Compound AI Systems Optimization: A Survey of Methods, Challenges, and Future Directions

Yu-Ang Lee* **Guan-Ting Yi*** **Mei-Yi Liu**
Jui-Chao Lu **Guan-Bo Yang** **Yun-Nung Chen**
National Taiwan University, Taipei, Taiwan
{r12946015,r13922053,r10228031,b09901142,r13922083}@ntu.edu.tw
y.v.chen@ieee.org

Abstract

Recent advancements in large language models (LLMs) and AI systems have led to a paradigm shift in the design and optimization of complex AI workflows. By integrating multiple components, compound AI systems have become increasingly adept at performing sophisticated tasks. However, as these systems grow in complexity, new challenges arise in optimizing not only individual components but also their interactions. While traditional optimization methods such as supervised fine-tuning (SFT) and reinforcement learning (RL) remain foundational, the rise of natural language feedback introduces promising new approaches, especially for optimizing non-differentiable systems. This paper provides a systematic review of recent progress in optimizing compound AI systems, encompassing both numerical and language-based techniques. We formalize the notion of compound AI system optimization, classify existing methods along several key dimensions, and highlight open research challenges and future directions in this rapidly evolving field.[1]

1 Introduction

The community has witnessed a new generation of AI systems centered on large language models (LLMs), incorporating several sophisticated components such as simulators, code interpreters, web search tools, and retrieval-augmented generation (RAG) modules. These systems have shown remarkable capabilities across domains and typically outperform standalone LLMs. For instance, LLMs communicating with symbolic solvers can tackle Olympiad-level math problems (Trinh et al., 2024), integrate with search engines and code interpreters to match human programmers' performance (Li et al., 2022; Yang et al., 2024b), and when coupled with knowledge graphs, drive biological materials discovery (Ghafarollahi and Buehler, 2024).

*Equal contribution
[1] https://github.com/MiuLab/AISysOpt-Survey

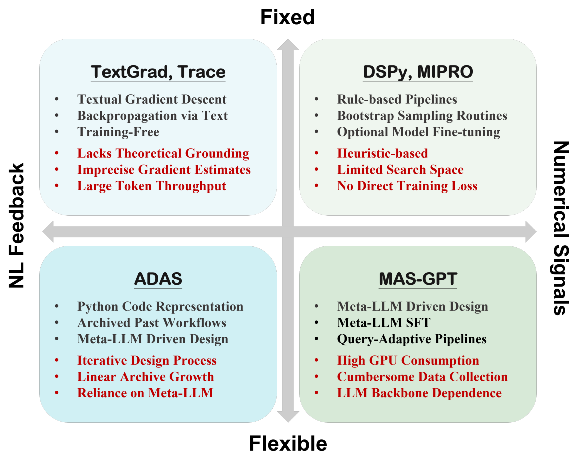


Figure 1: **The proposed 2×2 taxonomy** spans Structural Flexibility (y-axis) and Learning Signals (x-axis). Representative methods for each quadrant along with their key designs and potential drawbacks.

However, even with mature toolkits that streamline the design process of compound AI systems, such as, LangChain (Chase, 2022) and LlamaIndex (Liu, 2022), substantial human intervention remains essential to tailor these systems toward targeted downstream applications (Xia et al., 2024; Zhang et al., 2024c), often involving trial-and-error tuning prompt templates and system pipelines based on heuristics. This limitation has motivated recent efforts to develop principled, automated methods for end-to-end AI system optimization. Yet, the operational schemes of these approaches diverge significantly depending on whether they permit modifications to the system topology and how they transmit learning signals. Moreover, the field still lacks standardized terminology or a cohesive conceptual framework, making some articles difficult for newcomers to navigate (Cheng et al., 2024; Khattab et al., 2023). Despite existing surveys (Lin et al., 2024; Liu et al., 2025) have provided helpful frameworks, they focus on optimization driven by natural language, overlook critical

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 28760–28775
November 4-9, 2025 ©2025 Association for Computational Linguistics

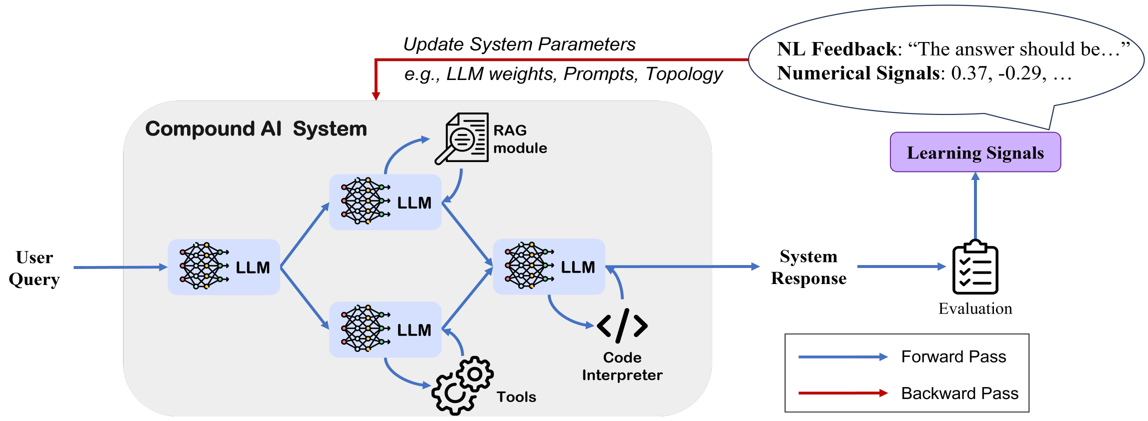


Figure 2: **Example of a compound AI system and its optimization.** Centered on LLMs and coupled with multiple interacting modules, the system handles complex user queries. Automated optimization strategies leverage two types of learning signals, i.e., natural language feedback and numerical signals (defined in Sec. 3.1), to backpropagate errors and guide system updates toward improved performance.

schemes that permit updates to system topology and do not cover the most recent advances.

To address these gaps, this study introduces four key dimensions for examining existing methods and, drawing on two of these dimensions, constructs a 2×2 taxonomy (Fig. 1) that holistically covers 26 representative works. Setting a reasonable scope for our survey, we excluded related but not directly relevant work. These include prompt optimization techniques for a single LLM (Zhou et al., 2022b; Pryzant et al., 2023; Yang et al., 2023; Das et al., 2024; Guo et al., 2025), systems built for task-specific applications without optimization (Zhuge et al., 2023; Hong et al., 2023), and papers that do not explicitly frame their problem as system optimization (Zhou et al., 2022a; Sordoni et al., 2023). The proposed taxonomy is inherently adaptable to future research, and we hope it will enable developers of compound AI systems to gain a high-level overview before delving into technical details, while allowing researchers proposing novel algorithms to better situate and compare their contributions.

The remainder of this paper is organized as follows: Sec. 2 provides the necessary background and formalization for compound AI systems and their optimization. Sec. 3.1 discusses the four key dimensions, followed by Sec. 3.2, which presents a detailed review of surveyed papers based on the proposed 2×2 taxonomy. Sec. 4 identifies the main challenges facing current methods and their corresponding future directions, and Sec. 5 concludes with a summary of our contributions.

2 Background and Preliminaries

2.1 Compound AI Systems

In contrast to single AI models that function as statistical models (e.g., the Transformer (Vaswani et al., 2017) for next-token prediction), compound AI systems are defined as systems that tackle AI tasks using multiple interacting components (Zaharia et al., 2024) (see Fig. 2). The term compound AI system somehow overlaps with related concepts and is often used interchangeably in the field. These include multi-agent systems (MAS) (Zhou et al., 2025; Wang et al., 2025b), language model pipelines (Khattab et al., 2023; Soylu et al., 2024), and language model programs (Opsahl-Ong et al., 2024). Hence, in this survey, we include any method that aims to optimize AI systems composed of multiple components, and adopt "compound AI system" as a unifying label.

Although end-to-end optimization of single models such as neural networks implemented in PyTorch (Paszke, 2019) is straightforward thanks to gradient-based backpropagation (Rumelhart et al., 1986) on their fully differentiable layer connections, compound AI systems are built from non-differentiable components and thus require novel optimization methods. Representative examples include heuristic bootstrap-based methods (Khattab et al., 2023) applied to find optimal in-context examples in LLM prompts, as well as approaches leveraging an auxiliary LLM to provide textual feedback on prompt updates (Yuksekgonul et al., 2025) or propose improved system topologies (Hu et al., 2024).

2.2 Formal Definitions

Due to inconsistent mathematical descriptions across the surveyed papers, we develop a unified graph-based formalization as well as notations for compound AI systems and their optimization[2].

Specifically, we use $\Phi = (G, \mathcal{F})$ to denote a compound AI system that inputs a user query $q \in \mathcal{Q}$ and outputs a response (answer) $a \in \mathcal{A}$. In particular, $G = (V, E)$ is a directed graph and $\mathcal{F} = \{f_i\}_{i=1}^{|V|}$ is a set of operations (e.g., LLM forward pass, RAG step, external tool calling). Each f_i is attached to node v_i, producing

$$Y_i = f_i(X_i;\, \Theta_i), \tag{1}$$

where X_i is the input to v_i, Y_i its output, and Θ_i its parameters. The edge matrix $E = [c_{ij}]$ comprises Boolean functions $c_{ij} : \Omega \to \{0, 1\}$ that determine whether the potential edge from v_i to v_j is active. This means given the contextual state $\tau \in \Omega$, the edge $(v_i \to v_j)$ is active if and only if $c_{ij}(\tau) = 1$. Consequently, even though G and $\mathcal{F}$ are fixed after optimization, the Φ's effective topology varies with τ, reflecting dependencies on both the input query q and intermediate outputs of the system, as visualized in Appendix Fig. 5.

In more detail, the node input is set by

$$X_i \;\leftarrow \bigoplus_{j:\, c_{ji}(\tau)=1} Y_j, \tag{2}$$

where $\oplus$ denotes concatenation of the outputs Y_j from all nodes v_j whose edge $v_j \to v_i$ is active. The node parameter Θ_i decomposes as $\Theta_i = (\theta_{i,N}, \theta_{i,T})$, where $\theta_{i,N}$ are numerical parameters (e.g., LLM weights, temperature) and $\theta_{i,T}$ are textual parameters (e.g., LLM prompts). Due to space constraints, descriptions of the input node v_{in}, the output node v_{out}, and our formalism's ability to either support or prohibit cyclic structures are deferred to Appendix Sec. A.

Given a training set $\mathcal{D} = \{(q_i, m_i)\}_{i=1}^N$, where each query q_i is associated with optional metadata $m_i \in \mathcal{M}$, such as output labels or hints useful for evaluating system correctness, and a performance metric $\mu : \mathcal{A} \times \mathcal{M} \to \mathbb{R}$, the goal of compound AI system optimization is defined by solving:

$$\max_{\Phi} \frac{1}{N} \sum_{i=1}^{N} \mu\big(\Phi(q_i), m_i\big). \tag{3}$$

[2]If our notations conflict with that of any original work, our definitions take precedence.

3 Compound AI System Optimization

With background established, we now turn to the core of our survey. Sec. 3.1 first present four principled dimensions underlying the optimization of compound AI systems, i.e., *Structural Flexibility*, *Learning Signals*, *Component Options*, and *System Representations*. Sec. 3.2 then review existing methods in the proposed 2×2 taxonomy.

3.1 Four Principled Dimensions

Structural Flexibility As described in Sec. 2.2, the pipeline of Φ can be modeled as a computation graph $G = (V, E)$. The concept of *Structural Flexibility* thus refers to the degree to which an optimization method can modify this graph. One class of methods, termed *Fixed Structure*, assumes a predefined topology (V, E) and focuses exclusively on optimizing the node parameters $\{\Theta_i\}$ (e.g., LLM prompts). In contrast, more recent methods acknowledge the importance of identifying optimal system topologies (Zhuge et al., 2024; Hu et al., 2024) and propose to jointly optimize both the node parameters and the graph structure itself, such as edge connections E, node counts $|V|$, and even the types of operations in $\mathcal{F}$. These methods, termed *Flexible Structure*, broaden the search space of Φ, enabling more effective orchestration of the system toward targeted tasks.

Learning Signals Regardless of structural flexibility, effective optimization of Φ requires *Learning Signals* that steer system updates toward improved task performance (Fig. 2). These signals naturally originate from system performance metric $\mu\big(\Phi(q_i), m_i\big)$, and can be conveyed in two distinct forms. The first type is natural language feedback (denoted as *NL Feedback*), where signals are generated by an auxiliary LLM that is separate from those used within the system itself (Yuksekgonul et al., 2025; Cheng et al., 2024; Hu et al., 2024). This approach leverages the reasoning capabilities of LLMs to provide textual guidance for system optimization. The second type, termed *Numerical Signals*, encompasses methods in which learning signals are represented numerically. We further categorize numerical signals into four different forms, as illustrated in Fig. 3.

Component Options This dimension distinguishes optimization methods by their considered types of operations in $\mathcal{F}$. Though most Φ center around LLMs, many frameworks integrate addi-

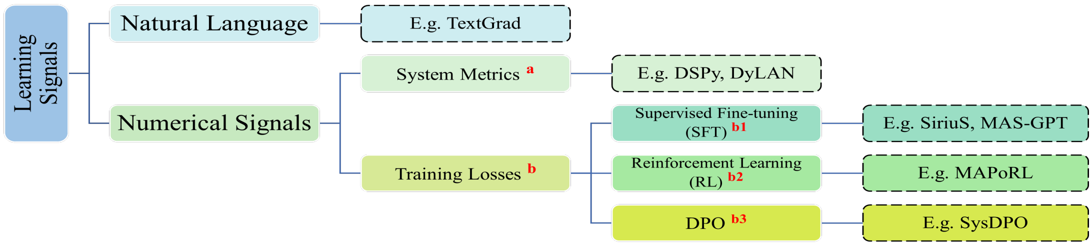


Figure 3: **Learning Signals** are classified into two categories, with Numerical Signals further divided by their utilization schemes: (a) one class of methods devises rule-based algorithms that directly learn from raw system performance metrics, and (b) another class transforms system evaluation results into formalized training objectives. These objectives are further split as (b1) supervised fine-tuning (SFT) losses, (b2) reinforcement learning (RL) reward functions, and (b3) direct preference optimization (DPO) (Rafailov et al., 2023) losses.

tional components to enrich domain knowledge or perform specialized tasks. These include RAG modules (Yin and Wang, 2025), code interpreters (CI) (Hu et al., 2024; Wang et al., 2025b), or Tools calling such as web search (Zhuge et al., 2024). In multi-modal contexts, additional components such as image generation models (IGM) are employed (Wang et al., 2025a). Note that some works assume an unrestricted components pool (Yuksekgonul et al., 2025) or do not explicitly specify their supported component options (Hu et al., 2024; Wang et al., 2025b). In those cases, we infer their component options by examining the components used in their experimental setups or by inspecting their system representations.[3]

System Representations Different representations are used to characterize Φ's operational topology across methods. The most common choice is to model Φ as a graph. A directed acyclic graph (DAG) ensures that each node is invoked at most once per forward pass (Khattab et al., 2023; Zhuge et al., 2024), while a cyclic graph supports schemes such as multi-turn debates (Subramaniam et al., 2025; Park et al., 2025) or multi-hop RAG (Yin and Wang, 2025). Although Liu et al. (2024) employs feed-forward networks as system representations, we also label them as graphs, given their equivalence. Acknowledging the potential limitations to optimize system topologies in graph space (Hu et al., 2024), another line of research represents Φ's workflow as natural language program (Li et al., 2024) or Python code (Hu et al., 2024; Zhang et al., 2024a; Ye et al., 2025), which supports conditional logic and loops without any acyclicity constraints.

[3]Methods that use Python code as *System Representations* inherently support code interpreters (CI).

3.2 Representative Methods

Equipped with the discussed four principled dimensions, here we review existing methods along two major axes: *Structural Flexibility* (Fixed vs. Flexible Structure) and *Learning Signals* (NL Feedback vs. Numerical Signals)[4]. All considered papers are listed in Table 1.

Fixed Structure, NL Feedback Most existing methods leverage LLMs' ability to generate rich, general natural language suggestions for prompt optimization in standalone LLMs (Zhou et al., 2022b; Pryzant et al., 2023; Yang et al., 2023). TextGrad (Yuksekgonul et al., 2025) is among the first to extend this idea to compound AI systems by updating their node parameters, e.g., LLM prompts ($\{\theta_{i,T}\}$). Drawing inspiration from numerical gradient descent in PyTorch (Paszke, 2019), TextGrad defines each node in the system graph as an independent computational unit, where optimization unfolds in three stages: (1) an *evaluator* LLM assesses the system's output $\Phi(q_i)$ against an expected reference m_i and generates textual loss signals; (2) for each participating node, a *gradient estimator* LLM generates node-specific textual suggestions conditioned on system dialogue and backpropagated loss; and (3) an *optimizer* LLM at each node refines node parameters using these suggestions. This process mirrors backpropagation via natural language, simulating differentiability across discrete modules.

Several works have since built upon TextGrad's framework. AIME (Patel et al., 2024) shows that for complex code generation tasks, using a single

[4]For a few methods that use both types of learning signals, we still categorize them into a single category. See Appendix Sec. C for more details.

Methods	Structural Flexibility	Learning Signals	Component Options	System Representations
TextGrad (2025)	Fixed	NL Feedback	LLM	Graph†
Trace (2024)	Fixed	NL Feedback	LLM	Graph†
AIME (2024)	Fixed	NL Feedback	LLM	Graph†
Revolve (2024b)	Fixed	NL Feedback	LLM	Graph†
GASO (2024a)	Fixed	NL Feedback	LLM	Graph†
LLM-AutoDiff (2025)	Fixed	NL Feedback	LLM; RAG	Graph
DSPy (2023)	Fixed	Numerical Signals^{a}	LLM	Graph†
MIPRO (2024)	Fixed	Numerical Signals^{a}	LLM; RAG	Graph
BetterTogether$*$ (2024)	Fixed	Numerical Signals^{a,b_1}	LLM; RAG	Graph†
SiriuS$*$ (2025)	Fixed	Numerical Signals^{b_1}	LLM	Graph
MAPoRL$*$ (2025)	Fixed	Numerical Signals^{b_2}	LLM	Graph
SysDPO$*$ (2025a)	Fixed	Numerical Signals^{b_3}	LLM; IGM	Graph†
Multiagent Finetuning$*$ (2025)	Fixed	Numerical Signals^{b_1}	LLM	Graph
Agent Symbolic Learning (2024)	Flexible	NL Feedback	LLM; RAG	Graph†
ADAS (2024)	Flexible	NL Feedback	LLM; CI	Python Code
AFlow (2024a)	Flexible	NL Feedback	LLM; CI	Python Code
MASS (2025)	Flexible	NL Feedback	LLM	Graph
DebFlow (2025)	Flexible	NL Feedback	LLM	Graph
DyLAN (2024)	Flexible	Numerical Signals^{a}	LLM	Graph
GPTSwarm (2024)	Flexible	Numerical Signals^{b_2}	LLM; Tools	Graph†
AutoFlow$*$ (2024)	Flexible	Numerical Signals^{b_2}	LLM	NL Programs
MaAS (2025)	Flexible	Numerical Signals^{b_2}	LLM; Tools; CI	Graph†
ScoreFlow$*$ (2025b)	Flexible	Numerical Signals^{b_3}	LLM; CI	Python Code
MAS-GPT$*$ (2025)	Flexible	Numerical Signals^{b_1}	LLM; CI	Python Code
W4S$*$ (2025)	Flexible	Numerical Signals^{b_2}	LLM; CI	Python Code
FlowReasoner$*$ (2025)	Flexible	Numerical Signals^{b_1,b_2}	LLM; CI	Python Code

Table 1: **Compound AI System Optimization methods**, sorted by their first publication date on arXiv. All methods and their properties along the four principled dimensions are listed. Superscripts ^{a}, ^{b_1}, ^{b_2}, and ^{b_3} denote the type of numerical signal each method employs (Fig. 3). An asterisk ($*$) indicates methods that require model fine-tuning. For graph-based system representations, a dagger (†) marks methods restricted to acyclic structures (i.e., DAGs).

evaluator LLM often allows errors in the generated code to go undetected, whereas concatenating outputs from multiple *evaluator* LLMs can mitigate this issue. REVOLVE (Zhang et al., 2024b) observes that *NL Feedback* is often applied in a first-order manner, causing stagnation and oscillations during system optimization. It therefore enriches the *gradient estimator* LLMs' input with a concise execution history of past prompts and responses, enabling the generation of curvature-aware feedback, similar to the Hessian in numerical optimization. GASO (Wang et al., 2024b) identifies the negligence of sibling-input interactions in TextGrad's backpropagation scheme, thus proposes semantic gradient descent to compute context-aware gradients and aggregate them for credit assignment. LLM-AutoDiff (Yin and Wang, 2025) addresses the intricacy of multi-component (e.g., large $|V|$, diverse $\mathcal{F}$) and cyclic system structures that prior work has not fully explored. In particular, it introduces *time-sequential gradients* to accumulate multiple textual gradients for nodes invoked repeatedly during a forward pass, and proposes an optional *skip-connections* mechanism, serving as powerful building block for optimizing large-scale systems via natural language.

Trace (Cheng et al., 2024), developed concurrently with TextGrad, introduces a joint optimization that updates all LLM prompts at once. It first obtains global *NL Feedback* from an *evaluator* LLM and then, in a single LLM invocation, updates every involved node by presenting the model with an minimal subgraph (analogous to execution trace in Python). This process address two caveats (Lin et al., 2024) in the backpropagation scheme of TextGrad: (i) error accumulation due to imprecise *NL Feedback*, and (ii) the linear growth in LLM calls with the number of nodes.

Serving as pioneers in the field, these methods demonstrate that learning from text is possible not only for single AI models but also for systems composed of discrete modules. Nevertheless, their successes are supported mainly by empirical results without theoretical grounding. In addition, the common reliance on proprietary LLMs to generate NL feedback incurs high API costs.

Fixed Structure, Numerical Signals Rather than relying on textual signals, methods in this category use numerical signals to update node parameters. DSPy (Khattab et al., 2023) provides a Python library[5] featuring declarative programming modules for designing and optimizing Φ. As a method that learns directly from raw system metrics, it introduces a suite of rejection-sampling–based routines (`Bootstrap-*`) for generating high-quality in-context demonstrations informed by corresponding system performance. Users may optionally fine-tune LLM weights ($\theta_{i,N}$) on the collected demonstrations via `BootstrapFinetune` procedure. MIPRO (Opsahl-Ong et al., 2024) advances by jointly optimizing demonstrations and instructions. Specifically, it employs a Bayesian surrogate model to maintain and update posterior distributions over instruction–demonstration configurations, favoring those that yield high performance. BetterTogether (Soylu et al., 2024) extends by alternating between LLM prompt and weight optimization, enabling LLMs to iteratively teach themselves and outperform single-strategy baselines.

Remaining methods in this category involve model fine-tuning, where numerical signals from system evaluation are instantiated as different types of training losses, as discussed in Fig. 3. In the SFT category, SiriuS (Zhao et al., 2025) constructs several system schemes by assigning predefined roles to its LLM nodes (e.g., "Physicist" and "Mathematician"). It then gathers reasoning trajectories, i.e., intermediate dialogue outputs, for queries q_i with high $\mu\big(\Phi(q_i), m_i\big)$ and independently supervised fine-tunes each LLM node using its corresponding input–output pairs. For q_i that results in failed attempts, SiriuS performs *trajectory augmentation* by resampling original attempts with feedback from an additional agent. Concurrent with SiriuS but building on the multiagent debate (Du et al., 2023) scheme, *multiagent finetuning* (Subramaniam et al., 2025) introduces novel rules for collecting training data for each *generation* and *critic* model for subsequent SFT.

Belonging to the RL category, MAPoRL (Park et al., 2025) targets the multiagent debate (Du et al., 2023) scenario as well. It differs from Subramaniam et al. (2025) by training a *verifier* LLM to assign immediate correctness rewards to each LLM node, and introduces influence-aware reward shaping to incentivize collaboration. Finally, in the DPO category, SysDPO (Wang et al., 2025a) features a system characterized by an LLM and a diffusion model (Rombach et al., 2022). Aiming at image generation tasks, SysDPO curates a preference dataset by computing preference scores based on images' *order consistency* and *distribution evenness*. Unlike original DPO loss, SysDPO incorporates probability decomposition to enable fine-tuning multiple components in Φ.

These methods introduce novel ways to leverage system performance signals, effectively mitigating the imprecision of natural language. However, excessive human-designed rules may limit generalizability. Additionally, fine-tuning each LLM in Φ incurs substantial GPU resource demands.

Flexible Structure, NL Feedback Methods discussed previously tune only node parameters within a predefined system topology. To overcome this constraints, methods in this category leverage NL Feedback to jointly optimize both node parameters and overall system structure. Concurrent with TextGrad (Yuksekgonul et al., 2025), Agent Symbolic Learning (Zhou et al., 2024) designs optimizers with three components: *PromptOptimizer*, *ToolOptimizer*, and *PipelineOptimizer*, making it goes beyond node tuning because the latter two components support tool creation as well as node addition, deletion, and movement. Through a series of analytical experiments, MASS (Zhou et al., 2025) demonstrates that optimizing LLM prompts $\{\theta_{i,T}\}$ can deliver performance gains in compound AI systems more easily than exploring the topology (V, E). Building on this insight, MASS devises a three-stage framework in which prompt optimization precedes topology search.

Recognizing that previous approaches using graphs as system representations may not fully cover the space of possible system designs and that this space is inherently difficult to search, ADAS (Hu et al., 2024) is the first to adopt Python code as system representation. Specifically, conditioned on an archive maintaining prior code and its corresponding performance metric, a meta LLM is asked to iteratively design novel workflows. Owing to the vast search space of code representations, ADAS faces challenges in its linear heuristic search processes and coarse workflow storage, which lead to the accumulation of irrelevant information for the meta LLM (Zhang et al., 2024a). Identifying this, AFlow (Zhang et al., 2024a) leverages the tree structure of Monte Carlo Tree Search (MCTS)

[5] `http://dspy.ai`

to preserve past experience and employs a set of predefined operators to efficiently identify optimal system designs. DebFlow (Su et al., 2025) argues the limitation posed by using a single meta LLM in this scheme, and introduces an innovative debate framework where multiple *debaters* propose their opinions on system designs, with a *judge* helping to conclude the refined workflow.

Methods in this category demonstrate how novel algorithms can leverage the power of proprietary LLMs for direct workflow design. Nonetheless, effective engagement with these meta LLMs demands high token consumption (Hu et al., 2024) or multiple LLM inferences (Su et al., 2025). Also, evaluations on less powerful open-source models remain scarce.

Flexible Structure, Numerical Signals Completing our review, this passage describes methods that employ *Numerical Signals* to refine Φ without posing fixed topology constraints. DyLAN (Liu et al., 2024) and GPTSwarm (Zhuge et al., 2024) are methods in this category that do not involve model fine-tuning. DyLAN models multiturn debate as a temporal feed-forward network in which agents prompted with distinct roles are unrolled across layers. The system is optimized through pruning unhelpful agents and adaptively connect the surviving agents between consecutive layers. Rule-based algorithms are employed to score each agent's importance, positioning DyLAN within the category that leverages raw system metrics as learning signals. GPTSwarm models Φ as a hierarchical framework composed of *nodes* ($v_i \in V$), *agents* (graphs that link nodes), and *swarms* (composite graphs that interconnect multiple agents), then employs an optimization procedure to refine the edge connections among *agents*. A parameterized probabilistic distribution D_θ is introduced to govern the connectivity within the swarm graph, which is then optimized using a gradient-ascent variant of the REINFORCE (Williams, 1992) algorithm, placing GPTSwarm within the class of RL methods.

Before discussing the remaining methods, we note that they operate in a query-adaptive manner, i.e., instantiating a new Φ for each q_i. This operating scheme contrasts with methods discussed so far in which a single task-specific Φ, once optimized, is used for all q_i. Similar to ADAS (Hu et al., 2024) and AFlow (Zhang et al., 2024a), these methods leverage a meta LLM to generate optimal system designs. However, rather than relying solely on inference, they fine-tune the meta LLM using constructed training datasets or reward functions to enable effective pipeline generation, given the limited prior knowledge about compound AI system design within LLMs

Starting from the SFT category, MAS-GPT (Ye et al., 2025) first constructs a *query pool* from open-source queries across various domains, and a *MAS pool* by manually implementing over 40 common system designs in Python code. Through *evaluation*, *selection*, and *refinement* of query–MAS pairs, MAS-GPT curates a training dataset that ensures consistency, minimizing ambiguity when fine-tuning the meta LLM.

Next, within the RL category, AutoFlow (Li et al., 2024) prompts the meta LLM to generate Φ using CoRE (Xu et al., 2024b) syntax, then iteratively fine-tunes it with the average score on task data $\mathcal{D}$ as the reward. It also offers a workaround for closed-source meta LLMs by in-context learning. MaAS (Zhang et al., 2025) constructs and optimizes an *agentic supernet*, a probabilistic distribution over agentic architectures. The token cost incurred during system execution is also considered as a loss term to achieve a trade-off between system performance and complexity. W4S (Nie et al., 2025) maximizes the meta LLM's flexibility by constraining only the workflow interfaces, without predefining any system modules. By using a small (weak) model to reduce training cost, W4S casts the problem as a multi-turn Markov Decision Process (MDP), in which the meta LLM progressively learns to design and refine Φ based on environmental feedback. FlowReasoner (Gao et al., 2025) combines SFT and RL by first fine-tuning the meta LLM for basic reasoning regarding system generation, and then applying RL with a multi-purpose reward to further optimize the model.

Finally, ScoreFlow (Wang et al., 2025b) introduces Score-DPO, an extension of the original DPO. During each iteration, for every query q_i, multiple candidate Φ are sampled from the meta LLM and evaluated by an LLM executor; preference data are then collected based on the observed differences in system performance.

Recent trends in this category frame compound system optimization as a meta LLM fine-tuning problem, thereby sidestepping the non-differentiability of modules within Φ. However, substantial effort is still required to curate high-quality training data, and the lack of comprehensive evaluation across different model families limits the practical applicability of these methods.

4 Challenges and Future Directions

After reviewing representative methods, this section presents several key challenges and their future directions. Due to space constraints, additional discussion is deferred to Appendix Sec. B.

Manual Hyperparameter Configuration Despite aiming to automate the optimization process, we identify residual human interventions in existing methods, particularly in the configuration of algorithm-related hyperparameters, that challenge automation claims and limit practical value.

In particular, methods in the Fixed Structure category require users to configure system topologies based on domain expertise. Although some studies evaluate their algorithms across multiple system designs (Yin and Wang, 2025; Zhao et al., 2025), there is no guarantee that these configurations will meet the requirements of all target applications. Textual hyperparameters also frequently appear in various methods. For example, the prompt templates used for the *evaluator*, *gradient estimator*, and *optimizer* in TextGrad (Yuksekgonul et al., 2025), as well as those for the meta LLM in ADAS (Hu et al., 2024), are handcrafted by the original authors and often lack a clear design rationale or sensitivity analysis of their wording.

Numerical hyperparameter decisions persist as well; for instance, the number of bootstrap samples in DSPy (Khattab et al., 2023) remains user-tunable and cannot be automated. Even seemingly automated numerical flexible-structure methods, such as MAS-GPT (Ye et al., 2025), require manual configuration, as evidenced by the prompt templates for *pair refinement*.

Despite the efforts of metaTextGrad (Xu et al., 2024a), which applies meta-learning to automatically optimize templates for *evaluator*, human intervention remains to craft the meta-learner's prompts. To move toward truly automated system optimization, akin to near hyperparameter-free neural network training, we urge future research to reduce reliance on both textual and numerical hyperparameters. For any hyperparameters that remain, we advocate thorough sensitivity analyses to help users understand each method's behavior and robustness.

Excessive Computation Burden Optimization of compound AI systems is inherently more challenging than tuning individual models. Existing approaches thus resort to various workarounds, leading to substantially higher computational overhead.

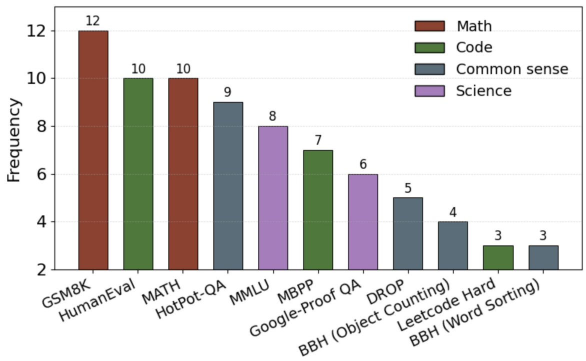


Figure 4: The frequency statistics of benchmarks tested in the surveyed 26 papers.

In NL feedback learning, methods such as TextGrad require multiple LLM calls to approximate a single gradient step. Although methods like Trace (Cheng et al., 2024) and ADAS (Hu et al., 2024) use global LLM call(s) per optimization step, they must embed extensive context in their prompts (e.g., *minimal subgraphs* or *agent archives*), which increases token throughput. Since these methods typically rely on proprietary models (Achiam et al., 2023), they incur substantial API costs. Conversely, numerical signal-based methods often leverage open-source LLMs to avoid API expenses. These models typically require fine-tuning to perform well, thereby shifting the burden to GPU resources. Developers are thus faced with a trade-off between API costs and GPU consumption.

Moreover, computational burden also arises during inference. By focusing primarily on system performance, current Flexible Structure methods often overlook the need to regularize system complexity (e.g., multi-round loops or lengthy executions), resulting in unbounded resource consumption at run time. Although a few methods (Zhang et al., 2025; Gao et al., 2025) have begun to encourage less complex system designs (e.g., lower token consumption), their applicability and scalability in large-scale deployments remain to be tested (Liu et al., 2025). We therefore suggest that future research develop resource-efficient optimization algorithms and at the same time devise principled ways to constrain system complexity without compromising performance.

Limited Experimental Scope Since compound AI systems are expected to address complex problems, it is important to investigate their effectiveness on more challenging tasks. Yet, papers in the field primarily evaluate their proposed methods on datasets widely used for single LLMs (Fig. 4), such as those for math reasoning (e.g., GSM8K (Cobbe et al., 2021)), commonsense reasoning (e.g., MMLU (Hendrycks et al., 2021)), and code generation (e.g., HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021)). While these evaluations reflect general effectiveness, we argue that it is also important to include benchmarks involving more complex tasks. For instance, AgentBench (Liu et al., 2023) and AgentGym (Xi et al., 2024) comprise multiple tasks requiring different LLMs within the system to cooperate and discuss, and GAIA (Mialon et al., 2023) examines system effectiveness in real-world scenarios requiring various tool usage. Furthermore, given the broad utility of compound AI systems (e.g., healthcare systems with doctors embedded as nodes in Φ (Chen et al., 2024)), it is necessary to evaluate algorithmic performance when humans function as nodes within the system and to devise principled methods for modeling their behavior.

Empirical NL Feedback While NL Feedback methods have shown promising empirical results, they lack theoretical guarantees. For example, the convergence of textual gradient descent remains unproven, whereas classical gradient descent are supported by formal convergence proofs (Cheridito et al., 2022; Hutzenthaler et al., 2021). Such proofs provide a solid foundation for the continuous advancement of single-model optimization. We therefore advocate future work to deliver rigorous convergence and optimality analyses for learning via NL Feedback, which will offer deeper insights and strengthen the field's theoretical underpinnings.

Potential Safety Risks While safety issues and corresponding defenses for single models have been extensively studied, such as jailbreak attacks on LLMs and their mitigation (Yi et al., 2024) and human-engineered AI pipelines to reduce harmful prompts and outputs (Han et al., 2024), the attack surface expands considerably in compound AI systems (Banerjee et al., 2024). For instance, Debenedetti et al. (2024) demonstrate that privacy-preserving models can still leak sensitive data when integrated as a component of a larger system. Furthermore, because compound AI systems are often embodied and executed as code within enterprise environments, latent failure modes may remain undetected and undermine system reliability even in the absence of explicit adversarial attacks. Despite these risks, research on compound AI system optimization that extends beyond downstream performance has so far addressed mainly execution efficiency (Zhang et al., 2025), with little attention to system-level alignment or safety (Zheng et al., 2025). Given the mature alignment and safeguarding optimization techniques available for single models (Achiam et al., 2017; Dai et al., 2023), we urge future work to adapt and extend these strategies to compound systems in order to balance capability enhancements with safety guarantees (Yang et al., 2024a).

Inconsistent Library Support During our survey, we observed a lack of a standardized and widely adopted library in the field. Although some well-maintained libraries such as TextGrad (Yuksekgonul et al., 2025) and DSPy (Khattab et al., 2023) have gained popularity among practitioners, still a great portion of existing works implement compound AI systems optimization using custom, self-crafted codebases. While frameworks such as TensorFlow (Abadi et al., 2015) and PyTorch (Paszke, 2019) dominate single-model training, best practices for implementing and optimizing compound AI systems are still under development. We suggest that future efforts focus on systematically benchmarking and comparing existing libraries for compound AI system optimization. Such efforts could support their improvement and help establish clearer guidelines for developers and researchers—for example, regarding when to use which library—as is done in the context of single-model training (Novac et al., 2022).

5 Conclusions

We survey recent advances in optimizing compound AI systems composed of interacting components like agents and tools. To unify diverse research efforts, we propose a graph-based formalism with conditional edges, enabling structured analysis of system interactions. With this framework, we examine existing methods across four principled dimensions and organize them into a 2×2 taxonomy. Our survey highlights key trends and trade-offs, including computational overhead, the NL interface, and challenges in scaling and generalization. We also identify open problems and outline future directions to guide continued progress in the field.

Limitations

We acknowledge several limitations of this survey. First, due to the lack of a universally accepted definition of "compound AI systems," we also include works that self-identify as optimizing multi-agent systems (MAS) or LM programs, without systematically analyzing their conceptual overlaps and distinctions.

Second, we focus exclusively on methods that explicitly optimize systems of multiple nodes, thereby excluding traditional prompt optimization techniques for standalone LLMs and contributions not framed as system optimization. Despite our efforts to draw a clear boundary, some relevant papers may have been inadvertently omitted; moreover, as this field is actively evolving during the preparation of this manuscript, studies published in the last two months may be only partially covered. To address this, we maintain an open-source repository where readers can submit works we may have missed.

Third, due to page limits, we highlight only the core motivation and algorithmic design of each method and omit details such as experimental setups and results. Readers are encouraged to consult the original papers and code repositories for full technical details once they have gained an overview from our survey.

Acknowledgements

We thank the reviewers for their insightful comments. This work was financially supported by the National Science and Technology Council (NSTC) in Taiwan, under Grants 111-2222-E-002-013-MY3 and 112-2223-E002-012-MY5, and the Center of Data Intelligence: Technologies, Applications, and Systems, National Taiwan University under the Grant 114L900901, from the Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) of Taiwan.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow, Large-scale machine learning on heterogeneous systems.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. 2017. Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Sarbartha Banerjee, Prateek Sahu, Mulong Luo, Anjo Vahldiek-Oberwagner, Neeraja J Yadwadkar, and Mohit Tiwari. 2024. Sok: A systems perspective on compound ai threats and countermeasures. *arXiv preprint arXiv:2411.13459*.

Harrison Chase. 2022. LangChain.

Gohar Irfan Chaudhry, Esha Choukse, Íñigo Goiri, Rodrigo Fonseca, Adam Belay, and Ricardo Bianchini. 2025. Towards resource-efficient compound ai systems. *arXiv preprint arXiv:2501.16634*.

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Matei Zaharia, James Zou, and Ion Stoica. 2025a. Optimizing model selection for compound ai systems. *arXiv preprint arXiv:2502.14815*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Minghui Chen, Ruinan Jin, Wenlong Deng, Yuanyuan Chen, Zhi Huang, Han Yu, and Xiaoxiao Li. 2025b. Can textual gradient work in federated learning? In *The Thirteenth International Conference on Learning Representations*.

Zhiliang Chen, Chuan-Sheng Foo, and Bryan Kian Hsiang Low. 2024. Towards autoai: Optimizing a machine learning system with black-box and differentiable components. In *Forty-first International Conference on Machine Learning*.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. 2024. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and LLMs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Patrick Cheridito, Arnulf Jentzen, Adrian Riekert, and Florian Rossmannek. 2022. A proof of convergence for gradient descent in the training of artificial neural networks for constant target functions. *Journal of Complexity*, 72:101646.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Josef Dai, Xuehai Pan, Ruiyang Sun, Jiaming Ji, Xinbo Xu, Mickel Liu, Yizhou Wang, and Yaodong Yang. 2023. Safe rlhf: Safe reinforcement learning from human feedback. *arXiv preprint arXiv:2310.12773*.

Sarkar Snigdha Sarathi Das, Ryo Kamoi, Bo Pang, Yusen Zhang, Caiming Xiong, and Rui Zhang. 2024. Greater: Gradients over reasoning makes smaller language models strong prompt optimizers. *Preprint*, arXiv:2412.09722.

Edoardo Debenedetti, Giorgio Severi, Nicholas Carlini, Christopher A Choquette-Choo, Matthew Jagielski, Milad Nasr, Eric Wallace, and Florian Tramèr. 2024. Privacy side channels in machine learning systems. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 6861–6848.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*.

Hongcheng Gao, Yue Liu, Yufei He, Longxu Dou, Chao Du, Zhijie Deng, Bryan Hooi, Min Lin, and Tianyu Pang. 2025. Flowreasoner: Reinforcing query-level meta-agents. *arXiv preprint arXiv:2504.15257*.

Alireza Ghafarollahi and Markus J Buehler. 2024. Sciagents: Automating scientific discovery through multi-agent intelligent graph reasoning. *arXiv preprint arXiv:2409.05556*.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2025. Evoprompt: Connecting llms with evolutionary algorithms yields powerful prompt optimizers. *Preprint*, arXiv:2309.08532.

Shanshan Han, Zijian Hu, Alay Dilipbhai Shah, Han Jin, Yuhang Yao, Dimitris Stripelis, Zhaozhuo Xu, and Chaoyang He. 2024. Torchopera: A compound ai system for llm safety. *arXiv preprint arXiv:2406.10847*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Preprint*, arXiv:2009.03300.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 3(4):6.

Shengran Hu, Cong Lu, and Jeff Clune. 2024. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*.

Martin Hutzenthaler, Arnulf Jentzen, Katharina Pohl, Adrian Riekert, and Luca Scarpa. 2021. Convergence proof for stochastic gradient descent in the training of deep neural networks with relu activation for constant target functions. *arXiv preprint arXiv:2112.07369*.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *Preprint*, arXiv:2310.03714.

Yafu Li, Xuyang Hu, Xiaoye Qu, Linjie Li, and Yu Cheng. 2025. Test-time preference optimization: On-the-fly alignment via iterative textual feedback. *Preprint*, arXiv:2501.12895.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and Yongfeng Zhang. 2024. Autoflow: Automated workflow generation for large language model agents. *arXiv preprint arXiv:2407.12821*.

Matthieu Lin, Jenny Sheng, Andrew Zhao, Shenzhi Wang, Yang Yue, Yiran Wu, Huan Liu, Jun Liu, Gao Huang, and Yong-Jin Liu. 2024. Llm-based optimization of compound ai systems: A survey. *arXiv preprint arXiv:2410.16392*.

Bang Liu, Xinfeng Li, Jiayi Zhang, Jinlin Wang, Tanjin He, Sirui Hong, Hongzhang Liu, Shaokun Zhang, Kaitao Song, Kunlun Zhu, Yuheng Cheng, Suyuchen Wang, Xiaoqiang Wang, Yuyu Luo, Haibo Jin, Peiyan Zhang, Ollie Liu, Jiaqi Chen, Huan Zhang, Zhaoyang Yu, Haochen Shi, Boyan Li, Dekun Wu, Fengwei Teng, Xiaojun Jia, Jiawei Xu, Jinyu Xiang, Yizhang Lin, Tianming Liu, Tongliang Liu, Yu Su, Huan Sun, Glen Berseth, Jianyun Nie, Ian Foster, Logan Ward, Qingyun Wu, Yu Gu, Mingchen Zhuge, Xiangru Tang, Haohan Wang, Jiaxuan You, Chi Wang, Jian Pei, Qiang Yang, Xiaoliang Qi, and Chenglin Wu. 2025. Advances and challenges in foundation agents: From brain-inspired intelligence to evolutionary, collaborative, and safe systems. *Preprint*, arXiv:2504.01990.

Jerry Liu. 2022. LlamaIndex.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv: 2308.03688*.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2024. A dynamic llm-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*.

Fan Nie, Lan Feng, Haotian Ye, Weixin Liang, Pan Lu, Huaxiu Yao, Alexandre Alahi, and James Zou. 2025. Weak-for-strong: Training weak meta-agent to harness strong executors. *arXiv preprint arXiv:2504.04785*.

Ovidiu-Constantin Novac, Mihai Cristian Chirodea, Cornelia Mihaela Novac, Nicu Bizon, Mihai Oproescu, Ovidiu Petru Stan, and Cornelia Emilia Gordan. 2022. Analysis of the application efficiency of tensorflow and pytorch in convolutional neural network. *Sensors*, 22(22):8872.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. *arXiv preprint arXiv:2406.11695*.

Chanwoo Park, Seungju Han, Xingzhi Guo, Asuman Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. 2025. Maporl: Multi-agent post-co-training for collaborative large language models with reinforcement learning. *Preprint*, arXiv:2502.18439.

A Paszke. 2019. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.

Bhrij Patel, Souradip Chakraborty, Wesley A. Suttle, Mengdi Wang, Amrit Singh Bedi, and Dinesh Manocha. 2024. Aime: Ai system optimization via multiple llm evaluators. *Preprint*, arXiv:2410.03131.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Keshav Santhanam, Deepti Raghavan, Muhammad Shahir Rahman, Thejas Venkatesh, Neha Kunjal, Pratiksha Thaker, Philip Levis, and Matei Zaharia. 2024. Alto: An efficient network orchestrator for compound ai systems. In *Proceedings of the 4th Workshop on Machine Learning and Systems*, pages 117–125.

Alessandro Sordoni, Eric Yuan, Marc-Alexandre Côté, Matheus Pereira, Adam Trischler, Ziang Xiao, Arian Hosseini, Friederike Niedtner, and Nicolas Le Roux. 2023. Joint prompt optimization of stacked llms using variational inference. *Advances in Neural Information Processing Systems*, 36:58128–58151.

Dilara Soylu, Christopher Potts, and Omar Khattab. 2024. Fine-tuning and prompt optimization: Two great steps that work better together. *arXiv preprint arXiv:2407.10930*.

Jinwei Su, Yinghui Xia, Ronghua Shi, Jianhui Wang, Jianuo Huang, Yijin Wang, Tianyu Shi, Yang Jingsong, and Lewei He. 2025. Debflow: Automating agent creation via agent debate. *Preprint*, arXiv:2503.23781.

Vighnesh Subramaniam, Yilun Du, Joshua B. Tenenbaum, Antonio Torralba, Shuang Li, and Igor Mordatch. 2025. Multiagent finetuning: Self improvement with diverse reasoning chains. In *The Thirteenth International Conference on Learning Representations*.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. 2024. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482.

Patara Trirat, Wonyong Jeong, and Sung Ju Hwang. 2024. Automl-agent: A multi-agent llm framework for full-pipeline automl. *Preprint*, arXiv:2410.02958.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Wenyi Wang, Hisham A Alyahya, Dylan R Ashley, Oleg Serikov, Dmitrii Khizbullin, Francesco Faccio, and Jürgen Schmidhuber. 2024a. How to correctly do semantic backpropagation on language-based agentic systems. *arXiv preprint arXiv:2412.03624*.

Wenyi Wang, Hisham A. Alyahya, Dylan R. Ashley, Oleg Serikov, Dmitrii Khizbullin, Francesco Faccio, and Jürgen Schmidhuber. 2024b. How to correctly do semantic backpropagation on language-based agentic systems. *Preprint*, arXiv:2412.03624.

Xiangwen Wang, Yibo Jacky Zhang, Zhoujie Ding, Katherine Tsai, and Sanmi Koyejo. 2025a. Aligning compound ai systems via system-level dpo. *arXiv preprint arXiv:2502.17721*.

Yinjie Wang, Ling Yang, Guohao Li, Mengdi Wang, and Bryon Aragam. 2025b. Scoreflow: Mastering llm agent workflows via score-based preference optimization. *arXiv preprint arXiv:2502.04306*.

Yangbo Wei, Zhen Huang, Huang Li, Wei W. Xing, Ting-Jung Lin, and Lei He. 2025. Vflow: Discovering optimal agentic workflows for verilog generation. *Preprint*, arXiv:2504.03723.

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

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. 2024. Agentgym: Evolving large language model-based agents across diverse environments. *Preprint*, arXiv:2406.04151.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2024. Agentless: Demystifying llm-based software engineering agents. *arXiv preprint arXiv:2407.01489*.

Guowei Xu, Mert Yuksekgonul, Carlos Guestrin, and James Zou. 2024a. metatextgrad: Learning to learn with language models as optimizers. In *Adaptive Foundation Models: Evolving AI for Personalized and Efficient Learning*.

Shuyuan Xu, Zelong Li, Kai Mei, and Yongfeng Zhang. 2024b. Core: Llm as interpreter for natural language programming, pseudo-code programming, and flow programming of ai agents. *arXiv e-prints*, pages arXiv–2405.

Chao Yang, Chaochao Lu, Yingchun Wang, and Bowen Zhou. 2024a. Towards ai-45° law: A roadmap to trustworthy agi. *Preprint*, arXiv:2412.14186.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024b. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652.

Rui Ye, Shuo Tang, Rui Ge, Yaxin Du, Zhenfei Yin, Siheng Chen, and Jing Shao. 2025. Mas-gpt: Training llms to build llm-based multi-agent systems. *arXiv preprint arXiv:2503.03686*.

Sibo Yi, Yule Liu, Zhen Sun, Tianshuo Cong, Xinlei He, Jiaxing Song, Ke Xu, and Qi Li. 2024. Jailbreak attacks and defenses against large language models: A survey. *arXiv preprint arXiv:2407.04295*.

Li Yin and Zhangyang Wang. 2025. Llm-autodiff: Auto-differentiate any llm workflow. *Preprint*, arXiv:2501.16673.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. 2025. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616.

Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. 2024. The shift from models to compound ai systems. `https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/`.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. 2025. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. 2024a. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*.

Peiyan Zhang, Haibo Jin, Leyang Hu, Xinnuo Li, Liying Kang, Man Luo, Yangqiu Song, and Haohan Wang. 2024b. Revolve: Optimizing ai systems by tracking response evolution in textual optimization. *Preprint*, arXiv:2412.03092.

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Arik. 2024c. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237.

Wanjia Zhao, Mert Yuksekgonul, Shirley Wu, and James Zou. 2025. Sirius: Self-improving multi-agent systems via bootstrapped reasoning. *Preprint*, arXiv:2502.04780.

Chengqi Zheng, Jianda Chen, Yueming Lyu, Wen Zheng Terence Ng, Haopeng Zhang, Yew-Soon Ong, Ivor Tsang, and Haiyan Yin. 2025. Mermaidflow: Redefining agentic workflow generation via safety-constrained evolutionary programming. *arXiv preprint arXiv:2505.22967*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. 2022a.

Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Ö Arık. 2025. Multi-agent design: Optimizing agents with better prompts and topologies. *arXiv preprint arXiv:2502.02533*.

Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, et al. 2024. Symbolic learning enables self-evolving agents. *arXiv preprint arXiv:2406.18532*.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022b. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*.

Mingchen Zhuge, Haozhe Liu, Francesco Faccio, Dylan R Ashley, Róbert Csordás, Anand Gopalakrishnan, Abdullah Hamdi, Hasan Abed Al Kader Hammoud, Vincent Herrmann, Kazuki Irie, et al. 2023. Mindstorms in natural language-based societies of mind. *arXiv preprint arXiv:2305.17066*.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*.

A More Details on Formal Definitions

Special Nodes In our formalism of compound AI systems, two special nodes play key roles:
(1) the input node $v_{\text{in}} \in V$, which receives the user query $q \in \mathcal{Q}$ as the system's entry point; and
(2) the output node $v_{\text{out}} \in V$, which terminates system execution. In our definition, the operation attached to v_{out} is the identity function I, so that it simply parses its input as the final system answer:

$$Y_{\text{out}} = I\big(X_{\text{out}}\big)\,,$$
$$a = Y_{\text{out}}\,.$$

Additionally, all outgoing conditional edges from v_{out} are set to zero ($c_{\text{out},j} = 0, \forall j$), ensuring that the system does not route its output back to other nodes. This terminal structure effectively signals computation completion, similar to how end-of-sequence tokens function in language models.

Support for Cyclic Structures Our formulation naturally accommodates both DAGs and cyclic topologies via conditional edges. Formally, a cycle at node v_i exists if and only if there is a path of length $L \geq 1$

$$\big(v_{i_0}, v_{i_1}, \ldots, v_{i_L}\big) \subseteq V$$

such that

$$v_{i_0} = v_{i_L} = v_i, \quad v_{i_t} \neq v_{\text{out}} \;\; \forall\, 0 \leq t < L,$$

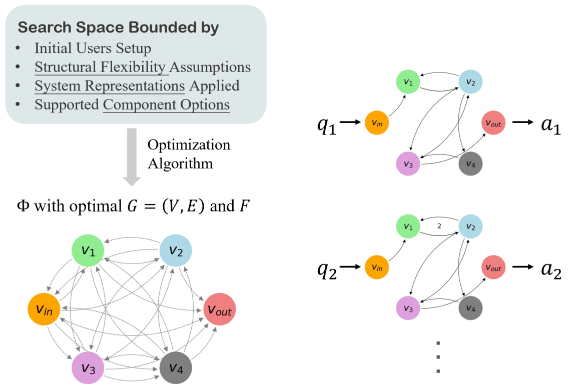
and

$$c_{i_t, i_{t+1}}(q, \tau_t) = 1 \quad \forall\, 0 \leq t < L.$$

Here, $\{v_{i_0}, \ldots, v_{i_L}\}$ denotes the sequence of nodes visited along the loop, and each $c_{i_t, i_{t+1}}(q, \tau_t) = 1$ indicates that the conditional edge from v_{i_t} to $v_{i_{t+1}}$ is active under input q and state τ_t. In other words, v_i can route its output back to itself before reaching the output node v_{out}, forming a cyclic loop.

If one would like to enforce acyclicity, it suffices to record the history of visited nodes in the state τ and disallow any transition $v_j \to v_k$ whenever v_k already appears in the history. This simple rule easily prevents the re-entry required to form a cycle.

Visualization Fig. 5 provides a visualization of system optimization under our formalism of graphs with conditional edges.



(a) Optimization Stage (b) Execution Stage

Figure 5: (a) During optimization, the algorithm explores the design space defined by user constraints and algorithmic parameters (Sec. 3.1). Although the conditional arguments in the edge matrix $E = [c_{ij}]$ are fixed once optimization completes, the actual on/off status of each c_{ij} remains undetermined. (b) At runtime, the optimized Φ instantiates different execution topologies based on $c_{ij}(\tau)$, reflecting dependence on the query input q_i and the induced contextual state τ.

B Advanced Topics and Applications

We briefly highlight several advanced topics and emerging applications in compound AI systems that may interest readers. Serving as advanced optimization methods, LLMSelector (Chen et al., 2025a) shows that selecting different LLMs at system nodes has a major effect on system performance, proposing an efficient algorithm for model selection in compound systems. Efforts have also focused on improving system execution efficiency, including network orchestration techniques (Santhanam et al., 2024) and hardware-level optimizations (Chaudhry et al., 2025). These strategies are particularly well suited for refining complex system pipelines, such as question answering with RAG or data processing with tool calling, by accounting for the run-time efficiency.

The advances in compound AI systems have also led to novel applications: SciAgents (Ghafarollahi and Buehler, 2024) leverage ontological knowledge graphs and data retrieval tools to drive automatic materials discovery that surpasses traditional human-driven research methods; AutoML-Agent (Trirat et al., 2024) devises a system to handle end-to-end AutoML pipeline, accepting user task descriptions and optional constraints and handling everything from data retrieval to model deployment; and VFlow (Wei et al., 2025) targets hardware design tasks (i.e., Verilog code generation) by augmenting AFlow (Zhang et al., 2024a) with domain-specific components, thereby optimizing system performance to achieve a higher pass@1 rate than previous methods.

Furthermore, the effectiveness of natural language feedback demonstrated by TextGrad (Yuksekgonul et al., 2025) has spurred applications in diverse domains. For example, FedTextGrad (Chen et al., 2025b) investigates the potential and challenges of integrating TextGrad into federated learning settings, where the server receives and aggregates locally optimized prompts from clients. Similarly, TPO (Li et al., 2025) proposes a method for aligning LLM preferences during inference by leveraging textual gradient signals.

C More Details of Learning Signals

Most surveyed papers leverage a single type of learning signal (i.e., either NL feedback or numerical signals) to guide the optimization of Φ. However, several works use both types of signals. For instance, GPTSwarm (Zhuge et al., 2024) employs an RL loss to numerically learn optimal connections within the swarm and then updates LLM prompts using OPRO (Yang et al., 2023), an algorithm based on NL feedback. Additionally, MaAS (Zhang et al., 2025) leverages a Bayesian Monte Carlo procedure to numerically update the probability distribution of the *agentic supernet* and uses textual gradients to refine its *operators*, including LLM prompts, temperature settings, and local node structures.

Despite the existence of these methods, we do not introduce a separate "hybrid" category for now. Instead, we classify them as either NL feedback or numerical signals based on two criteria: (1) the authors' primary design novelty—for example, GPTSwarm is categorized as numerical since OPRO is a previously designed algorithm; and (2) the signal used to update the system topology—for example, MaAS is categorized as numerical since the topology is updated via numerical signals.

To accommodate future developments, we will dynamically adjust our criteria to keep the taxonomy clear and intuitive. Introducing a "hybrid" category also remains an open option as the field evolves. We encourage future efforts that leverage both types of learning signals to leverage our proposed taxonomy and position their methods at the intersection of NL feedback and numerical signals.

D Clarification of Methods Name

Most surveyed papers introduce a single name for their method (i.e., main algorithm), while others employ multiple terms to refer to the target problem (Wang et al., 2024a; Hu et al., 2024), individual algorithms or components, or released libraries (Khattab et al., 2023). To improve readability, we assign these works a single, consistent name, chosen based on (1) its prevalence in subsequent literature and (2) its appearance in the original paper's title.