In this work, inputs and outputs are strings in natural language, but may take on other data types more generally. Node n applies the computational routine $f_n(z_n, x)$ and sends the output to its successor nodes. The graph output, denoted $\hat{y} = G(x)$, is the output $f_o(z_o, x)$ from the output node o. Note that in a DAG, some nodes will not have predecessors. For such nodes, the context z will be empty. This graph execution procedure is summarized in Algorithm 1.

Algorithm 1 Graph Execution

Require: Computational graph $G = (N, E, F, o)$, input x, empty context z for each node without predecessors.
  for n in TopologicalSort(N) **do**
    $z_n \leftarrow \{f_n(z_v, x) : v \in \text{pre}(n)\}$
  end for
Ensure: $f_o(z_o, x)$

In the context of language agents, for example, the input x may correspond to a question in natural language. Each node processes the input x and context information z from its predecessor nodes by applying a computational routine f. Examples of routines include LLM queries with input data from other agents, instructions to generate prompts for web searches that gather task-related information, or tool usage. Although our formalization specifies that the input x is given to each node, in practice, many routines might be designed to ignore the input and operate solely in the context provided by the predecessor nodes. Finally, the output provided by the output node corresponds to the answer to the input question or, more generally, to the solution of the input task.

Swarm of language agents as a composite graph. Given a set of K language agents, each represented by a computational graph $\{G_k = (N_k, E_k, F_k, o_k)\}_{k=1}^K$, one can compose these agents to achieve high performance in specific tasks. Let $N' = \cup_k N_k$ represent the union of the nodes of the agents, $E' = \cup_k E_k$ be the union of the edges of the agents, $F' = \cup_k F_k$ be the union of the computational routines of the agents, and $o' \in \cup_k \{o_k\}$ be the output node for the composite graph. Consider a selection of edges $\mathcal{E} \subset \cup_{i \neq j} N_i \times N_j$ that describe a set of connections between nodes from different agents. We define the composite graph representing the swarm of agents as $G_\mathcal{E} = (N', E_\mathcal{E}, F', o')$, where $E_\mathcal{E} = E' \cup \mathcal{E}$ is the union of the edges of the agents and the new edges connecting them. Composite graphs are restricted to DAGs. The composite graph $G_\mathcal{E}$ can be executed as described in Algorithm 1. In a swarm of language agents, the newly specified edges represent communication channels between agents. In the following sections, we explore how to optimize such a computational graph.

2.3. Edge Optimization

Given a task τ and its associated utility function u_τ that maps the candidate graphs to real numbers, we formulate an optimization problem about the choice of additional edges. The goal is to identify the edges that connect various language agents in a swarm, maximizing the utility. This process involves determining the most effective patterns of communication and information exchange among agents for the task at hand. We consider a set of potential edges $\{e_i\}_{i=1}^d = \mathcal{E}$, which leads to 2^d possible edge configurations, symbolized as $\mathcal{E} \in \{0, 1\}^d$. We further restrict the search space to only consider composite graphs that are DAGs. Formally, optimization of the composite graph of language agents is achieved by solving the problem $\max_\mathcal{E} u_\tau(G_\mathcal{E})$.

2.3.1. Problem Reformulation

DAG optimization through pruning of nodes and edges was already present in the first work on "deep learning" with deep feedforward networks (Ivakhnenko et al., 1965; Ivakhnenko, 1968). Due to the combinatorial complexity induced by DAGs, recent studies have increasingly focused on the continuous optimization approach (Vowels et al., 2022). This is particularly relevant in scenarios where most node executions require one or more queries to LLMs for moderate-scale applications. Moreover, the utility function is typically non-differentiable due to the tokenization of LLMs, and this remains true even when a differentiable DAG sampling technique is employed. Therefore, we reformulate our edge optimization as a continuous optimization problem. Instead of optimizing in a discrete space, our approach is to optimize over a continuum of probabilistic distributions, each representing a distribution over the feasible DAGs. Formally, rather than solving the maximum utility function $\arg\max_\mathcal{E} u_\tau(G_\mathcal{E})$, we propose solving

$$\arg\max_{\theta \in \Theta} \mathbb{E}_{G' \sim D_\theta}[u_\tau(G')], \tag{1}$$

where D_θ is a parameterized distribution and Θ represents a feasible set of real-valued parameters.

2.3.2. Solution Parameterization

A straightforward way to define a parameterized probabilistic distribution over DAGs with fixed nodes N and required edges E is to assign a real-valued parameter $\theta_i \in \mathbb{R}$ to each potential edge e_i. Let $\theta = [\theta_1; \theta_2; \ldots; \theta_d] \in [0, 1]^d$. The probability of $G' = G_\mathcal{E}$ for $G' \sim D_\theta$ is

$$\prod_{i=1}^{d} \begin{cases} \theta_i & \text{if } (N, E \cup (\{e_j\}_{j=1}^{i-1} \cap \mathcal{E}) \cup \{e_i\}) \text{ is a DAG,} \\ 0 & \text{otherwise.} \end{cases}$$

A sampling method that realizes this distribution is first to initialize a graph $G' \leftarrow (N, E)$. Then, iteratively sample

whether to include edge e_i in G' for all i's. If including e_i causes a cycle in current G', then the edge would not be included. Otherwise, add the edge to G' with probability θ_i.

2.3.3. Optimization Algorithm

To optimize the objective function (Equation (1)), we apply the REINFORCE algorithm (Williams, 1992) by applying a gradient ascent variant (e.g., Adam (Kingma & Ba, 2014)) with an unbiased gradient estimation:

$$\nabla_\theta \, \mathbb{E}_{G_\mathcal{E} \sim D_\theta} \left[u_\tau(G_\mathcal{E}) \right] \approx \frac{1}{M} \sum_{i=1}^{M} \hat{u}_\tau(G_i) \nabla_\theta \log(p_\theta(G_i)), \quad (2)$$

where $G_1, G_2, \ldots, G_N \sim D_\theta$ are mutually independent and $\hat{u}_\tau(G_i)$ is an independent unbiased estimate of $u_\tau(G_i)$ for all i and some $M \in \mathbb{N}$. Algorithm 2 describes the optimization algorithm with vanilla gradient ascent.

Algorithm 2 Edge Optimization with REINFORCE

Require: A parameterized probabilistic distribution over computation graphs D_θ, an unbiased utility estimator $\hat{u}_\tau(\cdot)$, and a learning rate α.
Initialize $\theta \in \mathbb{R}^d$.
while terminate condition not met **do**
  Sample $G_i \sim D_\theta$ for $i = 1, 2, \ldots, M$.
  Update $\theta \leftarrow \theta + \frac{\alpha}{M} \sum_{i=1}^{M} \hat{u}_\tau(G_i) \nabla_\theta \log(p_\theta(G_i))$.
end while

2.4. Node Optimization

In our framework, each node implements a fundamental operation, such as querying an LLM, using a tool, calling an API, etc. In a language agent, most of these operations involve prompting an LLM once or several times. Optimizing the prompts of these nodes is crucial for improving the system's overall performance.

Unlike many other works on prompt optimization, which optimize a single global prompt (e.g., Yang et al., 2023; Pryzant et al., 2023; Deng et al., 2022), our node optimization problem naturally involves several operations where each of them consists of a node-level prompt. In our case, the optimization problem is more complex due to prompts affecting how other prompts operate on connected nodes. At the same time, our graph representation leads to a separation of concerns where each node has a specific purpose with its own associated prompt. Due to this separation of concerns, we hypothesize that, for every optimization step, it is sufficient to update each node-level prompt individually, assuming that all other prompts are fixed.

Consider a parameterized computational graph $G^P = (N, E, F^P, o)$, where $F^P = \{f_n^{p_n}\}$ are computational routines, each parameterized by a prompt p_n to be optimized for all $n \in N$. To enable effective node optimization, we also require a natural language description of the intended function for each routine $f_n^{p_n} \in F$ denoted by d_n. For example, a suitable description for a node designed to write Python programs would be "a Python code generator". Here, existing prompt optimization methods, such as OPRO (Yang et al., 2023), can be described as a function I that iteratively maps a prompt, a function description, and a set of node input-output pairs (which may include annotations such as a quality measure for each pair) to an improved prompt. For example, I could take a prompt such as "generate Python code", a description "a Python code generator", and an input-output pair "Input: evaluate two divided by one as an integer. Output: 2 / 1", where the output yields 1.0 as the result of execution. A prompt optimization method would return an improved prompt "generate Python code and pay attention to data types".

Formally, our method begins by initializing an empty history set, denoted h_n, one for each node $n \in N$. The process then proceeds iteratively: first, the graph $G^P(x)$ is executed using a randomly sampled input x following Algorithm 1. Subsequently, for each node, a tuple consisting of the input to the node (z_n, x), where z_n is the context vector that includes the outputs of the predecessor nodes, and the node's own output $f_n^{p_n}(z_n, x)$, is added to the node's history h_n. The final step involves updating the node prompts. This is done by applying I to the node's updated history, its current prompt, and its function description, resulting in an improved prompt $I(h_n, p_n, d_n)$. This iterative process, described in Algorithm 3, continuously improves the operations of the nodes in the entire graph.

Algorithm 3 Node Optimization

Require: A parameterized graph $G^P = (N, E, F^P, o)$, natural language function descriptions $D = \{d_n\}_{n \in N}$, and a distribution of inputs D_X.
Initialize p_n for all $n \in N$.
Initialize $h_n \leftarrow \emptyset$ for all $n \in N$.
while terminate condition not met **do**
  Sample input $x \sim D_X$.
  $y \leftarrow G^P(x)$ following Algorithm 1.
  $h_n \leftarrow h_n \cup \{((z_n, x), f_n^{p_n}(z_n, x))\}$ for all $n \in N$.
  $p_n \leftarrow I(h_n, p_n, d_n)$, for all $n \in N$.
end while

2.5. General Applicability

Frameworks such as AutoGPT (Torantulino et al., 2023) and LangChain (Chase, 2022) have set a standard for flexibility and reusability in various language-based tasks. Our framework, GPTSwarm, introduces a graph-based design of agents and swarms. This design further simplifies the reuse of modular components (nodes & agents) and the integration of such modules. For instance, GPTSwarm supports 41 types of file analysis, web search (e.g., Google Search),

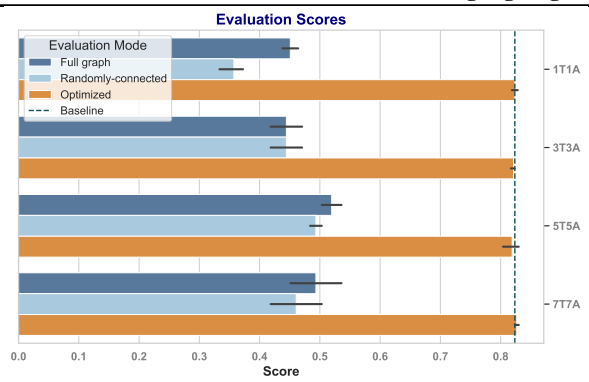


Figure 2. **Score recovery through edge optimization.** "T" denotes truthful and "A" adversarial agents, e.g., a 3T3A swarm has 3 of each. Ablation studies include a "full graph" and random graphs sampled according to distribution $D_{0.5}$. The dashed line corresponds to the direct answer baseline.

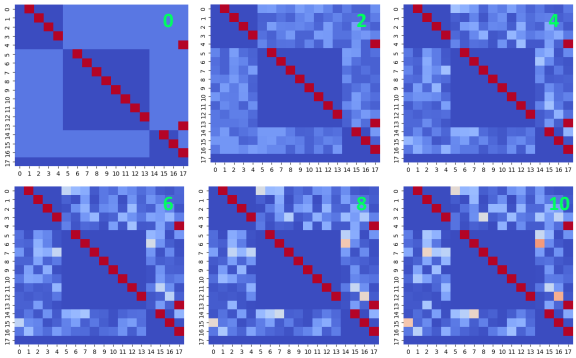


Figure 3. **Visualizing the evolution of the probability distribution during optimization in adjacency-like matrices.** In this figure, we show the probability parameters (one corresponds to an edge) in an adjacency-like matrix for iterations 0, 2, 4, 6, 8, and 10 of optimizing the objective for the Mini Crosswords task. We observe that the parameters first change chaotically. However, after iteration 6, the parameters change almost monotonically.

and index-based memory. By offering a wide range of modules, our framework makes it easier to implement various language agent systems. See Section 3.4 for further details.

3. Experiments

3.1. MMLU

Motivation: In our first experiments, we demonstrate that edge optimization effectively filters adversarial agents from a swarm, mirroring a scenario in multi-agent systems where some agents are detrimental rather than beneficial. Ideally, optimization would automatically eliminate harmful agents. We conducted this experiment using the 4-choice MMLU general knowledge question answering dataset, as detailed by Hendrycks et al. (2021b;a). Our setup involves initializing a swarm consisting of k Input-Output (IO) agents and k adversarial agents, following the terminology by Besta et al. (2023). The IO agents query an LLM and relay the LLM's responses directly. In contrast, adversarial agents are deliberately programmed to manipulate the LLM to provide incorrect answers. The collective decision on the final answer is made through majority voting, bypassing any additional LLM query that could introduce corrective intelligence against adversarial influence. We benchmark the performance of a single IO agent as our baseline. An effective optimization is expected to elevate the swarm's performance on the MMLU dataset to match this baseline level.

Analysis: In Figure 2, we present the comparative performance scores of different swarm configurations: the baseline, the graph formed by sequentially including edges that do not create loops (denoted as the 'full graph'), a randomly connected swarm sampled from the initial distribution D_θ with $\theta = 0.5$, and the optimized swarm. These scores are based on an evaluation of 10% of the MMLU validation set, which consists of 153 questions. The edge optimization process uses REINFORCE (Alg. 2) over 200 iterations. Each iteration assesses four graph samples, each on a specific problem sourced from the MMLU dev set. Throughout these experiments, we employed GPT-4-Turbo, with the token sampling temperature fixed at 0.2. Figure 9 demonstrates how the optimized swarm score aligns asymptotically with that of the baseline. Table 3 compiles the key statistics and findings of these experiments. The findings indicate that our approach successfully safeguards a swarm against harmful adversaries.

3.2. Mini Crosswords

Motivation: This section investigates to what extent edge optimization can improve the performance of standard agents from the literature. We conduct our evaluation on the Mini Crosswords dataset[1]. A subset of 20 problems is used to optimize and evaluate our methods, in agreement with previous studies (Yao et al., 2023; Sel et al., 2023). The choice of Mini Crosswords for this analysis is strategic, as it highlights how the algorithmic structure of the solvers, such as the tree search employed by TOT, significantly influences their performance (Yao et al., 2023). Our hypothesis is that edge connections can meaningfully determine the algorithmic structure. Through edge optimization, we anticipate the automatic discovery and implementation of high-performance algorithms.

Analysis: In our experiments, we explore the performance of swarms of three distinct agents. The first agent, which implements the TOT approach, iteratively branches over candidate solutions provided by an LLM, processing one word at each step. The second agent is based on the Reflexion method (Shinn et al., 2023). This agent first proposes a solution through a greedy approach and then creates an al-

[1] https://www.goobix.com/crosswords

ternative solution informed by feedback from a critic, which is based on an LLM analysis of the initial solution. The third agent we examine is a Chain of Thought (COT) agent consisting of three nodes. Each node within the COT performs an internal brute-force search to select the optimal subset of candidates generated by the LLM for the current state, scored by the LLM. The agent or swarm then returns all the solutions generated by their output node.

For the utility function, we choose the best of all the graph-returned solutions according to the number of words correctly filled (i.e., best state word accuracy) as done by Yao et al. (2023). During the evaluation, we average over 20 graph samples from the graph distribution, each evaluated on a unique question randomly sampled from the dataset.

We optimize our composite graph of agents using the REINFORCE (Alg. 2), setting the initial edge probability at $\theta = 10\%$ and the learning rate at $\alpha = 0.4$. For each iteration, the gradient is estimated according to equation (2) by sampling $M = 20$ graphs, each evaluated on a crossword problem. For cost-effectiveness, we optimize and evaluate graphs with the GPT-3.5-Turbo language model, where the temperature is set to zero. Figure 3 visualizes the evolution of probability parameters in the form of adjacency-like matrices over ten iterations. We observe that the parameters first change chaotically. However, after iteration 6, the parameters change almost monotonically.

We follow Alg. 2 to optimize the objective in Equation 1, achieving an average accuracy of $0.575(\pm 0.0275)$ after ten iterations (we report the average over 3 runs and the standard error). This surpasses the initial distribution's score of $0.465(\pm 0.0509)$. Furthermore, we evaluate the best-of-three performance by aggregating the top results from each problem across the three agents, which yields an accuracy of $0.320(\pm 0.0415)$.

Note that denser graphs are likely to require more computational resources. To verify that the improvements of our method are not solely due to an increase in the number of edges and therefore a larger computational budget, we compare it with a distribution with all parameters set to $\theta = 12.5\%$. This value reflects the average number of edges in the learned distribution, determined by sampling 1000 graphs from each run's resulting distribution. The expected number of edges for both the learned distribution and the 0.125 parameter-valued distribution are approximately $29.71(\pm 1.74)$ and $29.68(\pm 0.10)$, respectively. Despite the similarity in the edge count, the 0.125 parameter-valued distribution achieves an accuracy of $0.510(\pm 0.0552)$, allowing us to attribute the improvements to factors beyond the mere edge density.

Furthermore, we evaluate one of our final optimized distributions (randomly selected) with the GPT-4-Turbo language model[2]. It achieves an accuracy of $0.800(\pm 0.0616)$, significantly exceeding the previous state-of-the-art performance of 0.675 (Yao et al., 2023). All reported metrics are summarized in Figure 4.

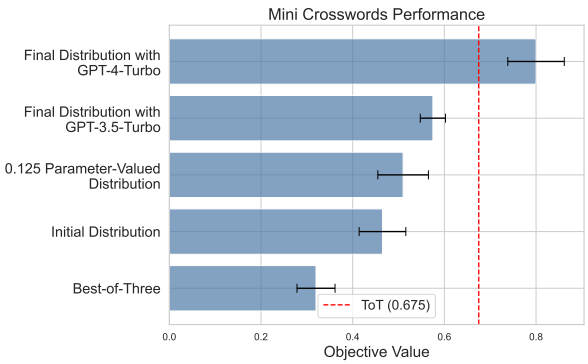


Figure 4. **Edge optimization on the Mini Crosswords dataset improves over standard methods** The baseline methods are evaluated with GPT-3.5-Turbo. The optimized final distribution outperforms several baselines. When evaluating the already optimized edge distribution with GPT-4-Turbo, we achieve better results compared to the previous state-of-the-art method (Tree of Thought).

3.3. HumanEval

Motivation: In the previous experiments on MMLU (math problems) and Mini Crosswords (open-ended puzzles) we have validated the utility of optimizing graph edges. In this section, we test the HumanEval dataset (Chen et al., 2021), which is known to be sensitive to prompt design. Previous research involved manually crafting prompts (Shinn et al., 2023; Hong et al., 2023) and achieved impressive performance. In contrast, here, we explore how node-based optimization can simplify this process. We employ an online learning setting, continuously optimizing without restarting.

Analysis: In this section, we optimize the prompts of a ReAct-style (Yao et al., 2022) agent. The agent first generates a Python program in response to a given question. If the generated program passes all test cases included in the problem statement, then the program is returned. Otherwise, the agent regenerates a program based on the execution feedback. We experiment with two node-level optimization strategies: (1) modifying the instruction prompts and (2) adding demonstration examples to the prompts.

Altering the instruction prompts rarely improved the results, possibly due to the limited sophistication of our current meta-prompts compared to OPRO (Yang et al., 2023) and PromptBreeder (Fernando et al., 2023). However, selectively incorporating previously executed input-output pairs,

[2]We are limiting our evaluation to a single graph distribution due to the high cost associated with API calls for this type of evaluation.

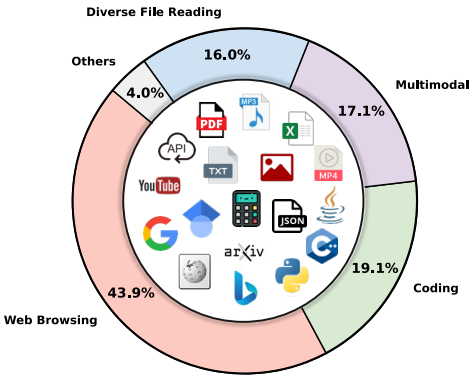


Figure 5. **Solving a wide range of tasks requires many different tools.** The GAIA benchmark (Mialon et al., 2023) tests for many of these capabilities by including questions that require several of these tools for successful completion.

Table 1. **Performance on the GAIA Benchmark (Mialon et al., 2023).** Using our framework, we demonstrate significant improvements across several levels of difficulty. The 'GPT-4 with plugins' baseline is less significant since it involves the manual selection of the appropriate tools per question. We report the mean and standard deviation across 5 runs.

Method	Level 1	Level 2	Level 3	Avg.
GPT-3.5	7.55	4.65	0	4.85
GPT-4	15.09	2.33	0	6.06
GPT-4-Turbo	20.75	5.81	0	9.70
AutoGPT	13.21	0	3.85	4.85
GPTSwarm	**30.56**$_{\pm 3.25}$	**20.93**$_{\pm 1.27}$	**3.85**$_{\pm 2.43}$	**18.45**
Improvement	47.3%↑	260.2%↑	0.0%	90.2%↑
GPT4 with Plugins*	30.30	9.70	0	14.6

which correspond to successful program generations of the graph, as demonstration examples in the context of the nodes, increases the pass@1 accuracy from 77% to 89%. We select input-output pairs by assessing their effectiveness as demonstration examples, particularly in improving the node operation applied to the node's ten most recent inputs. To evaluate a node operation, we determine if the generated program successfully solves the unit tests provided in the input problem statement. For more details on the node optimizer, please refer to Appendix D.3.1. We hypothesize that the performance could be further improved with more sophisticated (meta-)prompts.

3.4. GAIA

Motivation: GAIA is a benchmark specifically designed for testing the generality of AI assistants focusing on real-world questions (Mialon et al., 2023). Abilities required to answer GAIA questions include reasoning, multi-modality processing, web browsing, and other tool use. Although conceptually straightforward for humans, these questions present significant challenges for current AI systems.

Using this benchmark, we evaluate the general applicability of our framework. We construct swarms with multiple agents of the same type and employ self-consistency (a prompt-based majority vote) for the final decision (Wang et al., 2022). We also experimented with adding different types of agents to the swarm and using prompt-based best answer selection. The results indicate that prompt-based self-consistency yields the best performance. Note that these experiments are meant to demonstrate the generic capabilities of our modular framework and include neither edge-based nor node-level optimization, which is left for future work.

Analysis: Table 1 shows the results of our swarm with seven TOT agents and the self-consistency strategy for the final decision. We compare the performance of the GPT-Series (Achiam et al., 2023) with plugins and AutoGPT (Torantulino et al., 2023) performance as reported by Mialon et al. (2023). Our methods significantly outperform these baselines.

Table 2 presents a more comprehensive set of results. We experiment with varying numbers of agents and different node operations, such as different tool uses. Our observations indicate that the time requirement of a swarm grows approximately linearly with the number of agents. Despite the increased computational time, incorporating more agents notably improves the overall performance of the system. We also found that a greater variety of node operations leads to better performance. As illustrated in Figure 5, web browsing is required for 43.9% of the tasks. Our current implementation accesses the Internet by only downloading materials directly from the URLs provided in the problem statement or querying a Google search[3] without further website navigation. Therefore, we believe that enhancing web capabilities would further increase performance significantly.

4. Related Work

4.1. LLM-based Autonomous Agents

Current works on LLM-based autonomous agents or language agents vary in focus. Methods such as Chain of Thought (Wei et al., 2022), ReAct (Yao et al., 2022), Reflexion (Shinn et al., 2023), and Tree of Thought (ToT) improve prompt strategies and structure to improve reasoning capabilities. Single LLM agent frameworks such as AutoGPT (Gravitas, 2023), LangChain (Chase, 2022), LlamaIndex (Liu, 2022), and XAgent (XAgent Team, 2023) showcase problem solving through various external functions and tools. In the space of LLM-based multiagent systems (Xie et al., 2023; Chen et al., 2023a;b), NLSOMs (Zhuge et al., 2023) employ various social structures for task-specific applications (inspired by SOMs (Minsky, 1988)), without

[3]We use SearchAPI (https://serpapi.com) in the experiments

Table 2. **Ablations on the GAIA benchmark (Level 1 validation set) (Mialon et al., 2023).** DA = DirectAnswer, GQ = GenerateQuery, WS = WebSearch, FA = FileAnalyzer, CA = ComebineAnswer. '✓' indicates the presence of a specific feature in the corresponding framework, '✗' its absence. Each type of experiment is run five times to record the mean, standard deviation, and best run (marked as Best). Self-Consistency describes prompt-based self-consistency (Wang et al., 2022); Choose "Best" refers to the LLM's favorite answer among the different agents' answers. All agents and swarms are implemented using our GPTSwarm framework.

Agent or Swarm	DA	GQ	WS	FA	CA	Decision Strategy	Accuracy	Best	Duration (s)
(A) Agent: IO	✓	✗	✗	✗	✗	N/A	16.60±3.02	20.75%	∼13.37
(B) Agent: COT_{web}	✗	✓	✓	✗	✓	N/A	18.87±2.67	22.64%	∼60.90
(C) Agent: COT_{FA}	✗	✓	✗	✓	✓	N/A	25.28±3.50	30.18%	∼56.42
(D) Agent: TOT	✗	✓	✓	✓	✓	N/A	25.66±3.50	30.18%	∼71.31
(E) $\text{Swarm}_{(3\times\text{IO})}$	✓	✗	✗	✗	✗	Choose "Best"	15.85±0.92	18.87%	∼45.65
(F) $\text{Swarm}_{(3\times\text{COT})}$	✗	✓	✓	✗	✓	Choose "Best"	27.17±3.29	32.08%	∼152.89
(G) $\text{Swarm}_{(3\times\text{TOT})}$	✗	✓	✓	✓	✓	Choose "Best"	30.18±4.30	35.85%	∼198.50
(H) $\text{Swarm}_{(3\times\text{IO})}$	✓	✗	✗	✗	✗	Self-Consistency	18.11±3.07	22.64%	∼45.70
(I) $\text{Swarm}_{(3\times\text{COT})}$	✗	✓	✓	✗	✓	Self-Consistency	27.17±4.06	32.08%	∼150.26
(J) $\text{Swarm}_{(3\times\text{TOT})}$	✗	✓	✓	✓	✓	Self-Consistency	28.30±3.38	32.08%	∼181.15
(K) $\text{Swarm}_{(5\times\text{TOT})}$	✗	✓	✓	✓	✓	Self-Consistency	29.06±2.56	32.08%	∼291.07
(L) $\text{Swarm}_{(7\times\text{TOT})}$	✗	✓	✓	✓	✓	Self-Consistency	30.56±3.25	35.85%	∼414.89
(M) Human	-	-	-	-	-	-	94%	-	∼422.26

exploring optimization over the social structure of agents. CAMEL (Li et al., 2023), Generalist Agents (Park et al., 2023), ChatDev (Qian et al., 2023), and AutoGen (Wu et al., 2023) focus on role-play communication, but struggle with hallucinations. MetaGPT (Hong et al., 2023) introduces standard operating procedures for better role definition and communication, making the collaboration between agents more effective. In contrast to these frameworks, we automatically optimize nodes and edges in a self-organizing society of agents.

4.2. Language Agents with Graphs

Besta et al. (2023) introduced LLM-based problem-solving with graphs; however, the approach only encompasses LLM prompting schemes without modeling other fundamental capabilities of language agents, such as use of external tools. LangGraph (langchain ai, 2024), on the other hand, is a concurrent open-source framework that focuses on building multi-actor state LLM applications through possibly cyclical operations. However, its practical applicability has not yet been systematically studied. Unlike previous studies, our approach emphasizes the development of hierarchical intelligence, as discussed by Minsky (1988) and Kennedy (2006), through the construction of agent graphs and the composition of multiple graphs into swarms. Crucially, the graph representation facilitates automatic optimization on two levels. First, at the node level, since the majority of nodes in the graph involve prompting an LLM, prompt optimization methods can be employed. Second, at the edge level, we demonstrate the application of the REINFORCE algorithm (Williams, 1992) to optimize the potential connections between nodes.

4.3. Optimizing LLM Inference and Self-Improvement

Much of deep learning research is concerned with tuning the learning algorithms, architectures, hyper-parameters, and other aspects of the learning pipeline (Schmidhuber, 2015; Yan et al., 2015). Meta-learning attempts to automate large parts of that process (Schmidhuber, 1987; Elsken et al., 2019; Kirsch & Schmidhuber, 2021). Similarly, recently, a lot of research and engineering has gone into the prompting and structuring of LLM inference to make better use of LLMs and build better agents. Due to the ability of LLMs to learn in context (Brown et al., 2020; Kirsch et al., 2022), one can view this process as configuring learning algorithms. The optimization of the inference structure and the prompts can then be viewed as meta-learning in LLMs.

In the realm of prompt optimization, OPRO (Yang et al., 2023) generates better prompts through iterative LLM queries using prior solutions and their performance. PromptBreeder (Fernando et al., 2023) implements a mechanism that evolves and self-improves task-specific and meta-prompts through mutation and LLM prompting. Related to these works, we self-improve future prompts by prompting LLMs. Similarly to our work, DSPy (Khattab et al., 2023) implements LLM pipelines as computational graphs with modular LLM queries as nodes, parameterized by prompts and neural network weights. It proposes a two-stage process to optimize the parameters of these nodes. Initially, it generates a set of candidate solutions for each node. Subsequently, it optimizes across the Cartesian product of these candidate solution sets, aiming to identify an effective combination of parameters for the entire graph. To address the combinatorial optimization challenge raised in DSPy, we propose an iterative optimization process. By virtue of decomposing a

solution into nodes with expected functions, at each iteration, we improve each node individually, conditioned on the execution history of the graph with the current prompts of each node.

Regarding the optimization of the inference structure, DyLAN (Liu et al., 2023) uses a fixed heuristic to improve the collaboration of LLM agents by selecting agents and determining the number of communication rounds. In line with previous ideas on self-referential learning (Schmidhuber, 1993; Irie et al., 2022; Kirsch & Schmidhuber, 2022), STOP (Zelikman et al., 2023) optimizes both the prompts and the inference structure together by introducing an initial improver program that is applied to itself to iteratively improve its performance. In our work, we optimize the inference structure by employing RL techniques applied to the potential edges of a given graph.

5. Conclusion

This paper introduces GPTSwarm, an open-source framework that constructs language agents from graphs and agent societies from graph compositions. This approach allows for the easy implementation of existing methods from basic node operations and enables automatic optimization of the graph in the form of node-level improvement and edge-level REINFORCE optimization. Our experiments demonstrate the advantages of our language agent graphs and automatic optimization on several benchmarks.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. The societal consequences of our work are multifaceted. On the one hand, it could lead to significant advancements in the efficiency and effectiveness of machine learning systems. On the other hand, the increased capability and automation of LLM agents might raise ethical and employment concerns. As AI systems become more autonomous and powerful, it is crucial to consider their impact on job displacement and the importance of implementing safeguards to prevent biased or unethical AI behaviors. Furthermore, the potential for misuse of advanced AI technologies requires rigorous oversight and the development of ethical guidelines to ensure that these technologies are used responsibly and for the benefit of society as a whole.

Author Contributions

Mingchen initiated the project and conceived the initial idea, led the development of the codebase, conducted GAIA & HumanEval experiments, drafted the initial manuscript, and created most of the visualizations. Wenyi discussed the core ideas with Mingchen, contributed to the codebase, conducted Mini CrossWords & HumanEval experiments, and drafted the initial manuscript. Louis reviewed and polished the paper, extensively rewrote the introduction, and coordinated team meetings. Francesco reviewed and polished the paper, significantly revising the methods section. Furthermore, Louis and Francesco discussed and formalized various techniques for graph optimization. As the senior engineering lead, Dmitrii advised and made significant revisions to the codebase, conducted the MMLU experiments, and contributed to the visualizations. Juergen, as mentor and advisor, offered guidance and support throughout the project's progression.

References

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

Besta, M., Blach, N., Kubicek, A., Gerstenberger, R., Gianinazzi, L., Gajda, J., Lehmann, T., Podstawski, M., Niewiadomski, H., Nyczyk, P., et al. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687*, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chase, H. LangChain. https://github.com/hwchase17/langchain, 2022.

Chen, G., Dong, S., Shu, Y., Zhang, G., Sesay, J., Karlsson, B. F., Fu, J., and Shi, Y. Autoagents: A framework for automatic agent generation. *arXiv preprint arXiv:2309.17288*, 2023a.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Chen, W., Su, Y., Zuo, J., Yang, C., Yuan, C., Qian, C., Chan, C.-M., Qin, Y., Lu, Y., Xie, R., et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*, 2023b.

Deng, M., Wang, J., Hsieh, C.-P., Wang, Y., Guo, H., Shu, T., Song, M., Xing, E. P., and Hu, Z. Rlprompt: Optimizing discrete text prompts with reinforcement learning. *arXiv preprint arXiv:2205.12548*, 2022.

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

Fernando, C., Banarse, D., Michalewski, H., Osindero, S., and Rocktäschel, T. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

Gravitas, S. Auto-gpt. *GitHub repository*, 2023.

Hendrycks, D., Burns, C., Basart, S., Critch, A., Li, J., Song, D., and Steinhardt, J. Aligning ai with shared human values. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021a.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021b.

Hong, S., Zheng, X., Chen, J., Cheng, Y., Zhang, C., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

Irie, K., Schlag, I., Csordás, R., and Schmidhuber, J. A modern self-referential weight matrix that learns to modify itself. In *International Conference on Machine Learning*, pp. 9660–9677. PMLR, 2022.

Ivakhnenko, A., Lapa, V., and ENGINEERING., P. U. L. I. S. O. E. *Cybernetic Predicting Devices*. JPRS 37, 803. Joint Publications Research Service [available from the Clearinghouse for Federal Scientific and Technical Information], 1965. URL https://books.google.com.sa/books?id=l38DHQAACAAJ.

Ivakhnenko, A. G. The group method of data handling, a rival of the method of stochastic approximation. *Soviet Automatic Control*, 13(3):43–55, 1968.

Kennedy, J. Swarm intelligence. In *Handbook of nature-inspired and innovative computing: integrating classical models with emerging technologies*, pp. 187–219. Springer, 2006.

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kirsch, L. and Schmidhuber, J. Meta learning backpropagation and improving it. *Advances in Neural Information Processing Systems*, 34:14122–14134, 2021.

Kirsch, L. and Schmidhuber, J. Eliminating meta optimization through self-referential meta learning. *arXiv preprint arXiv:2212.14392 and First Conference on Automated Machine Learning (Workshop)*, 2022.

Kirsch, L., Harrison, J., Sohl-Dickstein, J., and Metz, L. General-purpose in-context learning by meta-learning transformers. *arXiv preprint arXiv:2212.04458*, 2022.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

langchain ai. LangGraph. https://github.com/langchain-ai/langgraph, 2024.

Li, G., Hammoud, H. A. A. K., Itani, H., Khizbullin, D., and Ghanem, B. Camel: Communicative agents for" mind" exploration of large scale language model society. *arXiv preprint arXiv:2303.17760*, 2023.

Liu, J. LlamaIndex, 11 2022. URL https://github.com/jerryjliu/llama_index.

Liu, Z., Zhang, Y., Li, P., Liu, Y., and Yang, D. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*, 2023.

Mialon, G., Fourrier, C., Swift, C., Wolf, T., LeCun, Y., and Scialom, T. Gaia: a benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*, 2023.

Minsky, M. *Society of mind*. Simon and Schuster, 1988.

Nakajima, Y. Babyagi. *Python. https://github.com/yoheinakajima/babyagi*, 2023.

Nepusz, T. and Vicsek, T. Hierarchical self-organization of non-cooperating individuals. *Plos one*, 8(12):e81449, 2013.

Park, J. S., O'Brien, J., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023.

Pryzant, R., Iter, D., Li, J., Lee, Y. T., Zhu, C., and Zeng, M. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*, 2023.

Qian, C., Cong, X., Yang, C., Chen, W., Su, Y., Xu, J., Liu, Z., and Sun, M. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023.

Schmidhuber, J. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

Schmidhuber, J. A 'self-referential'weight matrix. In *ICANN'93: Proceedings of the International Conference on Artificial Neural Networks Amsterdam, The Netherlands 13–16 September 1993 3*, pp. 446–450. Springer, 1993.

Schmidhuber, J. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Sel, B., Al-Tawaha, A., Khattar, V., Wang, L., Jia, R., and Jin, M. Algorithm of thoughts: Enhancing exploration of ideas in large language models. *arXiv preprint arXiv:2308.10379*, 2023.

Shinn, N., Cassano, F., Labash, B., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 14, 2023.

Torantulino et al. Auto-gpt. https://github.com/Significant-Gravitas/Auto-GPT, 2023.

Vowels, M. J., Camgoz, N. C., and Bowden, R. D'ya like dags? a survey on structure learning and causal discovery. *ACM Computing Surveys*, 55(4):1–36, 2022.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., et al. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*, 2023.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Zhang, S., Zhu, E., Li, B., Jiang, L., Zhang, X., and Wang, C. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.

XAgent Team. Xagent: An autonomous agent for complex task solving, 2023.

Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., Zhang, M., Wang, J., Jin, S., Zhou, E., et al. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.

Xie, T., Zhou, F., Cheng, Z., Shi, P., Weng, L., Liu, Y., Hua, T. J., Zhao, J., Liu, Q., Liu, C., et al. Openagents: An open platform for language agents in the wild. *arXiv preprint arXiv:2310.10634*, 2023.

Yan, L. C., Yoshua, B., and Geoffrey, H. Deep learning. *nature*, 521(7553):436–444, 2015.

Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., and Chen, X. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. R. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=5Xc1ecxO1h.

Zelikman, E., Lorch, E., Mackey, L., and Kalai, A. T. Self-taught optimizer (stop): Recursively self-improving code generation. *arXiv preprint arXiv:2310.02304*, 2023.

Zeng, A., Attarian, M., Ichter, B., Choromanski, K., Wong, A., Welker, S., Tombari, F., Purohit, A., Ryoo, M., Sindhwani, V., et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022.

Zhuge, M., Liu, H., Faccio, F., Ashley, D. R., Csordás, R., Gopalakrishnan, A., Hamdi, A., Hammoud, H. A. A. K., Herrmann, V., Irie, K., et al. Mindstorms in natural language-based societies of mind. *arXiv preprint arXiv:2305.17066*, 2023.

A. The GPTSwarm Framework

A.1. The Vision

Many recent language agents are described as compositions of components of different functionalities (Wang et al., 2023). A popular tweet states: "agent = LLM + memory + planning skills + tool use."[4] Such additive formulations highlight individual components, but fail to address the essential aspect of component *integration*. GPTswarm's computational graph formulation, however, precisely focuses on *integration* through edge optimization, to learn improved recommendations of agent orchestration and precise agent routing. This will become increasingly relevant as swarm size increases to millions or billions of agents.

A.2. Class Diagram

The GPTSwarm framework is developed using Python and PyTorch. Its class diagram is illustrated in Figure 6.

On the graph level, Node, Graph, and CompositeGraph are directly implemented as classes. Graph edges are implicitly stored as an adjacency list within each Node. Functionally, the framework distinguishes between Agents and Operations through various classes, such as DirectAnswer and WebSearch for operations, and IO and TOT for agents. To encapsulate the abstraction of an external LLM, we introduce an interface named after it. The primary implementation of this interface is a lightweight wrapper around the OpenAI API. To facilitate dataset integration for optimization and evaluation, we provide implementations for two interfaces: Dataset and PromptSet. The Dataset interface is designed to load benchmark datasets like GAIA and MMLU, while the PromptSet customizes node behavior for a specific Dataset. The Evaluator class manages the optimization processes for both edges and nodes.

The framework is highly customizable, allowing users to add more LLM backends, Dataset and PromptSet combinations, Agents, and Nodes as needed. Additionally, the framework makes extensive use of asynchronous computations for task parallelism, leveraging Python's async-await syntax.

[4] https://twitter.com/lilianweng/status/1673535600690102273

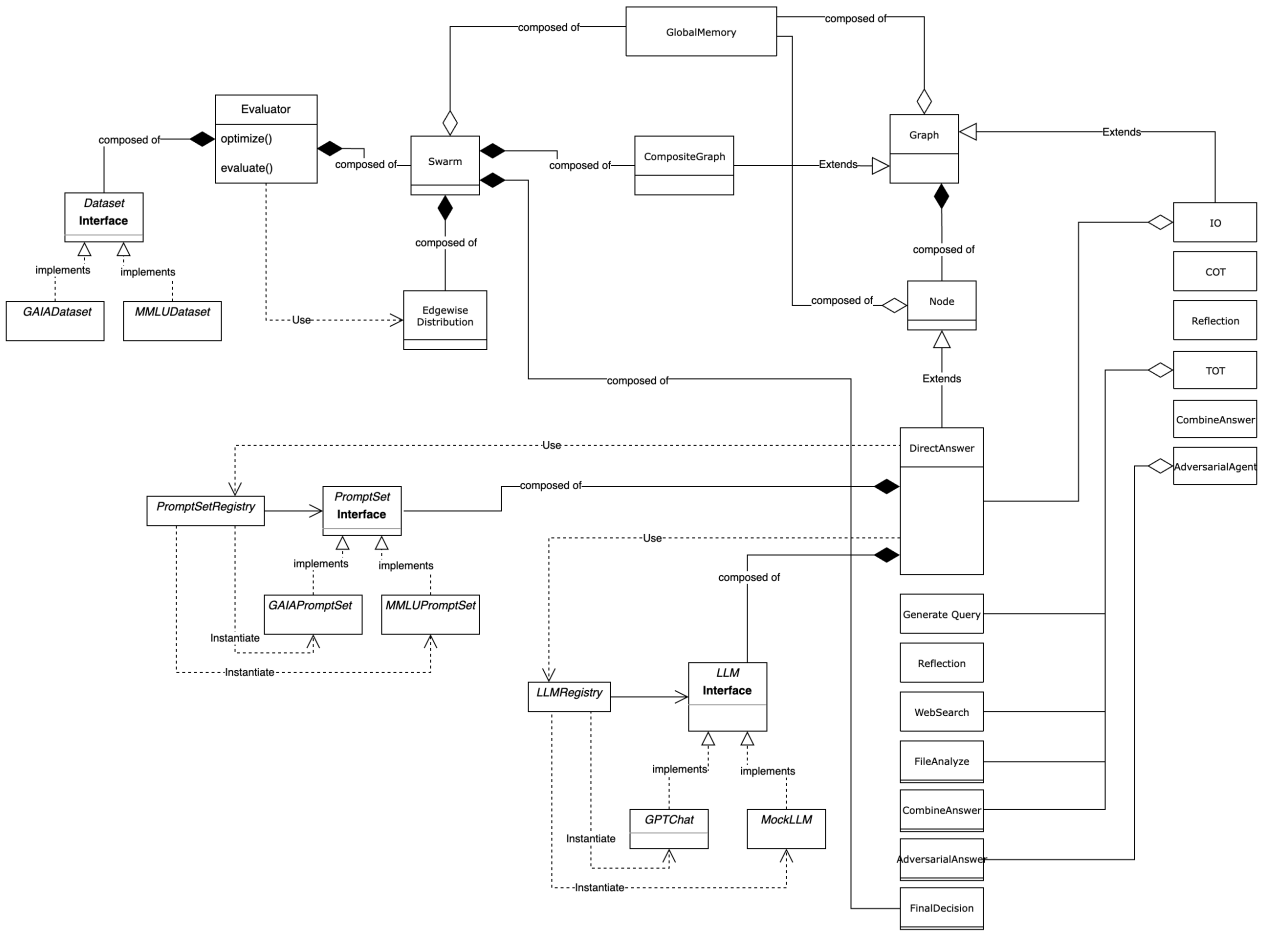


Figure 6. The cLass diagram of the GPTSwarm framework.

B. Swarm examples

To facilitate understanding of the concepts presented in this study, we are showing a simple example of a swarm consisting of 3 agents: Tree-of-Thought, Input-Output, and Decision Agents in Figure 7.

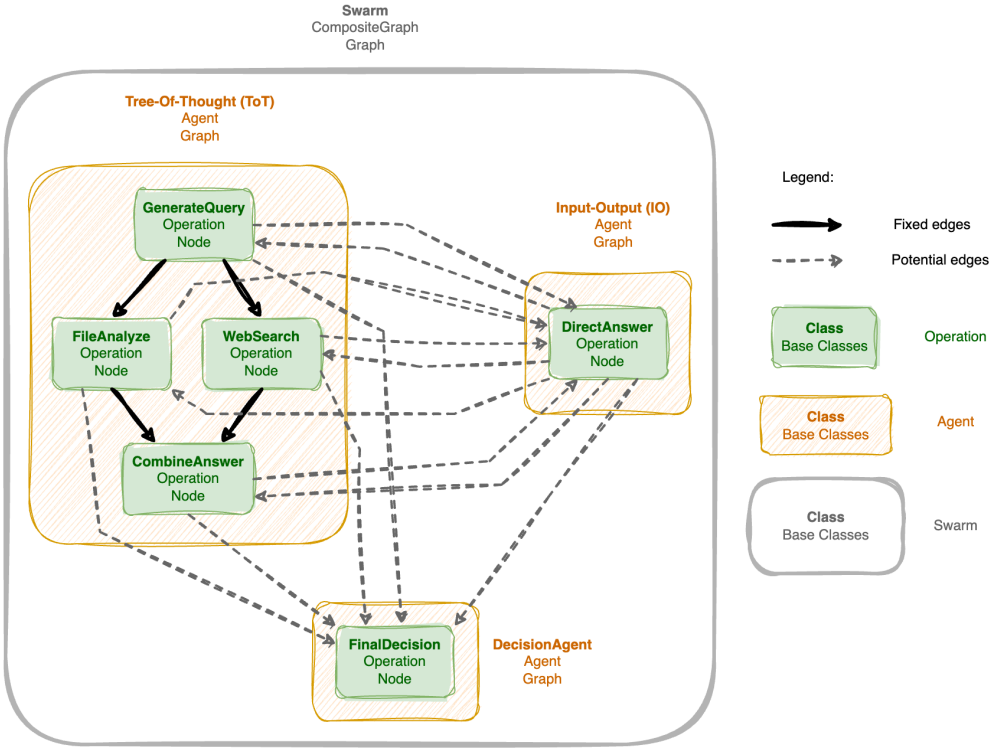


Figure 7. A simple example of a swarm consisting of one Tree-of-Thought, one Input-Output, and the Decision agent.

C. More Visualizations

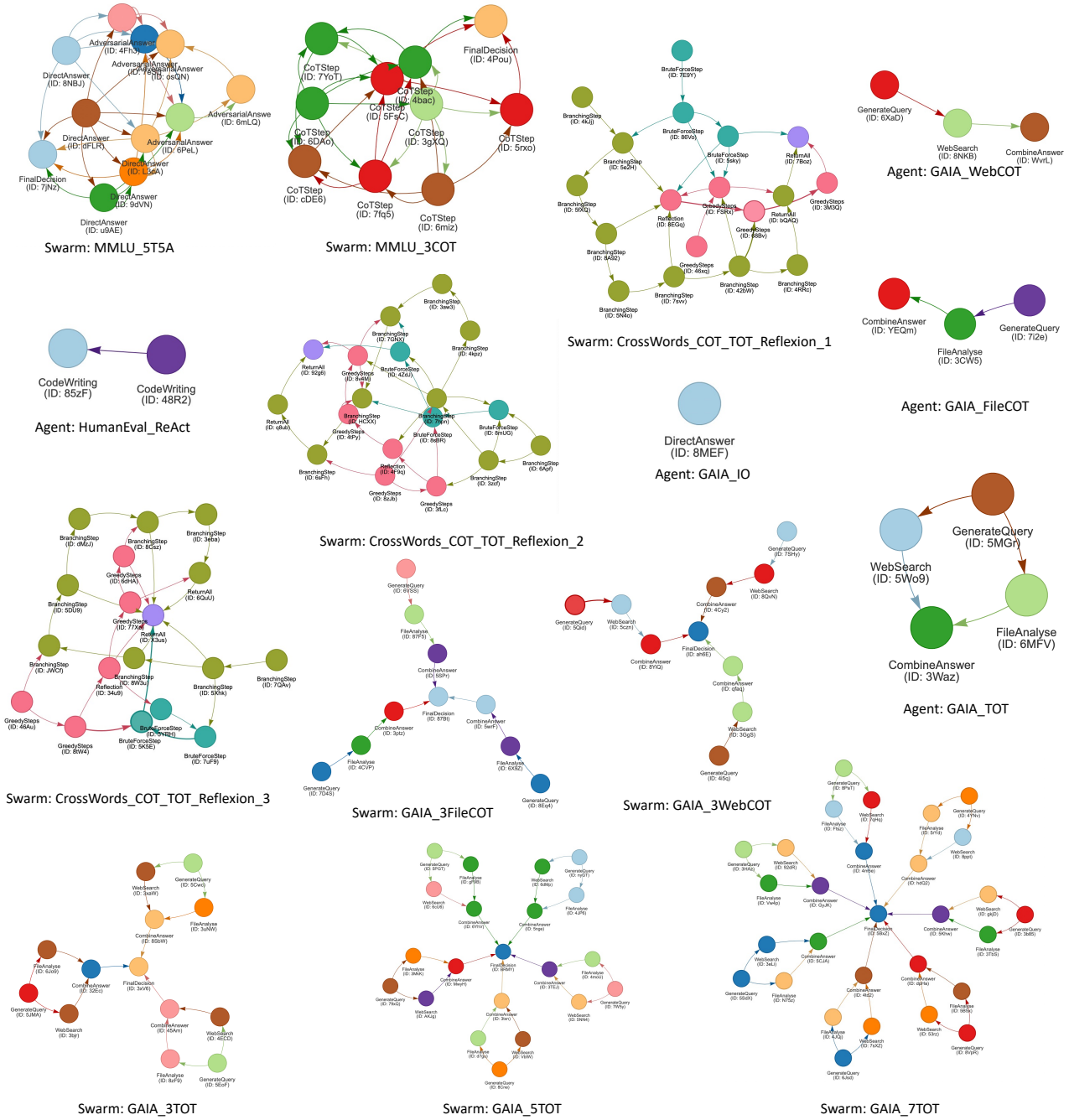


Figure 8. Different agents or swarms implemented by GPTSwarm.

D. Experimental Details

In our experiments involving multiple agents, we incorporate an additional virtual agent, represented by a single node, to serve as a final decision aggregator. This node is designated as the output node for the composite graph, and its specific implementation varies between different experiments. Common implementations for this node include employing a majority vote and a self-consistency strategy for decision-making. Unless explicitly stated, communication between agents within a composite graph does not include this virtual agent. Additionally, in all our experiments, the potential edge set of a composite graph is defined as all possible node pairs, provided that the nodes in each pair originate from different agents. We exclude any edges that would connect the output node of a composite graph to other nodes. Moreover, we employ the Adam (Kingma & Ba, 2014) optimizer with parameters $\beta_1 = 0.9, \beta_2 = 0.999$ and a variable learning rate in place of the vanilla stochastic gradient accent method described in Alg. 2. Finally, we use the version gpt-4-1106-preview and gpt-3.5-turbo-1106 for LLMs. For the vision-language model utilized in the GAIA experiments, we employed the gpt-4-1106-vision-preview version of GPT-4-Turbo.

D.1. MMLU

Statistical information on adversarial rubustness experiments is collected in Table 3. The convergence of the train utility with the baseline for the 3T3A experiment is shown in Figure 9.

Table 3. **Stats for the adversarial experiments.** #Nodes means the number of nodes in the swarm excluding the final decision node. #Potential edges is the total number of edges that are optimized and potentially realized. The optimization time is measured as the wall clock time. #LLM inferences is the total number of LLM queries made during the optimization cycle when graph pruning is turned off.

Swarm configuration	#Nodes	#Potential edges	Optimization time, mins
1 Trustful Agent + 1 Adversarial Agent	2	4	9
3 Trustful Agents + 3 Adversarial Agents	6	36	23
5 Trustful Agents + 5 Adversarial Agents	10	100	58
7 Trustful Agents + 7 Adversarial Agents	14	196	95

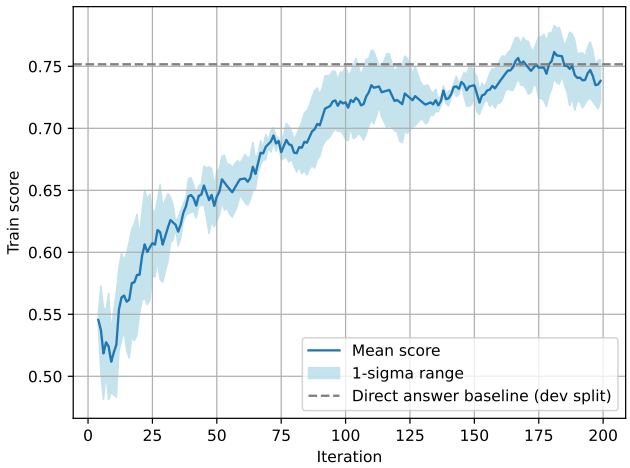


Figure 9. **The training score during the optimization of the adversarial swarm (3T3A) on MMLU.** We apply smoothing with an unbiased exponential moving average and the smoothness factor of 0.97.

D.1.1. Hyper-parameters & Prompts

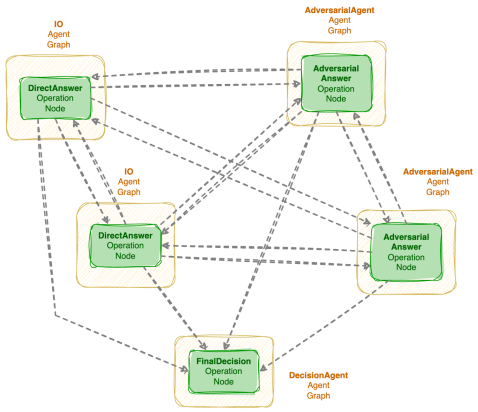
We use the Adam optimizer with a learning rate of 0.1 to update the logit parameters associated with each potential edge. The prompts that have been used for the adversarial robustness experiments are collected in Table 4.

Table 4. **Prompts for the Adversarial experiments on MMLU.**

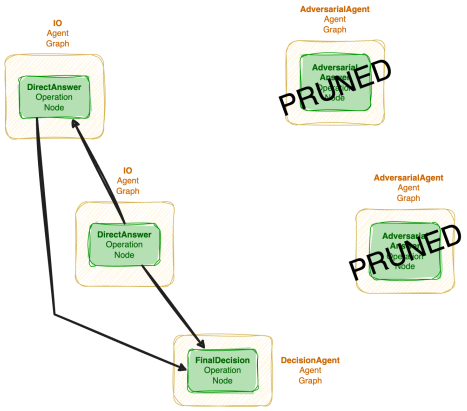
Prompt purpose	Prompt
System prompt	You are a knowledgeable expert in question answering. I will ask you a question. I will also give you 4 answers enumerated as A, B, C and D. Only one answer out of the offered 4 is correct. You must choose the correct answer to the question. Your response must be one of the 4 letters: A, B, C or D, corresponding to the correct answer. Only one letter (A, B, C or D) is allowed in your answer.
Direct answer template	{question}
Adversarial answer template	Answer a lie to the following question: {question}.
Question template	{open-ended question}. Option A: {option A}, Option B: {option B}, Option C: {option C}, Option D: {option D}.

D.1.2. Adversarial swarm optimization case study

An example of a swarm with 2 truthful and 2 adversarial examples is shown in Figure 10. Figure 10a shows all potential edges before optimization. Figure 10b shows only the edges that were connected after optimization was complete. Note that the disconnected agents and edges are pruned.



(a) An non-optimized swarm with 2 truthful agents and 2 adversarial agents. Dotted arrows depict potential edges.



(b) An optimized swarm with realized edges.

Figure 10. A 2T2A swarm before (a) and after (b) optimization

D.2. Mini Crosswords

D.2.1. Agents Setting

In our Mini Crosswords experiments, each node returns one or two solutions—either updated or unchanged—for each received solution. The solutions produced by a node are conditionally independent of each other, given the input solutions of the node. The output node of a composite graph forwards all received solutions without alteration. To ensure integration within the system, we mandate the existence of edges from any agent's output node directly to the composite graph's output node.

Our TOT agent uses a tree-search strategy across a perfect binary tree with a depth of eight. Instead of constructing a graph of $2^9 - 1$ nodes to represent this tree, the search is carried out through a chain of eight branching nodes. Each branching node is designed to generate two solutions from every input solution that it processes, effectively embodying the TOT strategy.

D.2.2. Hyper-parameters & Prompts

The candidate word generation prompt and the pruning prompt are adapted from the original TOT work (Yao et al., 2023) and detailed in Table 5. A clue is defined as a partial filling of the crossword, accompanied by its intended word description and specific position on the board.

D.3. HumanEval

D.3.1. The Node Optimization Method

For node optimization, we update each node after every four new problem executions. When addressing a new problem q with graph G, executing $G(q)$ produces a program, denoted by s, whose effectiveness is assessed against test examples associated with q. The input-output pairs of the nodes generated during the evaluation of $G(q)$ are classified as positive if s passes the tests and as negative otherwise. We limit each node to include a maximum of four demonstration examples. Let n be a node in the graph associated with a computational routine f_n^p, which returns Python programs, parameterized by demonstration examples p. During an optimization step of n, that is, an application of I as described in Section 2.4, we assess whether to retain existing demonstration examples (p_n^1) or to augment them with positive examples from the four most recent problems, subsequently randomly selecting up to four unique examples from this pool (denoted as p_n^2). More specifically, let Z be the set of the last ten inputs of node n received when solving the first-seen problems. We select p_n^i to update the demonstration examples of n, where $i = \arg\max_{i \in \{1,2\}} \sum_{z \in Z} \mathbb{1}_z(f_n^{p_n^i}(z, q_z))$, $\mathbb{1}_z$ determines whether a program passes the unit tests stated in z, and q_z is the original graph input associated with z.

The utility measure for Mini Crosswords experiments is defined as the best state word accuracy, as detailed in Section 3.2. To reduce the variance in gradient estimation with the REINFORCE algorithm, we adjust the utility by subtracting a constant of 0.4. For example, a perfectly completed solution results in a utility of 0.6, while an empty solution yields a utility of -0.4.

D.3.2. Hyper-parameters & Prompts

Table 6 shows the prompts used in our experiments following the principle of ReAct (Yao et al., 2022).

D.4. GAIA

D.4.1. Agent Setting

We design different agents and swarms. Representative agents and swarms are visualized in Figure 8.

D.4.2. Hyper-parameters & Prompts

We use GPT-4-Turbo for the experiments and design different node operations to solve the GAIA tasks. Table 7 and Table 8 show the prompts used in our experiments.

Table 5. **Prompts for the Mini Crosswords Experiments.**

Prompt purpose	Prompt
Candidate words generation prompt	Let's play a 5 x 5 mini crossword, where each word should have exactly 5 letters. {current board status} Unfilled: {Unfilled clues} Filled: {filled clues} Changed: {Changed clues} Suggestions: {suggestions generated by previous Reflection nodes} Given the current status, list all possible answers for unfilled or changed words, and your confidence levels (certain/high/medium/low), using the format "h1. apple (medium)". Use "certain" cautiously and only when you are 100% sure this is the correct word. You can list more then one possible answer for each word.
Pruning prompt	Evaluate if there exists a five letter word of some meaning that fit some letter constraints (sure/maybe/impossible). Incorrect; to injure: w _ o _ g The letter constraint is: 5 letters, letter 1 is w, letter 3 is o, letter 5 is g. Some possible words that mean "Incorrect; to injure": wrong (w r o n g): 5 letters, letter 1 is w, letter 3 is o, letter 5 is g. fit! sure A person with an all-consuming enthusiasm, such as for computers or anime: _ _ _ _ u The letter constraint is: 5 letters, letter 5 is u. Some possible words that mean "A person with an all-consuming enthusiasm, such as for computers or anime": geek (g e e k): 4 letters, not 5 otaku (o t a k u): 5 letters, letter 5 is u sure Dewy; roscid: r _ _ _ l The letter constraint is: 5 letters, letter 1 is r, letter 5 is l. Some possible words that mean "Dewy; roscid": moist (m o i s t): 5 letters, letter 1 is m, not r humid (h u m i d): 5 letters, letter 1 is h, not r I cannot think of any words now. Only 2 letters are constrained, it is still likely maybe A woodland: _ l _ d e The letter constraint is: 5 letters, letter 2 is l, letter 4 is d, letter 5 is e. Some possible words that mean "A woodland": forest (f o r e s t): 6 letters, not 5 woods (w o o d s): 5 letters, letter 2 is o, not l grove (g r o v e): 5 letters, letter 2 is r, not l I cannot think of any words now. 3 letters are constrained, and _ l _ d e seems a common pattern maybe An inn: _ d _ w f The letter constraint is: 5 letters, letter 2 is d, letter 4 is w, letter 5 is f. Some possible words that mean "An inn": hotel (h o t e l): 5 letters, letter 2 is o, not d lodge (l o d g e): 5 letters, letter 2 is o, not d I cannot think of any words now. 3 letters are constrained, and it is extremely unlikely to have a word with pattern _ d _ w f to mean "An inn" impossible Chance; a parasitic worm; a fish: w r a k _ The letter constraint is: 5 letters, letter 1 is w, letter 2 is r, letter 3 is a, letter 4 is k. Some possible words that mean "Chance; a parasitic worm; a fish": fluke (f l u k e): 5 letters, letter 1 is f, not w I cannot think of any words now. 4 letters are constrained, and it is extremely unlikely to have a word with pattern w r a k _ to mean "Chance; a parasitic worm; a fish" impossible {clue}
Suggestion Prompt	You are playing a 5 x 5 mini crossword, where each word should have exactly 5 letters. Given the current status: {current board status } The target words are classified as Impossible Words, Correct Words, and Incorrect Words. _ _ _ Impossible Words: {impossible clues} Correct Words: {correct clues } Incorrect Words: {incorrect clues } Respond at most five sentences, one sentence per line. Do not include the phrase "next time" in your response.

Table 6. **Prompts for the Node Optimization experiments on HumanEval.**

Prompt purpose	Prompt
System prompt	You are an AI that only responds with only Python code.
CodeWriting	You will be given a function signature and its docstring by the user. Write your full implementation (restate the function signature). Use a Python code block to write your response. For example: "'python print('Hello world!') "' {Demonstrations} {problem statement}
CodeWriting (ReAct)	You will be given a function signature and its docstring by the user. Write your full implementation (restate the function signature). Use a Python code block to write your response. For example: "'python print('Hello world!') "' {Demonstrations} Here is an unsuccessful attempt to solve the following question: Question: {problem statement} Attempted Solution: {previously generated program} Feedback: {internal unit test results} Rewrite the code based on the feedback and the following question: {problem statement}

Table 7. **Prompts for the Task-Solving experiments on GAIA (1).**

Prompt purpose	Prompt
System prompt	You are a general AI assistant. I will ask you a question. Report your thoughts, and finish your answer with the following template: FINAL ANSWER: [YOUR FINAL ANSWER]. YOUR FINAL ANSWER should be a number OR as few words as possible OR a comma separated list of numbers and/or strings. If you are asked for a number, don't use comma to write your number neither use units such as $ or percent sign unless specified otherwise. If you are asked for a string, don't use articles, neither abbreviations (e.g. for cities), and write the digits in plain text unless specified otherwise. If you are asked for a comma separated list, apply the above rules depending of whether the element to be put in the list is a number or a string.
DirectAnswer	{question}
GenerateQuery	# Information Gathering for Question Resolution Evaluate if additional information is needed to answer the question. If a web search or file analysis is necessary, outline specific clues or details to be searched for. ## Target Question: question ## Clues for Investigation: Identify critical clues and concepts within the question that are essential for finding the answer.
WebSearch	# Web Search Task ## Original Question: — {question} — ## Targeted Search Objective: — query — ## Simplified Search Instructions: Generate three specific search queries directly related to the original question. Each query should focus on key terms from the question. Format the output as a comma-separated list. For example, if the question is 'Who will be the next US president?', your queries could be: 'US presidential candidates, current US president, next US president'. Remember to format the queries as 'query1, query2, query3'.
DistillWebSearch	## Required Information for Summary: — {query} — ## Analyzed Search Results: — {results} — ## Instructions for Summarization: 1. Review the provided search results and identify the most relevant information related to the question and query. 2. Extract and highlight the key findings, facts, or data points from these results. 3. Organize the summarized information in a coherent and logical manner. 4. Ensure the summary is concise and directly addresses the query, avoiding extraneous details. 5. If the information from web search is useless, directly answer: N̈o useful information from WebSearch.̈
FileAnalyse	# File Analysis Task ## Information Extraction Objective: — {query} — ## File Under Analysis — {file} — ## Instructions: 1. Identify the key sections in the file relevant to the query. 2. Extract and summarize the necessary information from these sections. 3. Ensure the response is focused and directly addresses the query. Example: 'Identify the main theme in the text.'"

Table 8. **Prompts for the Task-Solving experiments on GAIA (2).**

Prompt purpose	Prompt
CombineAnswer	Reference information for FileAnalysis: — {file_analysis} — Reference information for Websearch: — {web_search} — Provide a specific answer. For questions with known answers, ensure to provide accurate and factual responses. Avoid vague responses or statements like 'unable to...' that don't contribute to a definitive answer. For example: if a question asks 'who will be the president of America', and the answer is currently unknown, you could suggest possibilities like 'Donald Trump', or 'Biden'. However, if the answer is known, provide the correct information."
FinalDecision (Self-Consistency)	# Self-Consistency Evaluation Task ## Question for Review: — {question} — ## Reviewable Answers: — {formatted_answers} — ## Instructions for Selection: 1. Read each answer and assess how it addresses the question. 2. Compare the answers for their adherence to the given question's criteria and logical coherence. 3. Identify the answer that best aligns with the question's requirements and is the most logically consistent. 4. Ignore the candidate answers if they do not give a direct answer, for example, using 'unable to ...', 'as an AI ...'. 5. Copy the most suitable answer as it is, without modification, to maintain its original form. 6. Adhere to the constraints: {constraint}. Note: If no answer fully meets the criteria, choose and copy the one that is closest to the requirements.
FinalDecision (Choose "Best")	## Question: — {question} — ## Candidate Answers for Evaluation: — {formatted_answers} — ## Evaluation Instructions: 1. Examine the question closely to understand its requirements. 2. Read each candidate answer thoroughly and assess its relevance and accuracy about the question. 3. Choose the answer that most accurately and completely addresses the question. 4. Ignore the candidate answers if they do not give a direct answer, for example, using 'unable to ...', 'as an AI ...'. "5. Copy the chosen answer exactly as it is presented, maintaining its original format. 6. Adhere to the constraints: {constraint}. Note: If none of the answers fully meet the question's criteria, select the one closest to fulfilling them.