LLM-based Optimization of Compound AI Systems: A Survey

Matthieu Lin¹, Jenny Sheng¹, Andrew Zhao¹, Shenzhi Wang¹, Yang Yue¹, Yiran Wu², Huan Liu³, Jun Liu³, Gao Huang¹, Yong-Jin Liu¹

¹Tsinghua University ²Pennsylvania State University ³Xi'an Jiaotong University {yh-lin21, sqq24, zqc21, wsz21, le-y22}@mails.tsinghua.edu.cn, {liuyongjin, gaohuang}@tsinghua.edu.cn

Abstract

In a compound AI system, components such as an LLM call, a retriever, a code interpreter, or tools are interconnected. The system's behavior is primarily driven by parameters such as instructions or tool definitions. Recent advancements enable end-to-end optimization of these parameters using an LLM. Notably, leveraging an LLM as an optimizer is particularly efficient because it avoids gradient computation and can generate complex code and instructions. This paper presents a survey of the principles and emerging trends in LLM-based optimization of compound AI systems. It covers archetypes of compound AI systems, approaches to LLMbased end-to-end optimization, and insights into future directions and broader impacts. Importantly, this survey uses concepts from program analysis to provide a unified view of how an LLM optimizer is prompted to optimize a compound AI system. The exhaustive list of paper is provided at this link.

1 Introduction

A compound AI system (AI, 2024; Liu, 2022; Zaharia et al., 2024; Wu et al., 2023) (also known as Language Model Programs or LLM agent) refers to a system that integrates multiple calls to an LLM (Lu et al., 2024) as well as combines an LLM with various components such as retrievers (Lewis et al., 2020), code interpreters (Wang et al., 2024d), or tools (Zhao et al., 2024b). These systems are designed to be flexible, adapting to different tasks by simply modifying parameters such as the prompt to an LLM call (Dong et al., 2024). Notably, an LLM can interface with any endpoint that processes or generates language, enabling seamless communication with various components through natural language inputs and outputs to solve tasks.

The purpose of a compound AI system is determined by its parameters (e.g., instructions, tool definitions). For instance, we can instruct an LLM

to either use a code interpreter to reason (Gao et al., 2023b) or compose actions (Wang et al., 2024d). As a result, a significant effort is put into hand-optimizing the parameters of a compound AI system for a particular task. This challenge is exacerbated by the varying sensitivity of different LLMs to prompts (Shi et al., 2023; Mirzadeh et al., 2024). For instance, GPT-4 (OpenAI, 2024) may require explicit instructions to avoid overthinking or concerning itself with external factual accuracy when serving as a judge (Verga et al., 2024).

Recent works (Yüksekgönül et al., 2024; Cheng et al., 2024) propose to optimize the parameters of the system in an end-to-end manner using an LLM. In particular, using an LLM as an optimizer offers an efficient alternative to gradient-based (Chen et al., 2024b) and RL-based (Zhang et al., 2022) techniques. An LLM can optimize complex heterogeneous parameters such as instructions and tool implementations in a single call. Furthermore, this approach works with closed-source LLMs and avoids computing any gradient.

We call this framework LLM-based optimization or compile-time optimization (Khattab et al., 2023) of compound AI Systems. It optimizes the parameters to minimize empirical risk on a training dataset. This approach involves iteratively adjusting parameters to reduce the discrepancy between predicted and actual outputs while ensuring generalization to unseen data. Instead of manually optimizing the parameters of a compound AI system, it emphasizes on prompting the LLM optimizer to generate the parameters using the dataset.

This survey focuses on training-time optimization methods, deliberately excluding instance optimization techniques that require an expensive inference procedure to adjust the parameters per individual instance, such as debugging a specific piece of code (Chen et al., 2024f). Instead, training-time optimization amortizes this cost using an expensive training procedure, with system parameters kept

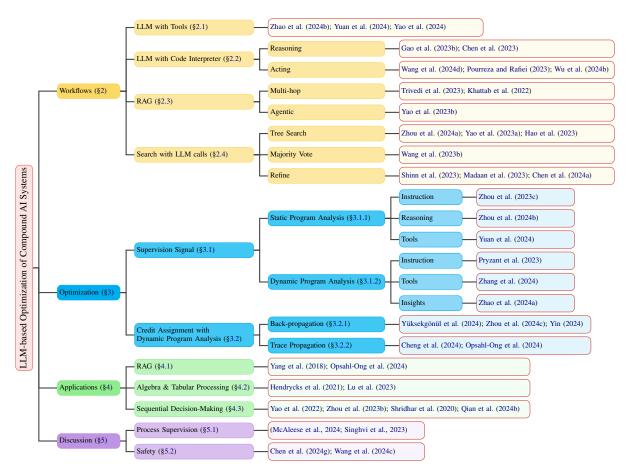


Figure 1: Organization of this survey. A non-exhaustive list of papers is provided.

fixed at inference. We also exclude architecture search (Hu et al., 2024; Saad-Falcon et al., 2024) methods that focus on discovering optimal system topologies. Multi-agent systems (Guo et al., 2024b) are omitted because their primary challenge lies in optimizing communication topologies (Zhuge et al., 2024), aligning them more closely with architecture search problems. Additionally, we do not cover methods like Mixture of Prompts (Wang et al., 2024b) that use clustering algorithms rather than LLMs for optimization.

Existing surveys primarily concentrate on LLM agents (Xi et al., 2023; Masterman et al., 2024) and their cognitive architecture (Sumers et al., 2024). Others explore LLMs in broader optimization contexts, including evolutionary algorithms, multi-objective optimization, and neural architecture search (Huang et al., 2024a). To the best of our knowledge, this paper is the first survey to explore the intersection of compound AI systems and their optimization at training time using an LLM.

Our main contribution is a framework for understanding compound AI systems and their optimization using an LLM. Notably, the main challenge is

to properly prompt/supervise the LLM optimizer to generate the parameters of a compound AI system. This survey draws an analogy from program analysis to understand the prompt to the LLM optimizer. In particular, we either prompt the LLM with information to analyze the system without executing the runtime behavior of the compound AI system (static) or based on observations obtained during the system's runtime behavior given input data (dynamic). Sec. 3.1 focuses on compound AI systems involving a single parameter where static program analysis or dynamic program analysis are used. Then, Sec. 3.2 extends to prompting the LLM to co-optimize multiple parameters using a dynamic program analysis approach to implement backpropagation (Karpathy, 2020) and trace propagation (Wikipedia, 2024).

We organize our paper as shown in Fig. 1. We explore existing archetypes within compound AI systems in Sec. 2. We discusse how the LLM-based optimizer is prompted in Sec. 3. Then, examples of LLM-based optimization of compound AI systems are shown in Sec. 4. Finally, Sec. 5 discusses the broader impact of this framework.

2 Workflow: Design of a Compound AI System

We introduce compound AI system archetypes with a focus on their parameters to facilitate the understanding of their optimization, as outlined in Sec. 3. These archetypes are presented from a system's perspective, illustrating how various components can be combined to perform a wide range of tasks. Notably, we avoid task-based categorization, as the same system can address multiple tasks through different parameterizations, making such categorization less clear-cut.

2.1 LLM with Tools

We can extend an LLM's functionality beyond generating text by instructing it to generate outputs in structured formats like JSON, which can then be used to call APIs (Yao et al., 2024; Huang et al., 2024b) or execute functions (Zhao et al., 2024b; Zhang et al., 2024). In particular, optimizing the instruction involves teaching the LLM when and how to use the tools. Meanwhile, the execution of functions can be improved by refining the set of tools available to the LLM as well as their implementation (Zhao et al., 2024b; Zhang et al., 2024).

2.2 LLM with Code Interpreter

LLMs can generate executable code as part of their output, allowing them to efficiently express complex reasoning (Chen et al., 2023; Gao et al., 2023b; Wu et al., 2024a) and actions (Wang et al., 2024d). Code interpreters are responsible for executing this code, enabling the system to solve tasks that require both high-level reasoning and detailed computational steps. For example, an LLM can generate Python code that calls multiple tools in a structured way, solving the task in fewer steps than would be possible using tools call (Wang et al., 2024d). Besides Python, LLMs can also write SQL queries to interact with SQL databases (Pourreza and Rafiei, 2023; Wang et al., 2023a; Wu et al., 2024b) or interact with Bash shells (Yang et al., 2024b; Lin et al., 2018). The use of code interpreters decouples algebraic computation from reasoning tasks, enhancing system performance (Gao et al., 2023b). As a result, it involves designing the instruction to teach the LLM how to use the code interpreter. Additionally, when the code interpreter invokes multiple tools in a structured manner, the implementation of these tools can also be optimized (Zhao et al., 2024b).

2.3 Retrieval Augmented Generation

Retrieval-augmented generation (RAG) (Lewis et al., 2020; Izacard and Grave, 2021; Ram et al., 2023; Anthropic, 2024; Jiang et al., 2020) integrates LLMs with retrievers to generate answers grounded on the retrieved information. A typical RAG system requires crafting an instruction for an LLM to query the retriever in a fixed amount of steps or a dynamic amount of steps, as well as an instruction for an LLM to use the retrieved information to generate a response. In setups like HotpotQA (Yang et al., 2018), the LLM queries a retriever to gather relevant information from multiple sources, which is then used to generate a response. RAG systems follow two approaches: the agentic setup (Yao et al., 2023b), where the LLM decides what to retrieve over a dynamic number of steps, and the multi-hop setup (Trivedi et al., 2023; Press et al., 2023; Khattab et al., 2022; Zhang et al., 2023; Shao et al., 2023), where information is gathered over a fixed number steps, with each step's output informing the next. Both approaches allow the system to accumulate knowledge iteratively and refine its final answer. To query information from multiple sources, the LLM calls use the same instruction; at each step, the information is simply added into the context. Furthermore, to improve the query to the retriever, prior work (Gao et al., 2023a; Siriwardhana et al., 2022) asks the LLM to generate additional context such as background information (Siriwardhana et al., 2022) or a hypothetical response (Gao et al., 2023a).

2.4 Search with LLM calls

Increasing the number of calls to an LLM exhibits scaling behaviors (Chen et al., 2024c; Snell et al., 2024). A straightforward approach, selfconsistency (Wang et al., 2023b), samples multiple i.i.d answers with a CoT prompt (Wei et al., 2022) to search the space of answers. Given multiple answers, the final answer is selected based on majority voting. When chained (Zhou et al., 2023a; Yao et al., 2023a), each LLM call adheres to the same instruction, incrementally appending new content to the context at each step. Other works have proposed to search for the best action (Zhou et al., 2024a) or reasoning sequence (Yao et al., 2023a). They rely on a tree search algorithm (Yao et al., 2023a; Hao et al., 2023; Wang et al., 2024a; Chen et al., 2024h). In particular, they define two instructions: a generator LLM and an evaluator LLM. The

generator LLM proposes multiple solutions, while an evaluator LLM scores candidates. Then, techniques like breadth-first search (BFS) (Yao et al., 2023a) or Monte Carlo tree search (MCTS) (Zhou et al., 2024a) are employed to explore different branches of possible solutions.

3 LLM-based Optimizer

The optimization of a compound AI system using an LLM is a black-box optimization problem. The optimization requires strategies that balance exploration (searching for promising parameter updates) and exploitation (refining promising parameter updates). An LLM-based optimizer for compound AI systems leverages the generative capabilities of the LLM to optimize multiple interdependent parameters within these systems. Thus, in practice, different methods differ in how they prompt the LLM optimizer to refine these parameters.

Sec. 3.1 starts by drawing an analogy with program analysis to study how an LLM can be prompted to optimize a parameter. In static program analysis, the prompt to the optimizer considers the compound AI system without executing it. In contrast, in dynamic program analysis, the optimizer's prompt observes its runtime behavior, which includes a textual feedback. Thus, we refer to the static or dynamic strategy as the supervision signal of the optimization process. Compound AI systems involving multiple dependent parameters result in complex programs that are hard to analyze without executing them. As a result, Sec. 3.2 presents how dynamic program analysis is extended to co-optimize parameters. As shown in Fig. 2, it implements a credit assignment algorithm where the recorded computational graph at execution is used to perform backpropagation or trace propagation.

3.1 Supervision Signal

We draw an analogy with program analysis to understand how the LLM is prompted to perform optimization on a parameter. We refer to it as the supervision signal of the optimization process since it informs the LLM about the objective function. In particular, the prompt takes two forms. In static program analysis, we provide the LLM with input-output pairs from the training data and ask the LLM to synthesize the parameter corresponding to the task. Instead, dynamic program analysis prompts the LLM with the runtime behavior of the system,

which provides a qualitative critique on its output. Then, the LLM reasons on the update of the parameter with respect to that textual critique.

3.1.1 Static Program Analysis

Traditionally, static analysis refers to examining the program's code without executing it. It involves techniques like type checking and data flow analysis to optimize code. Similarly, we can prompt LLMs to generate the parameters purely based on the problem's specifications. For instance, we provide the LLM optimizer with input-output pairs corresponding to a task and prompt it to generate instructions corresponding to that task. Since previous work has observed scaling behavior in generating more solutions (Snell et al., 2024; Chen et al., 2024c), we can optionally generate multiple solutions from the optimizer and evaluate each parameter's performance on an evaluate set to select the best one. Below, we summarize different approaches for synthesizing instruction (Zhou et al., 2023c; Schnabel and Neville, 2024; Yang et al., 2024a), reasoning structure (Zhou et al., 2024b), and tools (Yuan et al., 2024).

To optimize instructions, APE (Zhou et al., 2023c) begins by prompting the LLM with a task description represented as input-output samples to generate an initial set of instructions. Random variations of these prompts are created by prompting an LLM to generate semantically similar instructions. Each variation is scored, and the highest-scoring versions are retained for further iterations. The process continues until the performance plateaus and the highest-scoring instruction is selected as the final instruction. Later works propose variations based on evolutionary methods (Guo et al., 2024a; Fernando et al., 2024); the LLM combines parts of multiple prompts and then introduces random changes; we call this process crossover and mutation. To introduce fine-grained variations of instructions (Prasad et al., 2023), SAMMO (Schnabel and Neville, 2024) uses a more granular representation of prompts by breaking them down into smaller sections, such as instructions or input formatting. This fine-grained structure allows SAMMO to adjust specific components, such as changing the input format from raw text to a structured format (e.g., XML).

Beyond instructions, Self-Discover (Zhou et al., 2024b) generates a reasoning structure specific to a task, and Craft (Yuan et al., 2024) synthesizes tools specific to a problem instance. Self-Discover

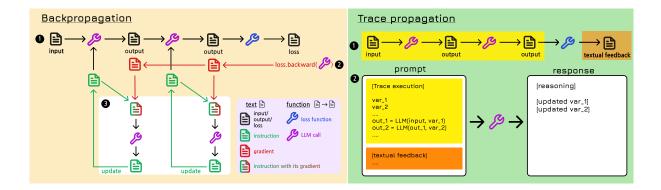


Figure 2: **Credit assignment:** a local vs. a global approach. In backpropagation, the optimizer updates each parameter individually. In trace propagation, the prompt contains the execution trace, which allows it to generate all updated variables in a single call. Note that the loss is a textual feedback. In addition, the gradient of the instruction is *not* the gradient of the output, but the gradient of the instruction with respect to the gradient of the output.

(Zhou et al., 2024b) defines a reasoning structure using a JSON format with keys representing distinct reasoning steps. Given a task description, the LLM optimizer selects, adapts, and implements reasoning steps. Craft (Yuan et al., 2024) generates tools that solve specific input-output pairs, validate their accuracy, and abstract the solutions into reusable functions. These reusable tools are then further refined to remove duplicates and improve generalization across tasks.

3.1.2 Dynamic Program Analysis

Dynamic program analysis refers to executing the program and observing its runtime behavior to analyze how the program performs. Similarly, we analyze how a compound AI system performs on the training data with a textual feedback. It provides qualitative or quantitative assessments that critique prediction with the desired output. This process involves running the input data through the compound AI system, which makes a prediction and then compare the prediction with the groundtruth. When the ground-truth is not available, we use a self-supervised approach and ask an LLM to provide a self-critique (e.g., in creative writing (Zhou et al., 2024c)). By iterating over data points, the goal is then to adjust the parameters so that the predictions get closer to the target values.

Given a textual feedback, the LLM-based optimizer reasons about a potential update and updates the parameter accordingly. In this approach, we iteratively sample data points and evaluate the system's output. We then provide a textual feedback such as execution errors, numerical values, or qualitative assessments that critique prediction with the desired output. Finally, the update of a parameter

is done in a single LLM call (Cheng et al., 2024) or multiple LLM call (Yüksekgönül et al., 2024). As shown in Fig. 2, predicting a direction of update in textual form and generating the update can be done separately (Yüksekgönül et al., 2024; Yin, 2024; Zhou et al., 2024c; Ma et al., 2024; Pryzant et al., 2023; Wang et al., 2024e) or in a single LLM call (Cheng et al., 2024; Zhang et al., 2024; Zhao et al., 2024b; Yang et al., 2024a; Zhao et al., 2024a). To avoid overfitting to a single sample, we can summarize multiple feedback (Chen et al., 2024e) or direction of updates (Austin and Chartock, 2024). The main challenge in supervising the optimization with textual feedback is that LLMs struggle to find the failure of the target model. This is further exacerbated by the target model's ability to follow an instruction (Ma et al., 2024). Thus, current works focus on approaches to circumvent this challenge.

Methods such as ProTeGi (Pryzant et al., 2023) and PromptAgent (Wang et al., 2024e) propose to search through the space of instructions. At each step, they sample a data point and propose multiple instructions. ProTeGi (Pryzant et al., 2023) uses a beam search approach where the best instructions are selected based on performance across a validation set. PromptAgent (Wang et al., 2024e) proposes a more expensive approach based on MCTS to explore different optimization paths. Each step of the optimization is treated as an action in the tree, and the optimizer navigates through different potential updates, selecting the path with the highest reward. The final solution is chosen based on the best path and the prompt with the highest score along that path. ABO (Ma et al., 2024) adopts a simpler approach by implementing the instruction as step-by-step guidelines. This allows the optimizer to identify the specific step that failed, resulting in more targeted and effective optimization.

OPRO (Yang et al., 2024a) proposes variations based on a history of past instructions and their score on the validation set. It begins with an initial instruction with its score. Then, at each step, an LLM is prompted to create an instruction with a higher score using previous instructions, their scores, and a random sample of input-output pairs from the training data.

Beyond optimizing instruction, AgentOptimizer (Zhang et al., 2024) and LearnAct (Zhao et al., 2024b) optimize tools in compound AI systems. Starting from an initial set of tools, they iterate over training data, modifying or refining the tools based on their execution history and success rate. LearnAct goes further by optimizing both the tools and the instructions guiding their use, allowing for the co-optimization of actions and strategies.

Additionally, ExpeL (Zhao et al., 2024a) proposes to optimize a list of insights to ground LLMs in their environment in sequential decision-making. We refer readers to Sec. 4.3 for more information.

Beyond a qualitative critique of the output, Trace (Cheng et al., 2024) provides execution errors as textual feedback. This is practical because it means if there is any error at runtime, it can get feedback from it.

3.2 Credit Assignment with Dynamic Program Analysis

Compound AI systems involving multiple dependent parameters result in complex programs. Specifically, it is challenging to craft a prompt that provides an accurate overview of the runtime behavior. Therefore, no work has yet addressed static program analysis for performing credit assignment when co-optimizing multiple parameters.

In end-to-end optimization of compound AI systems, credit assignment refers to evaluating the contribution of each parameter with respect to the textual feedback. In particular, dynamic program analysis allows us to record the computational graph at runtime and use it to perform credit assignment. Based on the recorded graph, backpropagation uses the chain rule to compute a gradient for each variable, and trace propagation transforms the computational graph into a textual representation akin to a Python traceback.

3.2.1 Backpropagation

In practice, the implementation of backpropagation for compound AI systems (Yüksekgönül et al., 2024; Yin, 2024; Zhou et al., 2024c) follows that of deep learning frameworks (Karpathy, 2020). Thus, we draw an analogy with deep neural network training to understand its implementation. We replace tensors with text (e.g., natural language, code). As shown in Fig. 2, functions such as the loss function return textual feedback, parameters can be instructions or tools implementation, and gradients are textual variables that inform on ways to improve that variable.

In practice, it involves the implementation of a variable class that wraps any computation done on text. This variable keeps track of its predecessors and its gradients. By composing variables and functions into a forward pass, we can compute the loss and call the backward pass on the loss variable. As illustrated in Fig. 2, the backward pass follows the chain rule: it computes the gradient at each function in the reverse order of the forward pass.

To illustrate how the backward pass computes the gradient for each parameter, let us use the example in Fig. 3.2. The example chains two LLM calls (i.e., two trainable instructions) and a loss function. It starts by computing the gradient of the last output with respect to the loss. An LLM (serving as the autograd engine) is prompted to suggest how "output" can be updated to reduce the error in "loss." This "output" has two parent variables: an "instruction" and the "output" from the previous LLM call. We obtain the gradient of the instruction by prompting the LLM to suggest how to improve "instruction" to address the gradient of the output (and not the output; this is equivalent to computing the gradient directly with respect to the loss as in the chain rule in calculus). We compute the gradient of the "output" of the first LLM call given the gradient of the other "output." Then, for the first "instruction," it takes the gradient of "output." Thus, backpropagation requires calling multiple times an LLM to compute each gradient.

After the backward pass, the LLM optimizer iterates over each trainable variable to compute an updated variable based on its value and gradient.

One caveat of this approach is the potential for error accumulation (Yüksekgönül et al., 2024; Yin, 2024; Zhou et al., 2024c), as the optimal gradient direction is unknown. For example, if the gradient with respect to the loss points in an erroneous di-

rection, subsequent updates to all other parameters will follow this incorrect path.

3.2.2 Trace propagation

Trace propagation (Cheng et al., 2024) refers to recording a textual representation of the computational graph akin to a Python traceback at runtime. In particular, it involves a propagation process to obtain the computational graph. When calling "backward" on the textual feedback, it records every parent variable involved in the computation of that textual feedback. Given the textual feedback and the execution trace (akin to a Python traceback), the LLM optimizer is prompted to improve the (trainable) variables of the execution trace according to the textual feedback. The response contains a reasoning field where the LLM reasons about which variable is responsible for the textual feedback. Thus, the LLM does not necessarily generate updates for all variables. MIPROv2 (Opsahl-Ong et al., 2024) shares similarity with Trace (Cheng et al., 2024); it executes the system to gather examples of runtime behavior. These examples are given to the LLM optimizer to synthesize instructions. In particular, MIPROv2 does not iteratively refine the synthesized instruction based on textual feedback.

4 Applications

In this section, we provide examples of tasks addressed through training time LLM-based optimization of compound AI systems. Specifically, we present case studies involving mathematical problem-solving, question-answering, and sequential decision-making. These involve multiple instructions optimization (Sec. 4.1), designing tools (Sec. 4.2), and learning insights from sequential decision-making (Sec. 4.3).

4.1 Optimizing RAG Pipeline for Question-Answering Tasks

HotpotQA (Yang et al., 2018) involves retrieving multiple relevant documents from a Wikipedia API (Yao et al., 2023b) to answer questions. It queries an LLM to generate a search query to retrieve documents alongside follow-up queries using previously retrieved documents as additional context. Then, it queries the LLM to generate a response based on the retrieved documents. In MIPROv2 (Opsahl-Ong et al., 2024), few-shot examples are initially bootstrapped by running the model on the training data. These examples are then fed into the optimizer's prompt to generate instructions. Since the

instructions are synthesized, the performance of the system with the combined few-shot examples and synthesized instructions is unknown. To address this, a Bayesian model (Akiba et al., 2019) is employed to search for the optimal combination of few-shot examples and instructions based on performance on a validation set. Unlike random search, a Bayesian model balances exploitation and exploration of the best combination.

4.2 Optimizing Tools for Algebra and Tabular Processing

Tabular Math Word Problems (TabMWP) (Lu et al., 2023) requires multi-hop mathematical reasoning across both textual and tabular data, with tables varying in structure, format, and layout. Each sample is composed of a table and a related problem in natural language. LLMs are tested on their ability to interpret natural language, extract relevant information from tables, and perform necessary calculations. Prior work proposes to optimize a set of tools to decouple numerical calculation (Hendrycks et al., 2021) and data processing of tables from reasoning. In particular, AgentOptimizer (Zhang et al., 2024) starts with a Python interpreter. Then, it iterates over the training data and attempts to solve the task. Given the execution history and success rate, the LLM optimizer decides to add, revise, or remove tools from the set of available tools.

4.3 Optimizing Knowledge for Sequential Decision-making

Sequential decision-making tasks (Yao et al., 2022; Zhou et al., 2023b; Shridhar et al., 2020; Qian et al., 2024b) involve an LLM interacting with an environment, making decisions based on observations to accomplish specific objectives. Optimizing prompts for LLMs in sequential decision-making requires grounding them to the environment. ExpeL (Zhao et al., 2024a) enhances this process by optimizing a list of insights generated from different trajectories. These trajectories consist of a sequence of actions and observations. During inference, this list is incorporated into the prompt of a ReAct policy (Yao et al., 2023b), providing additional grounding in the environment. They use a Reflexion (Shinn et al., 2023) policy to collect trajectory pairs. In particular, the Reflexion policy allows the LLM to retry the tasks upon failing, which allows the collection of unsuccessful and successful trajectories on the same task instance. These trajectories are used to optimize insights related to

a specific environment. Given a batch of trajectories, either k successful trajectories from different task instances or a successful and unsuccessful trajectory pair, the LLM optimizers modify the list of insights either by adding, editing, or removing insights. Variants of ExpeL include AutoGuide (Fu et al., 2024b), which selectively chooses which insights to add to the context window at each state, AutoManual (Chen et al., 2024d), which goes online by using insights with the behavior policy, and MSI (Fu et al., 2024a; Qian et al., 2024a) which proposes to categorize each insight.

5 Discussion

In this section, we study the broader impact of LLM-based optimization in compound AI systems. Sec. 5.1 discuss how compound AI systems facilitate the implementation of process supervision. In turn, Sec. 5.2 shows how it improves the controllability and interpretability of LLMs.

5.1 Process supervision

Compound AI systems contain interpretable parameters where intermediate results are interpretable. Moreover, supervision extends beyond a single scalar (Cheng et al., 2023), and constraints go beyond simple algebraic forms (Singhvi et al., 2023). This allows for the introduction of richer training signals using an LLM (McAleese et al., 2024).

In AI training pipelines, feedback is often provided only after the final output is generated, limiting the model's understanding of how intermediate steps contribute to success or failure. Compound AI systems, however, enable a much more nuanced approach as intermediate outputs are interpretable. This introduces the opportunity to provide targeted signals (Singhvi et al., 2023) at various stages of the reasoning or problem-solving process.

5.2 Safety

One of the major safety concerns in using LLMs (Anthropic, 2024; OpenAI, 2024) stems from their unpredictable emergent behavior during training. These behaviors arise from both the scale of the model and the objectives it is optimized for. Typically, LLMs are trained with RL-like objectives, which can be prone to issues like reward overoptimization. This can lead to unintended and potentially unsafe outcomes. Instead, compound AI systems, which consist of multiple components and interpretable parameters, offer a way to mitigate

these risks. They can use smaller specialized language models as well as decompose complex tasks into subtasks (Xia et al., 2024).

When a system breaks a task (Xia et al., 2024; Lu et al., 2024) into smaller, manageable pieces, it becomes easier for humans to track the decisionmaking process and understand how the system arrives at its conclusions. By contrast, if a large model attempts to solve the entire task in a single step, its internal reasoning processes may be hidden within the neural network's weights, leaving human operators with little insight into how the task was solved. However, it is important to note that while compound AI systems may have more predictable behavior, they suffer from novel types of attacks. For instance, AgentPoison (Chen et al., 2024g) has introduced adversarial attacks targeting long-term memory or RAG knowledge bases; furthermore, Wang et al. (2024c) have identified the prevalent deceptions and misinformation in multi-agent scenarios that could mislead LLMs.

6 Conclusion

We conclude this survey with an intuitive overview of the content covered. First, we explored various archetypes of compound AI systems, focusing on their components and parameters. Subsequently, we drew analogies with program analysis to understand different strategies to prompt the optimizer with the objective function, which we referred to as the supervision signal. To address the credit assignment problem, we showed how backpropagation and trace propagation uses the computational graph recorded in dynamic program analysis. Finally, we discussed how the works presented in this survey positively impact safety and process supervision.

Beyond the prompt of the LLM optimizer, we believe that unlocking the optimization of more complex compound AI systems will require training an LLM (McAleese et al., 2024) to provide accurate and rich textual feedback (Lightman et al., 2023). Textual feedback requires interpretation and reasoning, helping the optimizer better understand the task and improve output quality rather than exploiting superficial patterns (Ouyang et al., 2022). Additionally, using open-source models offers an opportunity to finetune the underlying LLMs of the system (Soylu et al., 2024; Putta et al., 2024; Rafailov et al., 2023).

7 Limitations

This survey on optimizing compound AI systems has two key limitations. First, the focus is exclusively on language-based systems, overlooking multimodal approaches like vision-language models that are becoming increasingly relevant in complex AI applications (Pan et al., 2024). This restriction limits the generalizability of the findings to broader AI systems that integrate different modalities. Second, the survey deliberately avoids discussing techniques such as architecture search (Saad-Falcon et al., 2024; Hu et al., 2024) or metaoptimizers (Yin et al., 2024; Zelikman et al., 2023), which are powerful tools for optimizing the design of the system and optimizer performance. As a result, this survey may not capture the full scope of optimization methods that could enhance compound AI systems.

References

- LangChain AI. 2024. Langchain studio: A platform for building language model applications. https: //studio.langchain.com/. Accessed: 2024-10-11
- Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
- Anthropic. 2024. Claude. https://www.anthropic.com/claude. Accessed: Month, Day, 2024.
- Anthropic. 2024. Introducing contextual retrieval. https://www.anthropic.com/news/contextual-retrieval. Accessed: 2024-10-01.
- Derek Austin and Elliott Chartock. 2024. Gradsum: Leveraging gradient summarization for optimal prompt engineering. *Preprint*, arXiv:2407.12865.
- Justin Chih-Yao Chen, Archiki Prasad, Swarnadeep Saha, Elias Stengel-Eskin, and Mohit Bansal. 2024a. Magicore: Multi-agent, iterative, coarse-to-fine refinement for reasoning. *Preprint*, arXiv:2409.12147.
- Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. 2024b. Instructzero: Efficient instruction optimization for black-box large language models. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.
- Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. 2024c. Are more llm calls all you need? towards scaling laws of compound inference systems. *Preprint*, arXiv:2403.02419.

- Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. 2024d. Automanual: Generating instruction manuals by llm agents via interactive environmental learning. *Preprint*, arXiv:2405.16247.
- Weizhe Chen, Sven Koenig, and Bistra Dilkina. 2024e. Reprompt: Planning by automatic prompt engineering for large language models agents. *Preprint*, arXiv:2406.11132.
- Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Trans. Mach. Learn. Res.*, 2023.
- Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024f. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*.
- Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. 2024g. Agentpoison: Red-teaming LLM agents via poisoning memory or knowledge bases. *CoRR*, abs/2407.12784.
- Ziru Chen, Michael White, Raymond J. Mooney, Ali Payani, Yu Su, and Huan Sun. 2024h. When is tree search useful for LLM planning? it depends on the discriminator. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 13659–13678. Association for Computational Linguistics.
- Ching-An Cheng, Andrey Kolobov, Dipendra Misra, Allen Nie, and Adith Swaminathan. 2023. Llf-bench: Benchmark for interactive learning from language feedback. *Preprint*, arXiv:2312.06853.
- Ching-An Cheng, Allen Nie, and Adith Swaminathan. 2024. Trace is the new autodiff unlocking efficient optimization of computational workflows. *CoRR*, abs/2406.16218.
- Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A survey on in-context learning. *Preprint*, arXiv:2301.00234.
- Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2024. Promptbreeder: Self-referential self-improvement via prompt evolution. In Fortyfirst International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024. OpenReview.net.
- Dayuan Fu, Biqing Qi, Yihuai Gao, Che Jiang, Guanting Dong, and Bowen Zhou. 2024a. Msi-agent: Incorporating multi-scale insight into embodied agents for superior planning and decision-making. *arXiv* preprint arXiv:2409.16686.

- Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024b. Autoguide: Automated generation and selection of state-aware guidelines for large language model agents. *arXiv preprint arXiv:2403.08978*.
- Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2023a. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023, pages 1762–1777. Association for Computational Linguistics.
- Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023b. PAL: program-aided language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR.
- Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2024a. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. 2024b. Large language model based multi-agents: A survey of progress and challenges. *Preprint*, arXiv:2402.01680.
- Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 8154–8173. Association for Computational Linguistics.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.
- Shengran Hu, Cong Lu, and Jeff Clune. 2024. Automated design of agentic systems. *CoRR*, abs/2408.08435.
- Sen Huang, Kaixiang Yang, Sheng Qi, and Rui Wang. 2024a. When large language model meets optimization. *Swarm Evol. Comput.*, 90:101663.
- Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. 2024b. Metatool benchmark for large language models: Deciding whether to use tools and which to use. In *The Twelfth*

- International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net.
- Gautier Izacard and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 23, 2021*, pages 874–880. Association for Computational Linguistics.
- Yichen Jiang, Shikha Bordia, Zheng Zhong, Charles Dognin, Maneesh Kumar Singh, and Mohit Bansal. 2020. Hover: A dataset for many-hop fact extraction and claim verification. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 3441–3460. Association for Computational Linguistics.
- Andrej Karpathy. 2020. Micrograd: A tiny autograd engine. Accessed: 2024-10-11.
- Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. 2022. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive NLP. *CoRR*, abs/2212.14024.
- 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. *CoRR*, abs/2310.03714.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.
- Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *Preprint*, arXiv:2305.20050.
- Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D Ernst. 2018. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. *arXiv* preprint *arXiv*:1802.08979.
- Jerry Liu. 2022. LlamaIndex.
- Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024. The AI Scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*.

- Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2023. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. In *The Eleventh International Conference on Learning Representations, ICLR* 2023, *Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.
- Ruotian Ma, Xiaolei Wang, Xin Zhou, Jian Li, Nan Du, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. Are large language models good prompt optimizers? *CoRR*, abs/2402.02101.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 16, 2023.
- Tula Masterman, Sandi Besen, Mason Sawtell, and Alex Chao. 2024. The landscape of emerging AI agent architectures for reasoning, planning, and tool calling: A survey. *CoRR*, abs/2404.11584.
- Nat McAleese, Rai Michael Pokorny, Juan Felipe Ceron Uribe, Evgenia Nitishinskaya, Maja Trebacz, and Jan Leike. 2024. LLM critics help catch LLM bugs. *CoRR*, abs/2407.00215.
- Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *Preprint*, arXiv:2410.05229.
- OpenAI. 2024. Chatgpt. https://openai.com/chatgpt. Accessed: Month, Day, 2024.
- 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. *CoRR*, abs/2406.11695.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 December 9, 2022.
- Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. 2024. Autonomous evaluation and refinement of digital agents. CoRR, abs/2404.06474.

- Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: decomposed in-context learning of text-to-sql with self-correction. In Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 16, 2023.
- Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2023. Grips: Gradient-free, edit-based instruction search for prompting large language models. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2023, Dubrovnik, Croatia, May 2-6, 2023*, pages 3827–3846. Association for Computational Linguistics.
- Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 5687–5711. Association for Computational Linguistics.
- Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 7957–7968. Association for Computational Linguistics.
- Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. 2024. Agent Q: advanced reasoning and learning for autonomous AI agents. *CoRR*, abs/2408.07199.
- Chen Qian, Yufan Dang, Jiahao Li, Wei Liu, Zihao Xie, Yifei Wang, Weize Chen, Cheng Yang, Xin Cong, Xiaoyin Che, Zhiyuan Liu, and Maosong Sun. 2024a. Experiential co-learning of software-developing agents. *Preprint*, arXiv:2312.17025.
- Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024b. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 15174–15186. Association for Computational Linguistics.
- 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. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 16, 2023.*

- Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *Trans. Assoc. Comput. Linguistics*, 11:1316–1331.
- Jon Saad-Falcon, Adrian Gamarra Lafuente, Shlok Natarajan, Nahum Maru, Hristo Todorov, Etash Guha, E. Kelly Buchanan, Mayee Chen, Neel Guha, Christopher Ré, and Azalia Mirhoseini. 2024. Archon: An architecture search framework for inference-time techniques. *Preprint*, arXiv:2409.15254.
- Tobias Schnabel and Jennifer Neville. 2024. Prompts as programs: A structure-aware approach to efficient compile-time prompt optimization. *CoRR*, abs/2404.02319.
- Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *Findings of the Association for Computational Linguistics: EMNLP* 2023, *Singapore, December* 6-10, 2023, pages 9248–9274. Association for Computational Linguistics.
- Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 16, 2023.
- Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *ArXiv*, abs/2010.03768.
- Arnav Singhvi, Manish Shetty, Shangyin Tan, Christopher Potts, Koushik Sen, Matei Zaharia, and Omar Khattab. 2023. Dspy assertions: Computational constraints for self-refining language model pipelines. *CoRR*, abs/2312.13382.
- Shamane Siriwardhana, Rivindu Weerasekera, Elliott Wen, Tharindu Kaluarachchi, Rajib Rana, and Suranga Nanayakkara. 2022. Improving the domain adaptation of retrieval augmented generation (RAG) models for open domain question answering. *CoRR*, abs/2210.02627.
- Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *Preprint*, arXiv:2408.03314.

- Dilara Soylu, Christopher Potts, and Omar Khattab. 2024. Fine-tuning and prompt optimization: Two great steps that work better together. *Preprint*, arXiv:2407.10930.
- Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas Griffiths. 2024. Cognitive architectures for language agents. *Transactions on Machine Learning Research*. Survey Certification.
- Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL 2023, Toronto, Canada, July 9-14, 2023, pages 10014–10037. Association for Computational Linguistics.
- Pat Verga, Sebastian Hofstätter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick S. H. Lewis. 2024. Replacing judges with juries: Evaluating LLM generations with a panel of diverse models. *CoRR*, abs/2404.18796.
- Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023a. MAC-SQL: A multi-agent collaborative framework for text-to-sql. *CoRR*, abs/2312.11242.
- Evan Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, Will Song, Vaskar Nath, Ziwen Han, Sean Hendryx, Summer Yue, and Hugh Zhang. 2024a. Planning in natural language improves llm search for code generation. *Preprint*, arXiv:2409.03733.
- Ruochen Wang, Sohyun An, Minhao Cheng, Tianyi Zhou, Sung Ju Hwang, and Cho-Jui Hsieh. 2024b. One prompt is not enough: Automated construction of a mixture-of-expert prompts. In *Forty-first International Conference on Machine Learning, ICML* 2024, *Vienna, Austria, July* 21-27, 2024. OpenReview.net.
- Shenzhi Wang, Chang Liu, Zilong Zheng, Siyuan Qi, Shuo Chen, Qisen Yang, Andrew Zhao, Chaofei Wang, Shiji Song, and Gao Huang. 2024c. Boosting llm agents with recursive contemplation for effective deception handling. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 9909–9953.
- Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024d. Executable code actions elicit better LLM agents. In Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024. OpenReview.net.
- Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P. Xing, and Zhiting Hu. 2024e. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *The Twelfth International Conference on Learning Representations, ICLR* 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net.

- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023.* OpenReview.net.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 December 9, 2022.
- Wikipedia. 2024. Tracing (software) Wikipedia, The Free Encyclopedia. [Online; accessed 10-October-2024].
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multiagent conversation framework. *arXiv preprint arXiv:2308.08155*.
- Yiran Wu, Feiran Jia, Shaokun Zhang, Hangyu Li, Erkang Zhu, Yue Wang, Yin Tat Lee, Richard Peng, Qingyun Wu, and Chi Wang. 2024a. Mathchat: Converse to tackle challenging math problems with LLM agents. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.
- Yiran Wu, Tianwei Yue, Shaokun Zhang, Chi Wang, and Qingyun Wu. 2024b. Stateflow: Enhancing llm task-solving through state-driven workflows. *arXiv* preprint arXiv:2403.11322.
- Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. 2023. The rise and potential of large language model based agents: A survey. *CoRR*, abs/2309.07864.
- Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2024. Agentless: Demystifying llm-based software engineering agents. *CoRR*, abs/2407.01489.
- Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024a. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024.* OpenReview.net.
- John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. 2024b. Intercode: Standardizing and benchmarking interactive coding with execution

- feedback. Advances in Neural Information Processing Systems, 36.
- Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. arXiv preprint arXiv:1809.09600.
- Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable realworld web interaction with grounded language agents. *ArXiv*, abs/2207.01206.
- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ -bench: A benchmark for toolagent-user interaction in real-world domains. CoRR, abs/2406.12045.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 16, 2023.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023.* OpenReview.net.
- Li Yin. 2024. AdalFlow: The Library for Large Language Model (LLM) Applications.
- Xunjian Yin, Xinyi Wang, Liangming Pan, Xiaojun Wan, and William Yang Wang. 2024. Gödel agent: A self-referential agent framework for recursive self-improvement. *Preprint*, arXiv:2410.04444.
- Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi Fung, Hao Peng, and Heng Ji. 2024. CRAFT: customizing llms by creating and retrieving from specialized toolsets. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11*, 2024. OpenReview.net.
- Mert Yüksekgönül, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. Textgrad: Automatic "differentiation" via text. *CoRR*, abs/2406.07496.
- 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/.
- Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. 2023. Self-taught optimizer (STOP): recursively self-improving code generation. *CoRR*, abs/2310.02304.

- Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. Repocoder: Repository-level code completion through iterative retrieval and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 2471–2484. Association for Computational Linguistics.
- Shaokun Zhang, Jieyu Zhang, Jiale Liu, Linxin Song, Chi Wang, Ranjay Krishna, and Qingyun Wu. 2024. Offline training of language model agents with functions as learnable weights. In Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024. OpenReview.net.
- Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E. Gonzalez. 2022. TEM-PERA: test-time prompting via reinforcement learning. *CoRR*, abs/2211.11890.
- Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024a. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.
- Haiteng Zhao, Chang Ma, Guoyin Wang, Jing Su, Lingpeng Kong, Jingjing Xu, Zhi-Hong Deng, and Hongxia Yang. 2024b. Empowering large language model agents through action learning. *CoRR*, abs/2402.15809.
- Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024a. Language agent tree search unifies reasoning, acting, and planning in language models. In Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024. OpenReview.net.
- Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. 2023a. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5*, 2023. OpenReview.net.
- Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. 2024b. Self-discover: Large language models self-compose reasoning structures. *Preprint*, arXiv:2402.03620.
- Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. 2023b. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.
- Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, Huajun

- Chen, and Yuchen Eleanor Jiang. 2024c. Symbolic learning enables self-evolving agents. *CoRR*, abs/2406.18532.
- Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023c. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023.* OpenReview.net.
- 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*.