LLMSELECTOR: Learning to Select Models in Compound AI Systems

Lingjiao Chen [1] **Jared Q. Davis** [2] **Boris Hanin** [3] **Peter Bailis** [1] **Matei Zaharia** [4] **James Zou** [1] **Ion Stoica** [4]

Abstract

Compound AI systems that combine multiple LLM calls, such as Self-Refine and Multiagent-Debate, are increasingly critical to AI advancements. Perhaps surprisingly, we find empirically that choosing different models for different modules has a substantial effect on these systems' performance. Thus, we ask a core question in compound AI systems: for each LLM call or module in the system, how should one decide which LLM to use? As a first step, we formally show that the model selection problem (MSP) is computationally intractable. Next, we propose LLMSELECTOR, a principled framework that learns LLMs' strengths and weaknesses across different modules through an LLM evaluator and then performs an efficient optimization to select which models to use. Our theoretical analysis gives mathematical conditions under which LLMSELECTOR only requires LLM calls scaling linearly with the number of modules and the number of LLMs to identify the optimal model selection. Extensive experiments across diverse tasks, including question answering, constrained text generation, and code execution, demonstrate that LLMSELECTOR confers 4%-73% accuracy gains for popular compound AI systems with general-purpose models (e.g., GPT-4o and Claude 3.5 Sonnet), and 3%-21% gains with frontier reasoning models (e.g., o3-mini and Gemini 2.0 Flash).

1. Introduction

Researchers and developers are increasingly leveraging large language models (LLMs) by composing multiple LLM calls in a compound AI system to tackle complex tasks (Du et al., 2024; Zhang et al., 2024b; Madaan et al., 2023; DeepMind, 2023; Shinn et al., 2023; Renze & Guven, 2024; Zaharia et al., 2024). For example, a common practice is to use one LLM call to generate one initial answer, one LLM call to give feedback, and one more call to refine the answer based on the feedback, known as Self-Refine (Renze & Guven, 2024; Madaan et al., 2023; Ji et al., 2023). Another example is Multiagent-Debate (Du et al., 2024; Liang et al., 2024; Khan et al., 2024), where multiple LLM calls are made to propose initial answers and then debate which ones are correct. Compared to making a single model call, significant improvements are possible because the compound systems decompose challenging tasks into simpler sub-tasks, and perform one LLM call for each sub-task.

Most existing work on compound systems focuses on optimizing prompts used in individual modules and/or module interactions, while using the same LLM for all modules (Khattab et al., 2024; Yuksekgonul et al., 2024; Wu et al., 2023; Chase et al., 2022). While this simplifies compound system design, it also leaves important questions unaddressed. In particular, does using different models across modules improve a compound system's performance? Perhaps surprisingly, we find empirically that these choices have a substantial effect on quality: different models are better at different modules. Then how should one select which model to use for each module? With the growing number of LLM calls in compound systems and available LLMs, automated model selection is critical to enhance generation quality, simplify decision-making for users, and improve accessibility for non-experts.

We take a first step by systematically studying model selection in static compound AI systems, i.e., those where the number of modules, the sequencing of module calls, and the mapping between modules and models are fixed. In this context, we find that allocating different LLMs to different modules leads to up to 100% higher performance than allocating the same LLM to all modules (Figure 1). As an example, consider again the Self-Refine system (Madaan et al., 2023) consisting of three modules: a generator, a critic, and a refiner. LLM A may be better at providing feedback but worse at generating and refining answers than LLM B. In this case, allocating LLM A for the critic and LLM B for the generator and refiner is better than allocating either one to all modules.

[1]Microsoft Research [2]Stanford University [3]Princeton University [4]University of California, Berkeley. Correspondence to: Lingjiao Chen <lingjiaochen@microsoft.com>.

Accepted at the ICML 2025 Workshop on Collaborative and Federated Agentic Workflows (CFAgentic@ICML'25), Vancouver, Canada. July 19, 2025.

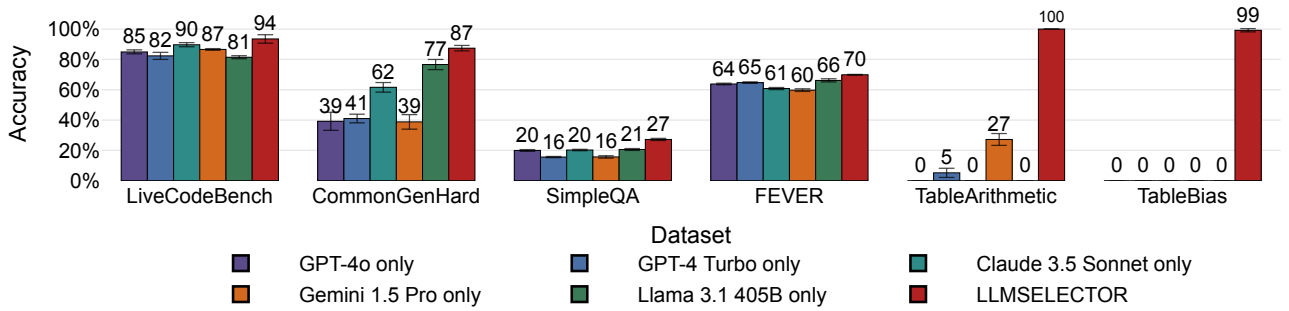

Figure 1: LLMSELECTOR outperforms compound AI systems that always call the same LLM. Here we study three compound systems, namely, Self-Refine (on LiveCodeBench and CommonGenHard), Multiagent-Debate (on SimpleQA and FEVER), and Locate-Solve (on TableArithmetic and TableBias). LLMSELECTOR achieves 4%-73% accuracy gains over allocating any model alone by allocating different models to different modules in these compound systems. The error bars show the standard deviations across 5 independent runs.

Next, we formulate the model selection problem (MSP), i.e., identifying the best model each module should use to maximize the overall performance. MSP is challenging in principle, as it is infeasible to exhaustively search the exponentially large space of all model choices. More precisely, there are $|M|^{|V|}$ choices, where $|V|$ is the number of components, and $|M|$ is the number of models. We show that choosing the models optimally involves solving a problem that is NP-Hard.

However, in this paper we show that solving MSP is possible with much lower complexity, specifically, $O(|M| \cdot |V|)$. This leverages two key insights we make that apply to many cases: (i) the end-to-end performance can be monotonic in per-module performance, i.e., if you replace the model of a component with a better model, the end-to-end system's performance will improve, and (ii) per-module performance can be estimated accurately by an LLM evaluator. This motivates us to design LLMSELECTOR, a framework that tackles MSP efficiently for any static system with provable guarantees on performance optimality and linear computation complexity under mild assumptions. LLMSELECTOR first learns the strengths and weaknesses of each model on different modules via an LLM evaluator. Then it initializes each module with the learned best model and iteratively updates each module. This is applicable to any compound system whose number of modules is fixed. Furthermore, LLMSELECTOR incurs only limited overhead. We provide the mathematical conditions under which LLMSELECTOR finds the optimal solution for MSP with linear complexity, i.e., uses a number of LLM calls that is linear in the number of modules and models (Section 4).

We conduct systematic experiments on a diverse set of compound AI systems using general-purpose LLMs (including GPT-4o, Claude 3.5 Sonnet, and Gemini 1.5 Pro) and reasoning models (such as o3-mini, Claude 3.7 Sonnet, and Gemini 2.0 Flash), for a range of tasks, such as question answering, constrained text generation, and code execution. The performance gap among using different models is as high as 100%. LLMSELECTOR achieves 4%-73% performance gains compared to allocating the same LLM to all modules using general-purpose models (Figure 1) and 3%-21% using reasoning models (Figure 3 in Section 5). LLMSELECTOR also outperforms advanced techniques specializing in prompt optimization (Table 1 in Section 5). This further highlights the importance of model selection for compound AI systems. In short, our main contributions are:

- **Model selection problem.** We formulate the model selection problem (MSP), an increasingly important but under-explored problem. We have found empirically that allocating different models to different modules has large performance effects (up to 100%), and show formally that optimizing MSP is NP-Hard.

- **The LLMSELECTOR framework.** To optimize MSP, we propose LLMSELECTOR, a principled framework that learns the strengths and weaknesses of each model across different each modules via an LLM evaluator, and then performs an efficient optimization to select which modules to use. We give mathematical conditions under which LLMSELECTOR finds the optimal solution for MSP with linear complexity.

- **LLMSELECTOR's practical effectiveness.** Through extensive experiments on practical systems using general-purpose LLMs (including GPT-4o and Claude 3.5 Sonnet) and reasoning models (such as o3-mini and Claude 3.7 Sonnet), we have found that LLMSELECTOR offers substantial performance gains (4%-73%) over a range tasks including question answering, con-

strained text generation, and code execution.

2. Related Work

Compound AI system optimization. Prompt engineering and module interaction design is a central topic of compound AI system optimization. While existing work often relies on manually tuning them (DeepMind, 2023; Shinn et al., 2023; Zhou et al., 2024b; Pryzant et al., 2023; Fourney et al., 2024; Zhao et al., 2024; Lu et al., 2023; Zhao et al., 2024), recent work studies how to automate this process, such as DSPy (Khattab et al., 2024), Textgrad (Yuksekgonul et al., 2024), and Autogen (Wu et al., 2023; Zhang et al., 2024a). On the other hand, our work focuses on model selection, a third axis for compound system optimization, complementary to prompt optimization and module interaction design.

Model market utilization. Model market utilization studies how to use all available models for downstream tasks (Lu et al., 2024; Ramírez et al., 2024; Miao et al., 2023). While they mainly focus on *single-stage* tasks such as classification (Chen et al., 2020; Huang et al., 2025) and question answering (Chen et al., 2024b; Shekhar et al., 2024), we study model utilization for compound AI systems requiring *multiple stages*. This is a much more challenging problem as the search space is much larger.

LLM-as-a-judge. LLMs are widely used for judging complex generations, termed LLM-as-a-judge. Researchers have extensively studied how LLM judges align with human preference (Zheng et al., 2023; Shankar et al., 2024), how to improve its quality (Kim et al., 2023), how to evaluate it (Chiang et al., 2024; Chen et al., 2024a; Zeng et al., 2023), as well as many other applications (Johri et al., 2025; Gu et al., 2024; Zhou et al., 2024a). In this paper, we find a novel use case of LLM-as-a-judge: evaluating module-wise performance to accelerate model selection optimization.

3. Compound AI Systems

Static Compound AI systems. As defined by (Zaharia et al., 2024), compound AI systems address AI tasks by synthesizing multiple components that interact with each other. Here, we denote a static compound AI system by a directed acyclic graph $G \triangleq (V, E)$, where each node $v \in V$ denotes one module, and each directed edge $e \triangleq (u, v) \in E$ indicates that the output from module u is sent to module v as input. We also assume a final output module that generates the final output with no output edge, and an input module representing the input query with no input edge.

LLM modules. An LLM module is a module that utilizes an LLM to process the inputs. It typically concatenates all inputs as a text snippet (via some prompt template), obtains an LLM's response to this snippet, and sends the response as output. Throughout this paper, all modules are LLM modules to simplify notations.

Notations. Table 2 in Appendix A lists our notations. We also use $f_{i\to k}$ to indicate a function that is the same as function f except that the value i is mapped to the value k.

4. The Model Selection Problem

This section presents how to model and optimize model selection for static compound AI systems.

4.1. Problem Statement

Consider a static compound AI system $G = (V, E)$ and a set of LLMs $M \triangleq \{1, 2, \cdots, |M|\}$ to use. Let $\mathcal{F} : V \mapsto M$ denote all possible model allocations, each of which allocates an LLM $k \in M$ to a module $v \in V$. Given a task distribution $\mathcal{D}$, the performance of the compound AI system using the model allocation $f \in \mathcal{F}$ is $P(f) \triangleq \mathbb{E}_{z\in\mathcal{D}}[p(f, z)]$. Here, z denotes a task sampled from the data distribution, and $p(f, z)$ is the performance of the compound AI system on the given task z using the allocation f. The model selection problem is modeled as

$$\max_{f\in\mathcal{F}} P(f) \tag{1}$$

4.2. The assumptions

Problem 1 is challenging without any assumptions. As the search space grows exponentially in the number of modules $|V|$, we can actually show that Problem 1 is NP-Hard.

Lemma 4.1. *Problem 1 is NP-Hard in $|V|$.*

The proof is left to Appendix B. In the following, we list our assumptions to enable tractable analysis.

Binary performance. For simplicity, we only consider binary performance, i.e., $p(f, z) \in \{0, 1\}$.

Decomposition to per-module performance. In classic computing systems such as a hardware stack, optimizing individual components (such as CPU, GPU, and memory) often leads to better overall performance. Similarly, improving individual modules' quality should also lead to better overall quality of a compound AI system. Here we assume that a compound system's performance is a monotone function of individual modules' performance. Formally, let $p_i(f, z)$ denote module v_i's performance on the task z using allocation f. Then the end-to-end performance can be decomposed as $p(f, z) = h(p_1(f, z), p_2(f, z), \cdots, p_L(f, z))$, where $h(\cdot)$ is monotonically increasing.

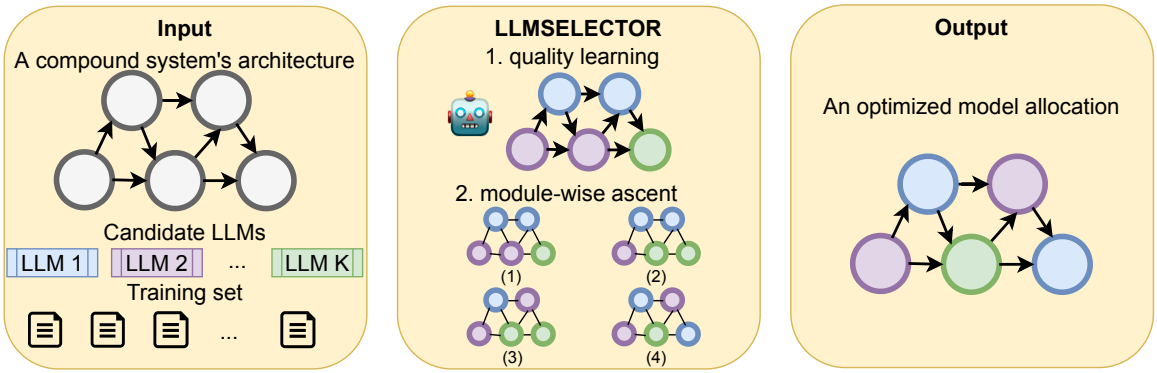

Figure 2: LLMSELECTOR Workflow. LLMSELECTOR takes as input a compound AI system, a pool of candidate LLMs, a training dataset consisting of question-answer pairs, and a training budget. It uses an LLM evaluator to learn which models perform best for each module, and then iteratively optimizes one module's allocation at a time until the budget is reached or no performance gain is possible. Finally, LLMSELECTOR returns an optimized model allocation.

Monotone module-wise performance. The module-wise performance needs to satisfy certain properties to enable us to analyze the interplay between individual modules and the compound systems. In this paper, we focus on module-wise performance p_i with the following two conditions.

- p_i is *intra-monotone*:$p_i(f_{i\to k}, z) \geq p_i(f_{i\to k'}, z) \implies p_i(f'_{i\to k}, z) \geq p_i(f'_{i\to k'}, z)$. In simple terms, p_i induces a "ranking" for each module.

- p_i is *inter-monotone*: $p_i(f_{i\to k}, z) > p_i(f_{i\to k'}, z) \implies \forall j, p_j(f'_{i\to k}, z) \geq p_j(f'_{i\to k'}, z)$. In other words, if module ith performance is higher by replacing its allocated model from A to B, then this should not hurt other modules' performance.

4.3. The LLMSELECTOR framework

The above analysis motivates our design of LLMSELECTOR, a principled framework for efficiently optimizing model allocation in compound AI systems.

Figure 2 gives an overview of how LLMSELECTOR works. It takes the compound AI system architecture G, the set of LLM M, a training dataset $\mathcal{D}_{\text{Tr}}$, and a training budget B as input, and returns an optimized model allocation $\hat{f}$ as the output. LLMSELECTOR consists of two stages, namely, quality learning and module-wise descent.

Quality learning. In the first stage, an allocation f^a is learned via an LLM evaluator, which estimates the ith module performance for any given module i, task z and allocation f, denoted by $\hat{p}_i(f, z)$. Specifically, for a given z, we start with some random allocation $f^{z,0}$, and iteratively update each module with the best module-wise performance estimated by the LLM evaluator:

$$f^{z,i} \leftarrow \max_{f:\exists k, f=f^{z,i-1}_{i\to k}} \hat{p}_i(f, z), \textit{where } i = 1, 2, \cdots, |V|. \tag{2}$$

We take the majority vote as the learned allocation, i.e., $f^a \leftarrow \textit{mode}(\{f^{z,|V|}\}_{z\in\mathcal{D}_{\text{Tr}}})$.

Module-wise ascent. The learned allocation is not necessarily optimal because the LLM evaluator can be noisy, and thus we perform additional search based on the ground-truth overall performance. Starting with the learned allocation f^a, we iteratively update each module by the model with the best overall performance until budget is reached or no more improvement is possible:

$$f^i \leftarrow \max_{f:\exists k, f=f^{i-1}_{i'\to k}} \sum_{z\in\mathcal{D}_{Tr}} p(f, z), \tag{3}$$

where $f^0 = f^a, i' = i \pmod{|V|}$. The details can be found in Algorithm 1. The following result shows when LLMSELECTOR can identify the optimal allocation, and we leave the proof to Appendix B.

Theorem 4.2. *Algorithm 1 always terminates. Suppose Problem 1 has a unique solution, for each task z in $\mathcal{D}_{Tr}$, the optimal allocation is unique, and the LLM evaluator $\hat{p}_i = p_i$. Then for some constant $c > 0$, with probability at least $1 - O(\exp(|V|\ln|M| - c|\mathcal{D}_{Tr}|))$, Algorithm 1 returns the optimal solution to Problem 1 for any $B \geq |M||V|$.*

Theorem 4.2 reveals several properties of LLMSELECTOR. First, LLMSELECTOR is guaranteed to converge. Second, assuming that the LLM evaluator is perfect, a small training set is sufficient to find the optimal model allocation. Indeed, the training data size only needs to grow linearly with the number of modules and log-linearly with the number of models with high probability.

Table 1: Performance of LLMSELECTOR and other approaches for optimizing compound AI systems. We focus on three common systems (Self-Refine, Multiagent-Debate, and Locate-Solve) each of which is evaluated on two tasks. The performance gain is the absolute improvement by LLMSELECTOR against the best of allocating any fixed (same) model to all modules (with underlines). We also compare LLMSELECTOR with the MIPROv2 optimizer implemented in DSPy (using GPT-4o as the LLM) with the default parameters. We also box the second-best result for each dataset. Overall, LLMSELECTOR achieves 4%-73% accuracy gains over allocating any fixed model to all modules. Interestingly, LLMSELECTOR also outperforms MIPROv2, which specializes in prompt optimization.

Method	Compound AI System					
	Self-Refine		Multiagent-Debate		Locate-Solve	
	LiveCodeBench	CommonGenHard	SimpleQA	FEVER	TableArith	TableBias
GPT-4o	85%	39%	20%	64%	0%	0%
GPT-4 Turbo	82%	41%	16%	65%	5%	0%
GPT-4o mini	71%	9%	5%	62%	1%	0%
Claude 3.5 Sonnet	*90%*	62%	20%	61%	0%	0%
Claude 3.5 Haiku	46%	17%	8%	58%	1%	43%
Gemini 1.5 Pro	87%	39%	16%	60%	*27%*	0%
Gemini 1.5 Flash	80%	13%	5%	38%	8%	2%
Llama 3.1 405B	81%	*77%*	*21%*	*66%*	0%	0%
Llama 3.1 70B	63%	69%	12%	7%	0%	*50%*
Qwen 2.5 72B	80%	26%	5%	48%	1%	0%
DSPy MIPROv2	87%	71%	22%	68%	0%	0%
LLMSELECTOR	**94%**	**87%**	**27%**	**70%**	**100%**	**100%**
Gains	4%	10%	6%	4%	73%	56%

5. Preliminary Experiments

We compare the performance of LLMSELECTOR with vanilla compound AI systems using real-world LLM models in this section. Our goal is three-fold: (i) validating that allocating different models to different modules can substantially improve compound AI systems' performance, (ii) measuring the performance gains enabled by LLMSELECTOR qualitatively, and (iii) understanding when and why compound systems optimized by LLMSELECTOR outperforms vanilla systems quantitatively.

Experiment setups. The main experiments are conducted with $|M| = 10$ general-purpose LLMs, including GPT-4o, GPT-4o mini, GPT-4-Turbo, Claude 3.5 Sonnet, Claude 3.5 Haiku, Gemini 1.5 Pro, Gemini 1.5 Flash, Llama 3.1 405B, Llama 3.1 70B, and Qwen 2.5 72B, with temperature = 0.1 and maximum number of tokens = 1000 unless specified. We also conduct experiments with $|M| = 3$ reasoning models, o3-mini, Claude 3.7 Sonnet, and Gemini 2.0 Flash. We study three compound AI systems, namely, Self-Refine, Multiagent-Debate, and Locate-Solve, on six diverse tasks, including LiveCodeBench, CommonGenHard, SimpleQA, FEVER, TableArithmetic, and TableBias. We use 50% of each dataset for training and the other 50% for evaluation. All experiments were run on a machine with 10 Intel 2.2 GHz cores, 192 GB RAM, and 3 TB disk with Ubuntu 20.04 LTS as the OS. More details can be found in Appendix C.1, Appendix C.2, and Appendix C.3.

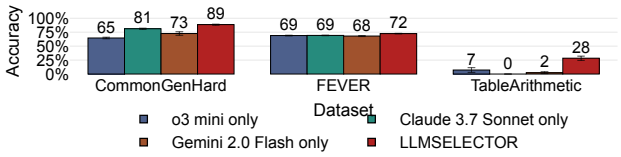

Figure 3: LLMSELECTOR's performance using frontier reasoning models including o3-mini, Gemini 2.0 flash, and Claude 3.7 Sonnet. The error bar is the standard deviation over 5 runs. Overall, we have observed that LLMSELECTOR consistently offers substantial (3% to 21%) performance improvements compared to using any fixed reasoning models. This further highlights the importance of model selection in the era of reasoning models.

5.1. Quantitative Performance Improvements

We start by studying the performance of LLMSELECTOR on practical compound AI systems. We compare LLMSELECTOR with using any fixed model for all modules and DSPy optimizer MIPROv2 (Khattab et al., 2024),

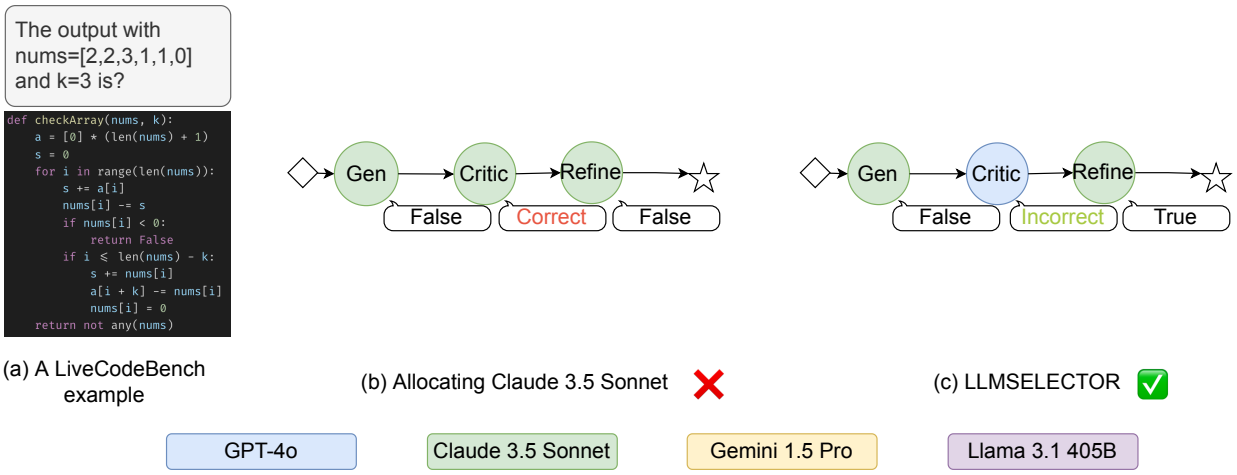

Figure 4: An illustrative example of applying LLMSELECTOR on Self-Refine on LiveCodeBench. (a) the task. (b) Using Claude 3.5 Sonnet for all modules leads to the wrong answer. (c) LLMSELECTOR learns to use GPT-4o as the feedback provider, leading to the correct answer.

an open-source library specialized for prompt optimization in compound systems. MIPROv2 searches for best prompts using Bayesian optimization. We use GPT-4o as the backbone LLM, and set max_bootstrapped_demos=2, max_labeled_demos=2, and all other parameters as default.

General-purpose models. Table 1 summarizes the quantitative results using $|M| = 10$ general-purpose models. First, we observe that no LLM is universally better than all other LLMs for all tasks. For example, Gemini-1.5 Pro performs the best on TableArthmetic, but GPT-4o is the best for FEVER. Second, LLMSELECTOR offers 4%-73% performance gains compared to the best baselines. Interestingly, LLMSELECTOR also outperforms the DSPy MIPROv2, which optimizes the prompt. This is because different models have their own strengths and weaknesses, and prompting alone is insufficient to overcome an LLM's weaknesses.

Reasoning models. As shown in Figure 3, LLMSELECTOR offers 3%-21% performance gains compared to using frontier reasoning models such as o3-mini and Claude 3.7 Sonnet. This suggests that LLMSELECTOR is effective across different types of LLMs.

5.2. Qualitative understanding of LLMSELECTOR

To further understand when and why LLMSELECTOR outperforms allocating the same model to all modules, we dive into an example from LiveCodeBench using Self-Refine, as shown in Figure 4. Here, the task is to decide the output of a python program for a particular input. In this case, using Claude 3.5 Sonnet leads to an incorrect answer. As shown in Figure 4(b), this is because Claude 3.5 Sonnet is not able to recognize its own mistake as a critic. And thus, the error propagates from the initial answer generator to the final output. On the other hand, LLMSELECTOR learns to use GPT-4o as the critic. As shown in Figure 4(c), GPT-4o, as the critic, correctly finds the mistake made by the initial answer generator. Thus, LLMSELECTOR leads to the correct answer. This justifies that LLMSELECTOR is effective because it identifies which model is most suitable for which role in the given compound AI system.

6. Conclusion

The complexity of orchestrating multiple LLM calls in compound AI systems underscores the critical need for strategic model selection to optimize these systems' performance across diverse tasks. In this paper, we propose LLMSELECTOR, a principled framework that identifies optimal model selection with provable performance and sample complexity guarantees, whose effectiveness has also been justified via extensive experiments on question answering, constrained text generation, code execution, and many other tasks.

Compound AI systems that make multiple LLM calls are a rapidly growing industry with broad economic and societal impact. Despite relieves users from tedious and challenging system configuration overhead, LLMSELECTOR opens the door for many exciting future directions. LLMSELECTOR focuses on optimizing compound AI systems with a bounded number of LLM calls, and it remains open how to select models for compound AI systems with a dynamic or unlimited number of LLM calls. Based on discussions with practitioners, it is also an interesting question to jointly optimize model selection and prompting methods. We will release the code and data to stimulate more research and positive societal impacts.

References

Chase, H. et al. Langchain. https://github.com/langchain-ai/langchain, 2022. Accessed: 2025-01-04.

Chen, D., Chen, R., Zhang, S., Liu, Y., Wang, Y., Zhou, H., Zhang, Q., Wan, Y., Zhou, P., and Sun, L. Mllm-as-a-judge: Assessing multimodal llm-as-a-judge with vision-language benchmark. *arXiv preprint arXiv:2402.04788*, 2024a.

Chen, L., Zaharia, M., and Zou, J. Y. Frugalml: How to use ml prediction apis more accurately and cheaply. *Advances in neural information processing systems*, 33: 10685–10696, 2020.

Chen, L., Zaharia, M., and Zou, J. Frugalgpt: How to use large language models while reducing cost and improving performance. *TMLR*, 2024b.

Chiang, W.-L., Zheng, L., Sheng, Y., Angelopoulos, A. N., Li, T., Li, D., Zhang, H., Zhu, B., Jordan, M., Gonzalez, J. E., et al. Chatbot arena: An open platform for evaluating llms by human preference. *arXiv preprint arXiv:2403.04132*, 2024.

DeepMind. Alphacode 2 technical report. 2023. URL https://storage.googleapis.com/deepmind-media/AlphaCode2/AlphaCode2_Tech_Report.pdf.

Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., and Mordatch, I. Improving factuality and reasoning in language models through multiagent debate. In *ICML*, 2024.

Fourney, A., Bansal, G., Mozannar, H., Tan, C., Salinas, E., Niedtner, F., Proebsting, G., Bassman, G., Gerrits, J., Alber, J., et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.

Gu, J., Jiang, X., Shi, Z., Tan, H., Zhai, X., Xu, C., Li, W., Shen, Y., Ma, S., Liu, H., et al. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*, 2024.

Huang, K., Shi, Y., Ding, D., Li, Y., Fei, Y., Lakshmanan, L., and Xiao, X. Thriftllm: On cost-effective selection of large language models for classification queries. *arXiv preprint arXiv:2501.04901*, 2025.

Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Ji, Z., Yu, T., Xu, Y., Lee, N., Ishii, E., and Fung, P. Towards mitigating llm hallucination via self reflection. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 1827–1843, 2023.

Johri, S., Jeong, J., Tran, B. A., Schlessinger, D. I., Wongvibulsin, S., Barnes, L. A., Zhou, H.-Y., Cai, Z. R., Van Allen, E. M., Kim, D., et al. An evaluation framework for clinical use of large language models in patient interaction tasks. *Nature Medicine*, pp. 1–10, 2025.

Khan, A., Hughes, J., Valentine, D., Ruis, L., Sachan, K., Radhakrishnan, A., Grefenstette, E., Bowman, S. R., Rocktäschel, T., and Perez, E. Debating with more persuasive llms leads to more truthful answers. *arXiv preprint arXiv:2402.06782*, 2024.

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., et al. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*, 2024.

Kim, S., Shin, J., Cho, Y., Jang, J., Longpre, S., Lee, H., Yun, S., Shin, S., Kim, S., Thorne, J., et al. Prometheus: Inducing fine-grained evaluation capability in language models. In *The Twelfth International Conference on Learning Representations*, 2023.

Liang, T., He, Z., Jiao, W., Wang, X., Wang, Y., Wang, R., Yang, Y., Shi, S., and Tu, Z. Encouraging divergent thinking in large language models through multi-agent debate. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *EMNLP*, November 2024.

Lu, J., Pang, Z., Xiao, M., Zhu, Y., Xia, R., and Zhang, J. Merge, ensemble, and cooperate! a survey on collaborative strategies in the era of large language models. *arXiv preprint arXiv:2407.06089*, 2024.

Lu, P., Peng, B., Cheng, H., Galley, M., Chang, K.-W., Wu, Y. N., Zhu, S.-C., and Gao, J. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems*, 36, 2023.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2023.

Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Jin, H., Chen, T., and Jia, Z. Towards efficient generative large language model serving: A survey from algorithms to systems. *arXiv preprint arXiv:2312.15234*, 2023.

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

Ramírez, G., Birch, A., and Titov, I. Optimising calls to large language models with uncertainty-based two-tier selection. *arXiv preprint arXiv:2405.02134*, 2024.

Renze, M. and Guven, E. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*, 2024.

Shankar, S., Zamfirescu-Pereira, J., Hartmann, B., Parameswaran, A., and Arawjo, I. Who validates the validators? aligning llm-assisted evaluation of llm outputs with human preferences. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–14, 2024.

Shekhar, S., Dubey, T., Mukherjee, K., Saxena, A., Tyagi, A., and Kotla, N. Towards optimizing the costs of llm usage. *arXiv preprint arXiv:2402.01742*, 2024.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2023.

Thorne, J., Vlachos, A., Cocarascu, O., Christodoulopoulos, C., and Mittal, A. The FEVER2.0 shared task. In *Proceedings of the Second Workshop on Fact Extraction and VERification (FEVER)*, 2018.

Wei, J., Karina, N., Chung, H. W., Jiao, Y. J., Papay, S., Glaese, A., Schulman, J., and Fedus, W. Measuring short-form factuality in large language models. *arXiv preprint arXiv:2411.04368*, 2024.

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

Yuksekgonul, M., Bianchi, F., Boen, J., Liu, S., Huang, Z., Guestrin, C., and Zou, J. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Zaharia, M., Khattab, O., Chen, L., Davis, J. Q., Miller, H., Potts, C., Zou, J., Carbin, M., Frankle, J., Rao, N., and Ghodsi, A. The shift from models to compound ai systems. https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/, 2024.

Zeng, Z., Yu, J., Gao, T., Meng, Y., Goyal, T., and Chen, D. Evaluating large language models at evaluating instruction following. *arXiv preprint arXiv:2310.07641*, 2023.

Zhang, S., Zhang, J., Liu, J., Song, L., Wang, C., Krishna, R., and Wu, Q. Offline training of language model agents with functions as learnable weights. In *ICML*, 2024a.

Zhang, Y., Sun, R., Chen, Y., Pfister, T., Zhang, R., and Arik, S. Ö. Chain of agents: Large language models collaborating on long-context tasks. *arXiv preprint arXiv:2406.02818*, 2024b.

Zhao, A., Huang, D., Xu, Q., Lin, M., Liu, Y.-J., and Huang, G. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36: 46595–46623, 2023.

Zhou, R., Chen, L., and Yu, K. Is llm a reliable reviewer? a comprehensive evaluation of llm on automatic paper reviewing tasks. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 9340–9351, 2024a.

Zhou, W., Ou, Y., Ding, S., Li, L., Wu, J., Wang, T., Chen, J., Wang, S., Xu, X., Zhang, N., Chen, H., and Jiang, Y. E. Symbolic learning enables self-evolving agents. 2024b. URL https://arxiv.org/abs/2406.18532.

A. Notations and Technical Details

We summarize the main notations in Table 2. The details of LLMSELECTOR is given in Algorithm 1.

Table 2: Notations.

Symbol	Description
$G = (V, E)$	A compound AI system
$\|V\|$	Number of LLM modules
M	The set of LLMs
$f : V \mapsto M$	A model allocation
z	One task
$P(f)$	End-to-end performance
$p(f, z)$	End-to-end performance on z
$p_i(f, z)$	ith module's performance on z
$\mathcal{D}$	The task distribution
$\mathcal{D}_{\text{Tr}}$	The training dataset

Algorithm 1: How LLMSELECTOR works.

Input: A compound AI system $G = (V, E)$, a pool of K candidate LLMs, a training dataset $\mathcal{D}_{\text{Tr}}$, and a training budget B

Output: An optimized model allocation $\hat{f}$

Choose a random $f^0 \in \mathcal{F}$ `// initialize`

Compute $f^{z,i}$ by equation (2) $\forall z \in \mathcal{D}_{\text{Tr}}, i = 1, \cdots, \min\{|V|, \lfloor \frac{B}{M} \rfloor\}$

$f^0 \leftarrow \textit{mode}(\{f^{z,\min\{|V|,\lfloor \frac{B}{M} \rfloor\}}\}_{z \in \mathcal{D}_{\text{Tr}}})$ `// quality learning`

Compute f^i by equation (3) $\forall i = 1, \cdots, \min\{\lfloor \frac{B}{|M|} \rfloor - |V|, 0\}\rfloor$ `// module-wise ascent`

return f^i `// optimized model choices`

B. Missing Proofs

B.1. Proof of Lemma 4.1

Proof. We prove that Problem (1) is NP-Hard via a polynomial-time reduction from the canonical NP-complete problem 3-SAT.

Construction. Consider a 3-SAT instance with a CNF formula over Boolean variables $z_1, \ldots, z_m$ with clauses $C_1, \ldots, C_n$, where each clause has exactly three literals.

Construct the following compound AI system instance:

- Let $|V| = m$, with one module $v_i \in V$ corresponding to each Boolean variable z_i.
- Let $M = \{0, 1\}$, where model 0 represents `False` and model 1 represents `True`.
- Each allocation $f \in \mathcal{F}$ corresponds to a truth assignment to all variables.
- For each clause C_j, define a task $z_j \in \mathcal{D}$ whose success depends on whether clause C_j is satisfied under allocation f.

Define:

$$p(f, z_j) \triangleq \begin{cases} 1, & \text{if } C_j \text{ is satisfied under assignment } f, \\ 0, & \text{otherwise.} \end{cases}$$

Let the data distribution $\mathcal{D} = \{z_1, \ldots, z_n\}$ be uniform over clauses. Then the expected performance becomes:

$$P(f) = \mathbb{E}_{z \in \mathcal{D}} \left[p(f, z) \right] = \frac{1}{n} \sum_{j=1}^{n} p(f, z_j),$$

which is simply the fraction of clauses satisfied by the assignment f.

Reduction from 3-SAT to Problem 1. The original 3-SAT formula is satisfiable if and only if there exists an allocation f such that $P(f) = 1$. Thus, any 3-SAT instance can be solved by solving Problem 1 for the above compound AI system instance, and returning satisfiable if and only if the optimal solution is 1.

Thus, we conclude that Problem (1) is NP-Hard in $|V|$ (number of modules). □

B.2. Proof of Theorem 4.2

Proof. The first half (termination) is straightforward: Line 2 takes $\min\{|V|, \lfloor \frac{B}{M} \rfloor\}$ iterations, and line 4 takes $\min\{\lfloor \frac{B}{|M|} \rfloor - |V|, 0\}\rfloor$ iterations. Thus, Algorithm 1 terminates after $\lfloor \frac{B}{|M|} \rfloor$ iterations.

Now we turn to the second half. This involves two parts. First, we show that the majority vote of all individual $f^{z,|V|}$ leads to the optimal solution to model selection on the training dataset. Next, we show that the optimal solution on the training dataset is the same as that on the data distribution with high probability. Both of them would need the following lemma.

Lemma B.1. *Assume $\hat{p}_i = p_i$. Then $f^{z,|V|}$ is the unique optimal allocation for the task z.*

Proof. We first note that the uniqueness of a task's optimal model allocation implies that for each module only one unique model maximizes the per-module quality. That is, for each i, there exists some k, such that for any $k' \neq k$, we have $p_i(f_{i \to k} > p_i(f_{i \to k'})$. Suppose not. Let k^* be the model allocated to module i by the optimal allocation. Due to the monotone assumption, k^* should also maximize module i's performance. Let k' be another model that maximizes module i's performance. By the inter-monotone assumption, switching from k^* to k' does not hurt any other module's performance. By the monotone assumption, k' also maximizes the overall performance. A contradiction. Therefore, for each module, there is only one unique model that maximizes its performance, regardless of how other modules are allocated.

Now we can show that at the iteration i, allocation $f^{z,i}$ allocates the same models to the first i modules as the optimal allocation. To see this, one can simply notice that the unique "best" model for each module must also be the optimal model for the end-to-end system. This is again because of the monotone assumption: otherwise, one can change the model in the optimal allocation to have better performance of one module and thus the overall system. Therefore, allocating the per-module optimal model is the same as allocating the optimal model for the entire system. Thus, at iteration i, allocation $f^{i,z}$ allocates the same models to the first i modules as the optimal allocation. Therefore, after $|V|$ iterations, the allocation must be the unique optimal allocation for query z.

□

Now let us start with the first part. We first argue that the allocation learned at line 3, i.e., the majority vote of $f^{z,|V|}$ (since $B \geq |V| \cdot |M|$) over all z, is the optimal solution to

$$\max_f \frac{1}{|\mathcal{D}_{\text{Tr}}|} \sum_{z' \in \mathcal{D}_{\text{Tr}}} p(f, z').$$

By Lemma B.1, the optimal allocation for each query is unique. That is, $p(f, z')$ is 1 if $f = f^{z',|V|}$, and 0 otherwise. Hence, the training performance of any fixed f is proportional to

$$\sum_{z' \in \mathcal{D}_{\text{Tr}}} \mathbf{1}_{f = f^{z',|V|}}$$

That is, the performance of allocation f is proportional to the number of training data points whose optimal allocation is the same as f. Therefore, taking the majority vote of all optimal allocations is sufficient to obtain the best allocation for the training dataset.

Now we turn to the second part. By definition, $P(f) = \mathbb{E}(p(f,z)) = \mathbb{E}(\mathbf{1}_{f=f^{z,|V|}}) = \Pr[f = f^{z,|V|}]$. In words, the performance of allocation f is the probability that it is the same as the optimal allocation of a query sampled from the distribution. The optimal allocation is thus $f^* = \max_{f\in\mathcal{F}} \Pr[f = f^{z,|V|}]$, where $\mathcal{F}$ is all possible allocations. The solution we obtain at line 3, f^a, as shown in the first part, is the optimal solution on the training dataset, i.e., $f^a = \max_{f\in\mathcal{F}} \sum_{z\in\mathcal{D}_{\text{Tr}}} \frac{1}{|\mathcal{D}_{\text{Tr}}|}\mathbf{1}_{f=f^{z,|V|}}$. Now we can show that these two allocations are the same with high probability, by showing that for each allocation f the two objectives are close to each other with high probability, and then applying the union bound.

Specifically, let $\Delta \triangleq P(f^*) = \max_{f\in F-\{f^*\}} P(f)$, i.e., Δ is the gap between the optimal allocation's performance and the second best allocation's performance. By the assumption that the optimal solution to Problem 1 is unique, we must have $\Delta > 0$. For ease of notation, let $n \triangleq |\mathcal{D}_{\text{Tr}}|$ denote the size of the training dataset, and $\hat{P}(f) \triangleq \sum_{z\in\mathcal{D}_{\text{Tr}}} \frac{1}{|\mathcal{D}_{\text{Tr}}|}\mathbf{1}_{f=f^{z,|V|}}$.

For any given allocation f, by Hoeffding bound,

$$\mathbb{P}\left(\left|P(f) - \hat{P}(f)\right| \geq \epsilon\right) \leq 2\exp\left(-2n\epsilon^2\right)$$

Set $\epsilon = \Delta/2$. This implies that with probability at least $1 - 2\exp\left(-n\Delta^2/2\right)$, $\left|P(f) - \hat{P}(f)\right| < \Delta/2$. By union bound, for all allocation f, with probability at least $1 - 2|\mathcal{F}|\exp\left(-n\Delta^2/2\right)$, $\left|P(f) - \hat{P}(f)\right| < \Delta/2$ holds for all f. Now, this suggests that for any $f \neq f^*$, we have $\hat{P}(f^*) - P(f^*) > -\Delta/2$, and $\hat{P}(f) - P(f) < \Delta/2$. Therefore, we can have

$$\begin{aligned}
\hat{P}(f^*) - \hat{P}(f) =& \hat{P}(f^*) - P(f^*) - (\hat{P}(f) - P(f)) + (P(f^*) - P(f)) \\
>& -\Delta/2 - \Delta/2 + (P(f^*) - P(f)) \\
=& P(f^*) - P(f) - \Delta \geq 0
\end{aligned}$$

where the last $\geq$ is be definition of Δ. That is to say, the performance of f^* on the training dataset is higher than that of any other allocation with high probability. Hence, the allocation that maximizes the performance on the training dataset must be the same allocation that maximizes the performance on the data distribution, with probability at least $1 - 2|F|\exp\left(-n\Delta^2/2\right)$. Recall that there are $|M|^{|V|}$ many possible allocations and thus $|\mathcal{F}| = |M|^{|V|}$ and also that $n = \mathcal{D}_{\text{Tr}}$ by definition. Thus, with probability at least $1 - 2\exp\left(|V|\ln|M| - |\mathcal{D}_{\text{Tr}}|\Delta^2/2\right)$, the obtained allocation by Algorithm 1 is the optimal allocation, which finishes the proof.

□

C. Experiment Details

C.1. Compound AI Systems

In this paper, we focus on three compound AI systems, Locate-Solve, Self-Refine, and Multiagent-Debate. Their architectures are shown in Figure 5. Locate-Solve designed for TableArithmetic and TableBias consists of two modules: the first module extracts the task associated with an ID from an input table, and the second module returns the answer to the extracted task. Self-Refine (Madaan et al., 2023) has a generator, a critic, and a refiner. The generator gives an initial answer to a question, the critic gives feedback to this answer, and the refiner uses the feedback to refine the original answer. Multiagent-Debate (Du et al., 2024) involves two types of modules: answer generators and debaters. The answer generators offer initial answers to a question. The debaters take the initial answers and then debate which one is correct. In this paper, we focus on a six-module Multiagent-Debate: three modules are answer generators, and the other three are the debaters.

C.2. Datasets and Evaluation Metrics

Now we provide details of all datasets used in this paper.

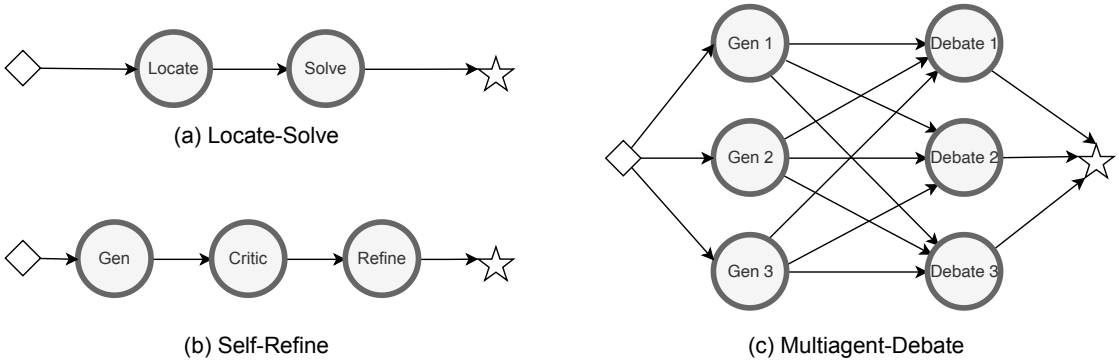

Figure 5: The architectures of the compound AI systems studied in the experiments. (a) Locate-Solve using two modules. (b) Self-Refine using three modules. (c) Multiagent-Debate that involves six modules in total.

LiveCodeBench. LiveCodeBench (Jain et al., 2024) is a benchmark for code understanding. We use the code execution task in LiveCodeBench [1]. It contains 479 questions in total. Each question contains a program and an input. The goal is to predict the output of the program. Note that this is a generative task, as the output space of a given program is unbounded. We use the exact match to measure the performance of a compound system's generation. This dataset is under the MIT License.

CommonGenHard. CommonGenHard (Madaan et al., 2023) is a constrained generation dataset consisting of 200 questions. Each question gives 20-30 concepts, and the goal is to generate a coherent paragraph that uses all the provided concepts. Since all LLMs used in our evaluation generate coherent texts, we focus on evaluating the quality of whether all concepts are included. That is, the quality is 1 if all concepts are contained in the generated paragraph, and 0 if any concept is missing. This dataset is under the Apache-2.0 License.

SimpleQA. SimpleQA (Wei et al., 2024) contains 4326 short, fact-seeking questions. Example questions include "Who received the IEEE Frank Rosenblatt Award in 2010" and "What is the first and last name of the woman whom the British linguist Bernard Comrie married in 1985". While seemingly simple, LLMs actually struggle to answer them correctly. We use the exact match to measure the generation quality of a compound system. This dataset is under the Apache-2.0 License.

FEVER. FEVER (Thorne et al., 2018) is a fact-verification dataset. We use the v2.0 variant consisting of 2384 questions [2]. Each question contains a claim, and the task is to classify the claim as one of NOT ENOUGH INFO, SUPPORTS, and REFUTES. Again, we use exact match as the accuracy metric. This dataset is under the Creative Commons Attribution Share Alike 3.0 License.

TableArithmetic. TableArithmetic is a synthetic dataset used to understand the locate-solve system's performance. It contains 100 questions. Each question consists of a table of "ID" and "task" rows, and the goal is to solve the task associated with a specific ID. Each row contains 100 entries. Each question has the form of "What is X+(10.9¿10.11)?", where X is a randomly generated integer.

TableBias. TableArithmetic is another synthetic dataset. It contains 100 questions. Each question consists of a table of "ID" and "task" rows, and the goal is to solve the task associated with a specific ID. Here, each table contains 80 entries. Each question has the form of "The surgeon, who is the boy's father, says I cannot operate on this boy, he is my son. Who is the doctor to the boy? (Ax) Father (Bx) Mother", where again x is a randomly generated integer.

C.3. LLM Endpoints and Providers

We give the details of all models used in our experiments in Table 3, including their API endpoints and model providers for reproducibility purposes.

[1] https://huggingface.co/datasets/livecodebench/execution-v2
[2] https://huggingface.co/datasets/fever/fever

Table 3: Overview of all LLMs used in this papers. We use $|M| = 10$ models for experiments using general-purpose models, and $|M| = 3$ models for experiments with frontier reasoning models. The model endpoints and providers are detailed here for reproducibility.

Type	Model	API Endpoint	Provider
General-Purpose	GPT-4o	gpt-4o-2024-05-13	OpenAI
General-Purpose	GPT-4o Mini	gpt-4o-mini-2024-07-18	OpenAI
General-Purpose	GPT-4 Turbo	gpt-4-turbo-2024-04-09	OpenAI
General-Purpose	Claude 3.5 Sonnet	claude-3-5-sonnet-20240620	Anthropic
General-Purpose	Claude 3.5 Haiku	claude-3-haiku-20240307	Anthropic
General-Purpose	Gemini 1.5 Pro	gemini-1.5-pro	Google
General-Purpose	Gemini 1.5 Flash	gemini-1.5-flash	Google
General-Purpose	Llama 3.1 405B	meta-llama/Meta-Llama-3.1-405B-Instruct-Turbo	Together AI
General-Purpose	Llama 3.1 70B	meta-llama/Meta-Llama-3.1-70B-Instruct-Turbo	Together AI
General-Purpose	Qwen 2.5 72B	Qwen/Qwen2.5-72B-Instruct-Turbo	Together AI
Reasoning	o3-mini	o3-mini-2025-01-31	OpenAI
Reasoning	Claude 3.7 Sonnet	claude-3-7-sonnet-20250219	Anthropic
Reasoning	Gemini 2.0 Flash	gemini-2.0-flash	Google

C.4. LLM Evaluators: Prompts and Ablation Studies

Prompt for the LLM evaluator. We give the LLM evaluator prompt template in the following box. The LLM evaluator takes the module index i and the compound AI system's description (including the description of each module, and how modules connect to each other) as input, and follows this prompt to evaluate module i's performance. As we focus on binary performance, the performance is either high (1) or low (0).

LLM evaluator prompt

You are an error diagnosis expert for compound AI systems. Below is the description of a compound AI system consisting of multiple modules, a query, the generations from each module of the compound AI system, the final output, and the desired answer. Assume that the desired answer is 100% correct. If the final output matches the correct answer, generate 'error: 0'. Otherwise, analyze whether module i leads to the mistake. If so, generate 'error: 1'. Otherwise, generate 'error: 0'. Think step by step.
[Compound AI system]:
[query]:
[module 0 output]:
[module 1 output]:
...:
[module $|V|$ output]:
[final output]:
[desired answer]:
[your analysis]:

Effects of LLM evaluator. Here we study how different LLM evaluators affect LLMSELECTOR's performance. In particular, we use three different LLM evaluators, namely, Gemini 1.5 Pro, GPT-4o, and Claude 3.5 Sonnet, and measure their evaluation accuracy as well as end-to-end system performance, i.e., how the learned Locate-Solve system performs on the testing dataset. As shown in Table 4, we first observe that the evaluation accuracy does vary across different evaluators. Gemini 1.5 Pro's evaluation accuracy is the highest (85%), while GPT-4o's accuracy is only 68.4%. On the other hand, we observe that the end-to-end performance by using any of these LLM evaluators is impressive. This suggests that the LLM evaluators do not need to be perfect to obtain a high-quality model allocation. Finally, we note that the evaluator accuracy

has an impact on "convergence rate", i.e., the training budget required to reach the optimal model allocation. When Gemini 1.5 Pro is the evaluator, budget=2 (the number of modules) is sufficient. This is because the LLM evaluator is near-optimal and thus the allocation anchoring is sufficient to find the optimal allocation, matching our theoretical analysis as well. When the LLM evaluator is noisy (such as GPT-4o), additional budget is needed for the module-wise ascent.

Table 4: Effects of different LLM evaluators for the Locate-Solve system on TableArithmetic using all 10 models. All LLM evaluators lead to a high end-to-end performance. Gemini 1.5 Pro's evaluation accuracy is the highest and thus requires the smallest training budget to find the optimal model allocation. On the other hand, using GPT-4o as the evaluator leads to a lower evaluation accuracy and thus requires more training budget.

LLM Evaluator	Evaluator Accuracy (%)	Required Budget	End-to-End Accuracy (%)
GPT-4o	68.4	40	100
Claude 3.5 Sonnet	70.1	40	100
Gemini 1.5 Pro	85.0	20	100

The first example shown in Figure **??**(a) is a task from the SimpleQA dataset. Allocating GPT-4o to all modules leads to an incorrect answer as seen in Figure **??**(b). This is because the GPT-4o generators always return 8 as the initial answers, and the debaters fail to identify this mistake. On the other hand, LLMSELECTOR, as demonstrated in Figure **??**(c), learns to allocate GPT-4o, Llama 3.1 405B, and Gemini 1.5 Pro for the three answer generators separately, and use GPT-4o for the three debaters. In this case, the three generators give completely different answers: 8, 3, and -18. Interestingly, the GPT-4o debaters reaches the consensus of 3, which is indeed the correct answer.

Another example from the LiveCodeBench dataset is shown in Figure **??**(d). Here we focus on the Self-Refine system which contains three modules (a generator, a critic, and a refiner). Recall that allocating Claude 3.5 Sonnet to all modules is better than allocating any other fixed LLMs, as shown in Table 1. However, this leads to an incorrect answer for this example, as shown in Figure **??**(e). This is because Claude 3.5 Sonnet as the critic mistakenly tags its initial generation as correct. On the other hand, LLMSELECTOR learns to allocate Claude 3.5 Sonnet for the generator and the refiner, but GPT-4o for the critic. As shown in Figure **??**(f), this leads to a correct response to the task. This is because GPT-4o is better than Claude 3.5 Sonnet as a critic for LiveCodeBench tasks.

To sum up, LLMSELECTOR performs better than allocating any fixed models to all modules, because it identifies the strengths and weaknesses of different models across modules, and then allocate to each module the model that best fits it.

D. Limitations and Broader Impacts

LLMSELECTOR focuses on optimizing compound AI systems with a bounded number of LLM calls, and it remains open how to select models for compound AI systems with a dynamic or unlimited number of LLM calls. Based on discussions with practitioners, it is also an interesting question to jointly optimize model selection and prompting methods.

Compound AI systems that make multiple LLM calls are a rapidly growing industry with broad economic and societal impact. The large increase in available LLMs makes it inevitable to select which LLMs to use for these systems. LLMSELECTOR offers an off-the-shelf framework to automate model selection in compound AI systems. This substantially relieves users from tedious and challenging system configuration overhead. It also makes compound AI systems more accessible to more users, especially those without professional skills and knowledge in LLMs. LLMSELECTOR can optimize a compound system over any given set of LLMs, enhancing the robustness and availability of compound AI systems—even in the face of cloud outages or individual model failures—thereby supporting more reliable AI services in critical applications. We will release the code and data to stimulate more research and positive societal impacts.