FLORA: GNNs as Predictors of Agentic Workflow Performances

Yuanshuo Zhang
Shanghai Jiao Tong University
BIGAI
`yuanshuo.z@sjtu.edu.cn`

Yuchen Hou
Shanghai Jiao Tong University

Bohan Tang
University of Oxford

Shuo Chen
BIGAI

Muhan Zhang
Peking University

Xiaowen Dong
University of Oxford

Siheng Chen [*]
Shanghai Jiao Tong University
`sihengc@sjtu.edu.cn`

Abstract

Agentic workflows invoked by Large Language Models (LLMs) have achieved remarkable success in handling complex tasks. However, optimizing such workflows is costly and inefficient due to extensive invocations of LLMs. To fill this gap, this paper formulates agentic workflows as computational graphs and proposes **FLORA** (work**FLO**w g**RA**ph neural networks), which are variants of Graph Neural Networks (GNNs), as efficient predictors of agentic workflow performances, thus avoiding repeated LLM invocations for evaluation. To empirically ground the effectiveness and efficiency of FLORA, we construct **FLORA-Bench**, a unified platform for training and benchmarking predictors of agentic workflow performances. With extensive experiments, we arrive at the following conclusions: (1) FLORA is a simple yet effective method to accurately predict agentic workflow performances, and (2) it demonstrates significant practical benefits, achieving up to a 125× speedup with minimal performance loss. These conclusions support new applications of GNNs and a novel direction towards automating agentic workflow optimization. All codes and datasets are public at this url.

1 Introduction

The rise of Large Language Models (LLMs) and LLM-based agents have showcased their ability to execute complex tasks across various domains such as code generation [1, 12, 19], problem solving [38, 40], reasoning [27, 34, 36], and decision making [20, 28]. Building upon these capabilities, agentic workflows, which leverage task decomposition and communications within multi-agent systems, have unlocked the potential to tackle more challenging tasks [12, 15, 24, 44]. Notably, OpenAI's five-level framework of Artificial General Intelligence (AGI) and Artificial Superhuman Intelligence (ASI) positions such agentic workflows as vital path to transitioning from narrow AI (Level 1-2) to organizational-agentic superintelligence (Level 3-5), where systems autonomously manage large-scale agents. Despite the immense potential of agentic workflows, reliance on manual design limits their ability to achieve the self-evolving nature of AGI and ASI. Therefore, automating agentic workflow optimization is increasingly important as an emerging research topic [17].

To automate the optimization process, researchers have conceptualized agentic workflows as optimizable graphs, where nodes represent agents handling specific subtasks and edges signify the task

[*]Corresponding Author

dependencies and collaborative interactions as shown in fig. 1. While existing methods have made strides by optimizing these graphs to improve system performance [19, 27, 42–44, 47], they share a critical limitation: the optimization process heavily relies on extensive LLM invocations to execute workflows and evaluate their performance. This dependence introduces significant computational, temporal, and financial overhead, making the process costly and inefficient in real-world applications. Therefore, a necessary advancement lies in developing predictors capable of evaluating agentic workflows without costly LLM invocations.

We propose FLORA (work**FLO**w g**RA**ph neural networks), Graph Neural Networks (GNNs) variants, as efficient predictors of agentic workflow performances. GNNs effectively capture graph-level information by leveraging a message-passing mechanism to encode node attributes and edge relationships, thereby generating comprehensive graph representations [8, 9, 18, 39]. Compared to LLMs, GNNs require significantly fewer parameters, facilitating efficient operation on local hardware, and have demonstrated success in high-throughput screening for rapid evaluation in drug discovery, material design, and biological research [7, 37]. Inspired by these capabilities, we propose utilizing GNNs to evaluate agentic workflow performances without execution, thus reducing the substantial computational and financial costs of LLM invocations. By representing agentic workflows as optimizable graphs, GNNs can efficiently predict their performances, which then serve as rewards to guide the optimization process. This method not only addresses inefficiencies in existing agentic workflow optimization but also introduces a novel and impactful application of GNNs in the development of LLM-based multi-agent systems.

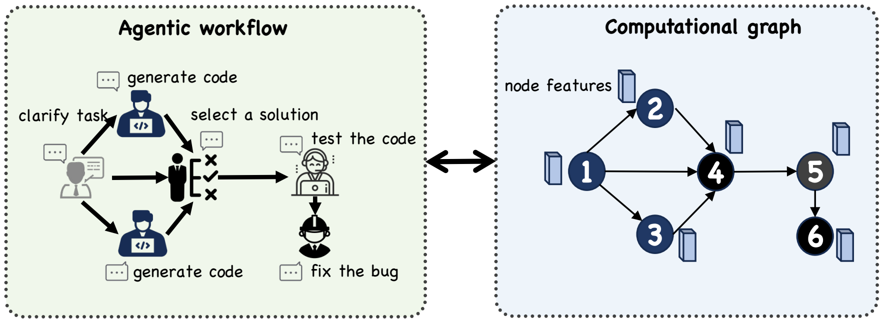


Figure 1: An illustration of an agentic workflow and its corresponding computational graph. Nodes are agents handling subtasks and edges are the task dependencies.

To support our proposed method and encourage technological iterations in the future, we construct the FLORA-Bench (work**FLO**w g**RA**ph benchmark), a unified platform for training and benchmarking predictors of agentic workflow performances. Key features of FLORA-Bench include:

• **Large scale with high quality.** The entire benchmark consists of **600k** workflow-task pairs and their binary labels denoting success or failure. Moreover, we control the quality of tasks and workflows, as well as inference results by a filtering process. This process ensures the high quality of our data, thereby guaranteeing the reliability and robustness of our experimental conclusions. More details are in section 4.

• **Dual evaluation metrics.** We employ 2 evaluation metrics in our benchmark: i) *accuracy*, which evaluates the correctness of predictions against the ground truth, reflecting the model's ability to assess absolute workflow quality; and ii) *utility*, which assesses the consistency between the predicted workflow rankings and the ground truth rankings, focusing on the model ability in determining the relative quality order of different workflows.

By conducting comprehensive experiments, we make the following key contributions:

• **Validate the ability of existing GNNs to predict agentic workflow performances.** We implement FLORA using existing GNNs. Results show that they are effective and efficient predictors of agentic workflow performances compared to alternative predictors, which serves as an experimental ground for FLORA and subsequent research.

• **Find a new application and drive the technological advancements of GNNs.** We explore applying GNNs as predictors of agentic workflow performances, which is an under-explored area. Moreover, while existing GNNs are promising, there is a pressing need to develop GNNs with enhanced generalization capabilities, enabling them to perform effectively across diverse domains. Our FLORA-Bench serves as a fundamental platform for this process.

• **Advance an efficient paradigm for agentic workflow optimization.** We propose a shift in agentic workflow optimization from a trial-and-error approach reliant on repeated LLM invocations to a prediction-driven paradigm powered by rapid evaluations using GNNs. This approach substantially enhances the efficiency of optimization, enabling faster self-evolution of agentic workflows.

2 Related Work

Agentic Workflows. Agentic workflows enable multiple agents to collaborate by sharing information and decomposing complex tasks into manageable subtasks, achieving greater effectiveness compared to single-agent systems [10]. They have demonstrated notable success in various areas such as code generation [1, 12, 19], math [38, 45], and reasoning [27, 34, 36]. Due to the task dependency between nodes, agentic workflows are modeled as graphs [29, 43, 47] or codes [17, 44].

Automated agentic workflow optimization. Recent works optimizing agentic workflows primarily fall into two categories: probability-based and LLM-guided methods. Probability-based approaches, such as GPTSwarm and G-Designer [43, 47], apply variants of the REINFORCE algorithm [35] to optimize edges and prompts across probabilistic distributions of Directed Acyclic Graphs (DAGs). In contrast, LLM-guided methods refine workflows by directly querying LLMs with workflow structures and performances. Examples include ADAS [17] modifying code-based workflows, EvoMAC [19] using textual backpropagation informed by environmental feedback, and AFLOW [44] employing a Monte Carlo Tree Search (MCTS) variant for iterative search. Despite promising results, both types of approaches rely heavily on repeated LLM invocations for workflow execution and performance evaluation. This incurs substantial computational, temporal, and financial overhead, limiting their efficiency and real-world practicality.

Graph Neural Networks. Graph Neural Networks (GNNs) have emerged as a cornerstone in graph representation learning with various architectures [3, 11, 22, 33], offering a powerful framework for capturing both local node attributes and global structural patterns through the message-passing mechanism. Foundational studies [9, 39] established that GNNs inherently encode relational dependencies between entities (nodes) and interactions (edges), enabling holistic graph-level representations. Moreover, their parameter efficiency, requiring orders of magnitude fewer parameters than LLMs [18], positions GNNs as an efficient solution for resource-constrained settings [8]. This efficiency drives their usage in high-throughput applications such as molecular property prediction, where the rapid iteration is critical [7, 37]. Inspired by these successes, we propose for the first time that GNNs are efficient predictors of agentic workflow performances.

3 Methodology

In this section, we formalize the key concepts of agentic workflows and our task: agentic workflow performance prediction. Building on these preliminaries, we formally present our proposed method, the workflow graph neural network, which utilizes GNNs as predictors of agentic workflow performances.

3.1 Model Agentic Workflows as Graphs

Notations. In agentic workflows, agents communicate and connect to others through task dependencies. Intuitively, the agentic workflows can be modeled as graphs. Therefore, we formalize the agentic workflow involving N agents as a Directed Acyclic Graph (DAG), $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{P}\}$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ represents the set of agents, with v_i denoting the agent i, and $\mathcal{E}$ represents the set of edges defining the connections between agents. Additionally, we define $\mathcal{P} = \{p_1, p_2, \ldots, p_N\}$ as the system prompt set, where p_i corresponds to the system prompt of the agent i. The agent i receives the task instruction T and outputs from its predecessor agents, which is formulated as:

$$\mathcal{X}_i = \{T\} \cup \{y_j : v_j \in \mathcal{N}_i^{(\text{in})}\}, \tag{1}$$

where $\mathcal{X}_i$ denotes the input of agent i, $\mathcal{N}_i^{(\text{in})}$ denotes the predecessor agents of the agent i, and y_j is the output of the agent j. For the agent i, the output y_i is derived by querying a LLM with the input $\mathcal{X}_i$ as follows,

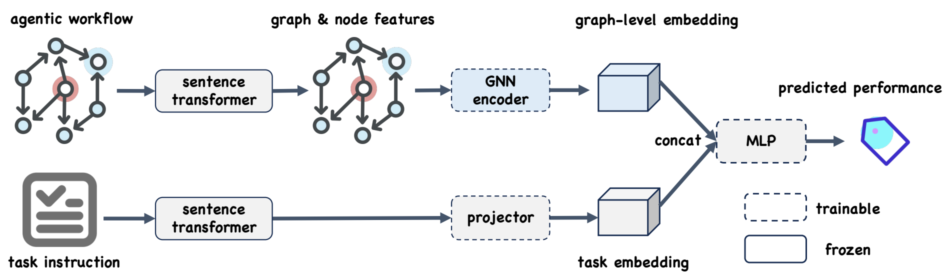


Figure 2: Architecture of workflow graph neural network, a framework for predicting agentic workflow performances with GNNs.

$$y_i = M(\mathcal{X}_i, p_i), \tag{2}$$

where $M(\cdot)$ denotes the invoked LLM and p_i is the system prompt of agent i, describing the subtask assigned to it.

Agentic workflow execution. Given a task T, the agentic workflow $\mathcal{G}$ autonomously executes agents recursively in topological order, as described by eqs. (1) and (2). Upon completing the execution of all agents, the response of this agentic workflow is derived as: $r = f_M(\mathcal{G}, T)$, where f_M is the execution process driven by M. Note that due to the sampling in the decoding of M, r is random.

Agentic workflow evaluation. Given the agentic workflow's response r, we can evaluate its performance. The performance of the agentic workflow e can be obtained as

$$e = U(r, T) = U(f_M(\mathcal{G}, T), T), \tag{3}$$

where $U(\cdot, \cdot)$ is the criterion function, which is usually implemented through either human evaluation or LLM-as-a-judge. For example, in the current literature of agentic workflow for coding, [17, 19, 43, 47], each coding task is associated with human-designed unit tests or the evaluation using GPT-4o.

According to eq. (3), the generation of ground-truth performances involves the complete inference of the agentic workflow and the rigorous evaluation, which is costly with both computational and financial overhead. This poses a huge challenge for the improvement, optimization and iteration of the agentic workflows.

3.2 Agentic Workflow Performance Prediction Task

Here, we present a new direction to evaluate agentic workflows without costly LLM invocations; that is, predicting the performances of agentic workflows through a lightweight model. We detail the model paradigm for this task and the corresponding learning objective.

Model paradigm. In this task, we require a predictor taking the agentic workflow and the task instruction as inputs to predict the corresponding performance. The predicted performance of the agentic workflow $\mathcal{G}$ on the task instruction T is obtained as

$$\hat{e} = \texttt{Preditor}_{\Theta}(\mathcal{G}, T), \tag{4}$$

where Θ denotes the learnable parameters of the predictor. When the predictor is lightweight and precise, it is extremely efficient to use the predictor to guide the optimization of an agentic workflow.

Learning objective. To train this predictor, the learning objective is to ensure the predicted performance $\hat{e}$ closely aligns with the ground-truth performance e. Mathematically, the learning objective is then,

$$\min_{\Theta} \mathbb{E}_{(\mathcal{G},T)} \left[\mathcal{L}\left(e, \hat{e}\right) \right], \tag{5}$$

where $\mathcal{L}(\cdot, \cdot)$ is a function that quantifies the difference between the ground truth and predicted performance, which could be specifically defined according to the type of prediction such as classification or regression. Importantly, this function measures the performance of $\texttt{Preditor}_{\Theta}(\cdot, \cdot)$, but not the agentic workflow performance.

3.3 GNNs as Predictors of Agentic Workflow Performances

In this section, we propose predicting the performances of the agentic workflows through GNNs; that is, the predicted performance of the agentic workflow $\mathcal{G}$ on the task instruction T is obtained as $\hat{e} = \texttt{FLOW-GNN}_\Theta(\mathcal{G}, T)$, where $\texttt{FLOW-GNN}_\Theta(\cdot, \cdot)$ is the workflow graph neural network. The intuition is that the agentic workflow is inherently a computational graph and the graph neural networks can exploit the graph structure effectively and efficiently. The proposed workflow graph neural network involves three stages.

Stage 1: Workflow encoding. This stage utilizes GNNs to encode the given agentic workflow $\mathcal{G}$ into a graph-level embedding for prediction. Given the system prompt p_i of the agent i, We first generate the node feature $\boldsymbol{x}_i$:

$$\boldsymbol{x}_i \leftarrow \texttt{Enc}(p_i) : v_i \in \mathcal{V}, \tag{6}$$

where $\texttt{Enc}(\cdot) : p_i \mapsto \mathbb{R}^{d_0}$ is a sentence transformer and d_0 is the output dimension. After this process, The node features $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N \times d_0}$ is derived. Given $\boldsymbol{X}$ and $\mathcal{G}$, GNNs generate the graph-level embedding $\boldsymbol{g}$ as,

$$\boldsymbol{g} = \texttt{GNN}_\theta\left(\mathcal{G}, \boldsymbol{X}\right), \tag{7}$$

where $\texttt{GNN}_\theta(\cdot, \cdot)$ denotes a GNN parameterized by θ[2]. Specifically, $\texttt{GNN}_\theta(\cdot, \cdot)$ utilizes the graph structure to produce informative node embeddings through iterative processes of message passing and information fusion. The final graph-level representation is then derived by applying a pooling function to these node embeddings [41]. This detailed process can be formulated as follows,

$$\begin{aligned}
\boldsymbol{m}_i^{(l)} &= \texttt{Aggregate}^{(l)}\left(\left\{\boldsymbol{h}_j^{(l-1)} : v_j \in \mathcal{N}_i^{(\text{in})}\right\}\right), \\
\boldsymbol{h}_i^{(l)} &= \texttt{Propagate}^{(l)}\left(\boldsymbol{h}_i^{(l-1)}, \boldsymbol{m}_i^{(l)}\right), \\
\boldsymbol{g} &= \texttt{Pool}\left(\{\boldsymbol{h}_i^{(L)} \mid v_i \in \mathcal{V}\}\right),
\end{aligned} \tag{8}$$

where $\boldsymbol{h}_i^{(l)}$ is the embeddings of the node i at layer l, $\mathcal{N}_i$ is the set of neighbors of node i, and $\boldsymbol{m}_i^{(l)}$ is the neighborhood embeddings for the node i at layer l. The initial node embeddings are set as $\boldsymbol{h}_i^{(0)} = \boldsymbol{x}_i$. The $\texttt{Aggregate}^{(l)}(\cdot)$ function performs message passing, aggregating features from the neighbors of node i to generate the neighborhood embeddings $\boldsymbol{m}_i^{(l)}$. The $\texttt{Propagate}^{(l)}(\cdot)$ function updates the embeddings of node i with previous embeddings at layer $l-1$ and the generated neighborhood embeddings at layer l. Finally, $\texttt{Pool}(\cdot)$ represents the pooling operation that takes the node embeddings from the final layer to generate $\boldsymbol{g}$.

Stage 2: Task encoding. The task instruction T is encoded into a dense representation $\boldsymbol{t} = \texttt{Proj}(\texttt{Enc}(T))$, where $\texttt{Proj}(\cdot)$ represents a projection module implemented by Multi-Layer Perceptrons (MLPs). This step transforms the raw task instruction into computational embeddings.

Stage 3: Prediction. We fuse the graph-level embedding $\boldsymbol{g}$ and the task embedding $\boldsymbol{t}$ through concatenation, yielding the combined embedding $\boldsymbol{o} = \texttt{Concat}(\boldsymbol{g}, \boldsymbol{t})$, where $\texttt{Concat}(\cdot, \cdot)$ denotes the concatenation operation. The predicted performance is obtained by,

$$\hat{e} = \texttt{MLP}(\boldsymbol{o}), \tag{9}$$

where $\texttt{MLP}(\cdot)$ denotes a multi-layer perceptron. The implementation of $\texttt{MLP}(\cdot)$ can be adapted to different types of prediction tasks such as classification and regression, which ensures the generality of the workflow graph neural network.

The architecture of the proposed workflow graph neural networks is summarized in fig. 2.

4 FLORA-Bench

In this section, we propose the **FLORA-Bench** (work**FLO**w g**RA**ph benchmark), an original large-scale benchmark designed to evaluate GNNs as predictors of agentic workflow performances. FLORA-Bench consists of **600k** workflow-task pairs with annotated performance data across coding, mathematics, and reasoning domains. In the following subsections, we provide a detailed introduction of its construction, key statistics, and evaluation metrics.

[2] Here θ represents learnable parameters of GNNs, which is different from Θ.

4.1 Benchmark Construction

FLORA-Bench is based on five representative datasets spanning three key topics frequently studied in the agentic workflow literature: coding, mathematics, and reasoning. Specifically, it incorporates the HumanEval [4] and MBPP [2] for coding, GSM8K [6] and MATH [14] for mathematics, and MMLU [13] for reasoning. We construct FLORA-Bench via a three-stage process: 1) workflow generation, 2) task filtering, and 3) label generation. The construction process is summarized in fig. 5.

Workflow generation. To obtain a diverse and high-quality set of agentic workflows as inputs for the agentic workflow performance prediction task, we derive workflows from the optimization processes of two state-of-the-art agentic workflow optimization systems: $G\text{-}Designer$ (GD) [43] and $AFLOW$ (AF) [44]. We run GD and AF on each task in our selected datasets and extract the agent connections and system prompts from all generated agentic workflows. An extracted agentic workflow is illustrated in fig. 6, more details of the extraction process are in appendix A.2.

Task filtering. After generating agentic workflows for each task in our selected datasets, we apply a filtering process to ensure the high quality of data in FLORA-Bench. Specifically, we remove tasks that are either too simple or too difficult, as such tasks provide limited value for performance prediction. To achieve this, we first infer a subset of workflows on tasks from the selected datasets and compute the success rate for each task. Consequently, we filter out tasks that achieve success rates out of a certain range. The statistics of the filtered tasks are presented in table 5, with further details provided in appendix A.3.

Label generation. After filtering out unsuitable tasks, we get the performance label for each agentic workflow by running inference on the remaining tasks. The output of this inference stage serves as the ground-truth performance, denoted as $e \in \{0, 1\}$. To ensure the stability of e, we perform inference three times using GPT-4o-mini, setting the temperature to 0 to strictly control randomness in the outputs. The evaluation criterion U is predefined by the source datasets: pass@1 is used for HumanEval and MBPP, while accuracy is used for MATH, GSM8K, and MMLU, as summarized in table 5. We denote the final dataset as $\mathcal{D} = \{d_i(\mathcal{G}_i, T_i, e_i) | e_i \in \{0, 1\}\}_{i=1}^{|\mathcal{D}|}$. More details about the criterion can be found in appendix A.4

4.2 Benchmark Statistics

FLORA-Bench consists of six domains: Coding-GD, Coding-AF, Math-GD, Math-AF, Reason-GD, and Reason-AF. Each domain is defined by a task domain (coding, mathematics, or reasoning) and a system domain (GD or AF). For instance, in the Coding-GD domain, the dataset is constructed using agentic workflows generated by the GD system and tasks sourced from the coding domain. An overview of FLORA-Bench is provided in table 6, illustrating its comprehensive coverage and large scale. To facilitate model training and evaluation, we randomly split each domain into $\mathcal{D}^{\text{train}}$, $\mathcal{D}^{\text{val}}$, and $\mathcal{D}^{\text{test}}$ according to a $0.8 : 0.1 : 0.1$ ratio.

4.3 Evaluation Metrics

Our FLORA-Bench employs two evaluation metrics: *accuracy* and *utility*. Accuracy quantifies how well a model predicts agentic workflow performance and is defined as:

$$\textit{accuracy} = \frac{1}{|\mathcal{D}^{\text{test}}|} \sum_{i=1}^{|\mathcal{D}^{\text{test}}|} \mathbf{1}\left(\hat{e}_i = e_i\right), \tag{10}$$

where $|\mathcal{D}^{\text{test}}|$ is size of test dataset, $\hat{e}_i$ is predicted performance, e_i is ground-truth performance, and $\mathbf{1}(\cdot)$ is the indicator function, which returns 1 if $\hat{e}_i = e_i$, and 0 otherwise. Utility evaluates the consistency between the workflow rankings predicted by the GNNs and the ground truth rankings, with an emphasis on the model's ability to determine the relative order of different workflows. To obtain utility, we firstly calculate the ground truth success rate s_i and predicted success rate $\hat{s}_i$ of $\mathcal{G}_i$ by averaging e and $\hat{e}$ of $\mathcal{G}_i$ on all the tasks in $\mathcal{D}^{\text{test}}$. Then we rank the workflows and extract top-k workflows respectively, according to s_i and $\hat{s}_i$, which produces 2 ordered set of workflows $\mathcal{H} = \{\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_k\}$ and $\hat{\mathcal{H}} = \{\mathcal{G}_1^{'}, \mathcal{G}_2^{'}, \dots, \mathcal{G}_k^{'}\}$. Finally, we calculate utility by:

$$\textit{utility} = \frac{1}{k} \sum_{i=1}^{k} \mathbf{1}(\mathcal{G}_i^{'} \in \mathcal{H}), \tag{11}$$

where $\mathbf{1}(\cdot)$ is the indicator function that returns 1 if $\mathcal{G}_i^{'} \in \mathcal{H}$, and 0 otherwise.

5 Experiments

We validate the effectiveness and efficiency of FLORA by answering 5 research questions with extensive experiments on FLORA-Bench: **(RQ1)** Can existing GNNs demonstrate promising performance in this prediction task? **(RQ2)** Are the GNNs' prediction performances robust when LLM driving the agents is different? **(RQ3)** Can GNNs generalize well across domains? **(RQ4)** Whether using GNNs as predictors benefit the optimization of the agentic workflow? **(RQ5)** Are GNNs better than graph-agnostic alternative predictors?

5.1 Experimental Setups

Datasets & Models. Our experiments are conducted on our FLORA-Bench across 6 domains mentioned in table 6.Moreover, we select five GNN-based models: 1) **GCN** [22], 2) **GAT** [33], 3) **GCNII** [5], 4) **Graph Transformer** [31] denoted as GT, and 5) **One For All** [26] denoted as OFA. In section 5.6, we compare these GNN-based models with three alternative predictors: 1) **MLP** that processes the raw text-attributed workflow directly using a multilayer perceptron instead of a GNN encoder; 2) **FLORA-Bench-Tuned Llama 3B** [32] denoted as Llama, fine-tuned on the training dataset in 6 domains of FLORA-Bench; and 3) **DeepSeek V3** [25] denoted as DeepSeek, a mixture-of-experts (MoE) architecture-based LLM, which is prestine.

Implementation Details. For the MLP and GNN baselines, we employ a 2-layer backbone with a hidden dimension of 512. The models are optimized using the Adam optimizer [21] with a learning rate of 1×10^{-4} and a weight decay of 5×10^{-4}. Training is conducted for 200 epochs on a single NVIDIA RTX-4090 GPU. The best model checkpoint is selected based on the highest *accuracy* on the validation set. For fine-tuning Llama 3B, we utilize Lora [16] with the workflow and task text as input, training for 10 epochs. During inference, the temperature is set to 0 to ensure deterministic predictions. For DeepSeek V3, we directly call its API and generate predictions using prompts with temperature also set to 0.

Evaluation Metrics. We use *accuracy* and *utility*, which are defined in eqs. (10) and (11) respectively.

5.2 RQ1: Can existing GNNs demonstrate promising performances in this prediction task?

The feasibility of GNNs as predictors of agentic workflow performances is the foundation of FLORA and promotes future research. To validate this, we train and test GNNs on the 6 domains of FLORA-Bench respectively. We evaluate GNNs' performances by *accuracy* and *utility* and report the results in table 1. These results show that all the GNN baselines demonstrate promising prediction *accuracy* and *utility* in most scenarios with an average *accuracy* of **0.7812** and an average *utility* of **0.7227** over 6 domains. Generally, GNNs have demonstrated significant potential in predicting agentic workflow performances. This supports the feasibility of GNNs as predictors of agentic workflow performances.

Table 1: Main results of GNN baselines on the test set. The best and second-best results for each metric are in **bold** and underlined. GNNs demonstrate promising results in predicting agentic workflow performances.

Model	Coding-GD		Coding-AF		Math-GD		Math-AF		Reason-GD		Reason-AF	
	acc.	*util.*	*acc.*	*util.*	*acc.*	*util.*	*acc.*	*util.*	*acc.*	*util.*	*acc.*	*util.*
GCN	0.8423	0.7931	0.8345	0.6716	0.6412	0.6303	**0.8010**	**0.7491**	0.7222	0.5918	0.8674	0.8909
GAT	0.8514	0.7950	0.8223	0.6941	**0.6484**	0.6232	0.7985	0.7400	0.7216	0.5944	0.8668	0.8966
GCNII	0.8381	0.7845	0.8290	0.7054	0.6356	0.6602	0.7936	0.7100	**0.7229**	0.5910	**0.8708**	**0.9066**
GT	**0.8524**	**0.8020**	0.8250	0.6379	0.6325	0.6497	0.8000	0.7267	0.7226	0.6092	0.8692	0.8647
OFA	0.8374	0.7593	**0.8374**	**0.7593**	0.6317	**0.6665**	0.7985	0.6686	**0.7229**	**0.6035**	0.8142	0.9064
Avg.	0.8443	0.7868	0.8296	0.6936	0.6379	0.6460	0.7983	0.7189	0.7224	0.5980	0.8577	0.8930

Table 2: Cross-system generalization results. Compared with table 1, GNNs exhibit a notable drop in performance when tested on unseen domains.

Model	Coding-GD-AF		Coding-AF-GD		Math-GD-AF		Math-AF-GD		Reason-GD-AF		Reason-AF-GD	
	acc.	*util.*	*acc.*	*util.*	*acc.*	*util.*	*acc.*	*util.*	*acc.*	*util.*	*acc.*	*util.*
GCN	0.5821	0.5733	0.5676	0.5429	**0.6757**	0.5463	0.4964	**0.5192**	0.5751	0.5337	0.6137	0.5413
GAT	0.5929	0.5968	0.5725	0.5605	0.6634	0.5270	0.4829	0.4871	0.5607	0.5038	0.5703	0.5312
GCNII	0.5875	0.6117	**0.6416**	**0.6267**	0.6732	0.5296	0.4885	0.5056	0.5593	0.5219	**0.6555**	0.5276
GT	0.6052	**0.6144**	0.6083	0.5839	0.5897	0.5749	0.4773	0.4665	0.5650	**0.5486**	0.5588	0.4887
OFA	**0.6201**	0.5457	0.5897	0.5325	0.5872	**0.6123**	**0.5060**	0.5102	**0.5940**	0.5417	0.6384	**0.5522**
Avg.	0.5976	0.5884	0.5959	0.5693	0.6378	0.5580	0.4902	0.4977	0.5708	0.5299	0.6073	0.5282

5.3 RQ2: Are the GNNs' prediction performances robust when LLM driving the agents is different?

In practical applications, different LLMs may be used to drive the agentic workflow. We wonder whether GNNs' prediction performances are robust when LLM driving the agents is different. To validate the robustness of GNNs' performances, which is important for real-world applications, we use 4 LLM: GPT-4o-mini, DeepSeek V3, Qwen 7B, and Mistral 7B, to infer the same agentic workflows and tasks collected during benchmark construction to generate 4 datasets. We train and test GNNs on these datasets respectively. We report the *accuracy* and *utility* in table 8. It can be observed that prediction accuracies of GNNs remain above 0.8 across 4 datasets. Therefore, we believe that GNNs' performances remain robustly strong when LLM driving the agents is different.

5.4 RQ3: Can GNNs generalize well across domains?

To evaluate the crucial generalization ability of GNNs for predicting agentic workflow performances, we conducted cross-system-domain and cross-task-domain tests. The cross-system-domain test involved training and testing GNNs on different workflow domains while keeping the task domain the same (e.g., train Coding-GD, test Coding-AF, denoted Coding-GD-AF). Conversely, the cross-task-domain test used different task domains for training and testing while keeping the workflow domain constant (e.g., train Coding-AF, test Reason-AF, denoted Coding-Reason). We report *accuracy* and *utility* in tables 2 and 3. These experiments revealed a significant decrease in GNN performance in the test domains compared to within-domain results in table 1, with GNNs failing to predict accurately. This indicates that existing GNNs lack generalization ability across system and task domains.

Table 3: Cross-task-domain generalization results. Compared with table 1, GNNs show a considerable performance drop when transferring across tasks.

Model	Coding-Math		Coding-Reason		Math-Reason		Math-Coding		Reason-Coding		Reason-Math	
	acc.	*util.*	*acc.*	*util.*	*acc.*	*util.*	*acc.*	*util.*	*acc.*	*util.*	*acc.*	*util.*
GCN	0.4889	0.5407	0.5261	0.5329	0.4938	0.4669	0.5007	0.4875	0.3256	0.5053	0.3342	0.5057
GAT	0.4595	0.4942	**0.5371**	0.5790	0.4683	0.3890	0.5102	0.4740	0.3379	**0.5262**	0.3342	0.5110
GCNII	**0.5602**	0.4449	0.5344	0.4593	**0.5038**	**0.4736**	0.3948	0.5155	0.3813	0.5135	0.3661	**0.5793**
GT	0.4767	0.5618	**0.5371**	**0.5795**	0.4790	0.4363	0.5400	0.5620	0.6092	0.5237	**0.4177**	0.5291
OFA	0.3661	**0.6111**	0.5033	0.3982	0.4492	0.4588	**0.6540**	**0.5624**	**0.6336**	0.5060	0.3808	0.4527
Avg.	0.4703	0.5305	0.5276	0.5098	0.4788	0.4449	0.5199	0.5203	0.4575	0.5149	0.3666	0.5156

5.5 RQ4: Whether using GNNs as predictors benefits the optimization of the agentic workflow?

We investigate whether GNNs enhance agentic workflow optimization efficiency and quantifies any performance loss. Using the AFLOW [44] platform, we employed a GCN predictor to provide performance estimates as rewards during agentic workflow optimization across the HumanEval, MMLU, and MBPP benchmarks. The scores of these optimized agentic workflows were then determined by their success rates on test tasks. We compared this GCN-based approach to 3 baselines:

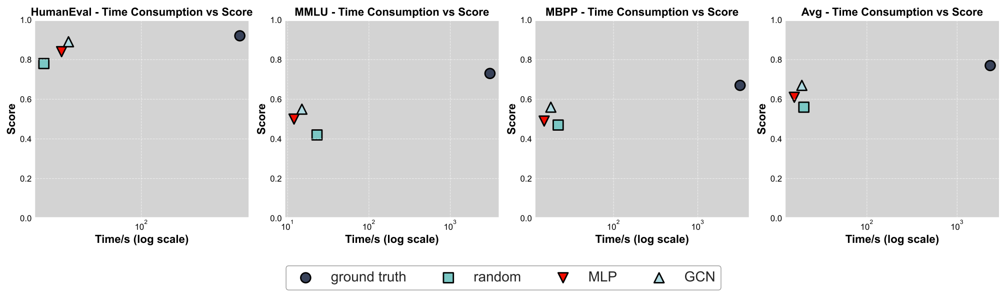


Figure 3: GCN as the predictor indeed benefits the optimization of the agentic workflow in terms of both effectiveness and efficiency.

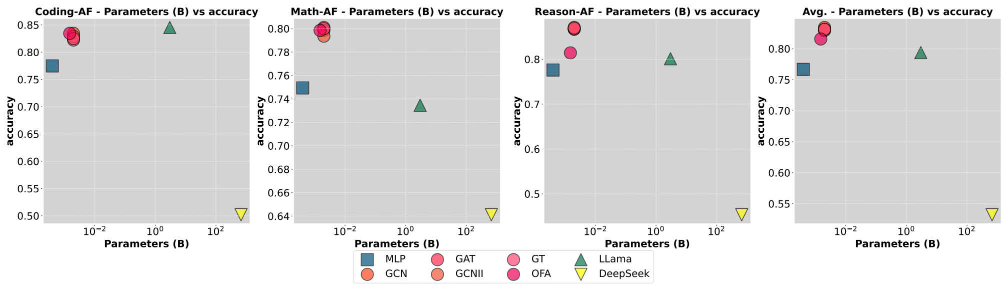


Figure 4: GNNs are more effective and efficient than others.

’ground truth’ (using actual inferred performances as rewards), ’random’ (using random predictions), and ’MLP’ (using MLP’s predictions). We report the duration of the optimization process and scores across these benchmarks and their average in fig. 3, with detailed results in table 7. It can be observed from these results that the GCN predictor significantly accelerates refinement cycles by 125 times while the performance loss is 0.1 on average. Furthermore, GCN outperformed the ’random’ baseline by a 0.11 performance gain, attributed to its accurate predictions. These findings support GNNs’ potential for advancing an efficient paradigm for agentic workflow optimization.

5.6 RQ5: Are GNNs better than graph-agnostic alternative predictors?

We compared GNN-based predictors with alternative predictors such as LLMs, as described in section 5.1. We report *accuracy* in fig. 4. More results are shown in table 9. It indicates that GNNs outperform all alternative predictors in most scenarios. Notably, we observe that DeepSeek-V3, despite its significantly larger parameter volume, fails to achieve accurate predictions compared to GNNs or MLP. This is likely due to the lack of relevant training data for predicting agentic workflow performances and it is not tuned, further highlighting the advantages of using GNNs as predictors. Moreover, the FLORA-Bench-Tuned Llama 3B can outperform GNNs in the Coding-AF domain. However, considering the training cost and inference time, it is not an efficient alternative. Overall, GNNs are more efficient compared to alternatives.

6 Conclusions

In this paper, we propose **FLORA** (work**FLO**w g**RA**ph neural networks) as efficient predictors of agentic workflow performances. By conducting extensive experiments we ultimately arrive at the following conclusions: GNNs are simple yet effective predictors and demonstrate significant practical benefits. These conclusions support new applications of GNNs and a novel direction towards automating agentic workflow optimization.

7 Limitations and Future Work

As we pointed out in the experiment, GNNs as predictors for agentic workflows suffer from issues such as poor generalization performance. This problem requires joint resolution in future work, and we propose potential directions for solutions here.

The most straightforward solution for generalization is large-scale pretraining, requiring large amounts of tasks and workflows of high quality from different domains. However, given the associated costs and overhead, we believe this is not a viable solution at present. Although large-scale pretraining is challenging, we observed that GNNs converge efficiently within a single domain. Thus, techniques like transfer learning, domain adaptation, and online learning [23, 30, 46] might enable GNNs to generalize efficiently. This may be a promising approach to improving GNNs' generalization ability.

Acknowledgements

This research is supported by National Natural Science Foundation of China under Grant NSFC 62450162. Part of this work is done when Yuanshuo Zhang was at Beijing Institute for General Artificial Intelligence.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 1, 3

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. 6, 15

[3] Ling-Hao Chen, Yuanshuo Zhang, Taohua Huang, Liangcai Su, Zeyi Lin, Xi Xiao, Xiaobo Xia, and Tongliang Liu. Erase: Error-resilient representation learning on graphs for label noise tolerance. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 270–280, 2024. 3

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. 6, 15

[5] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. PMLR, 2020. 7

[6] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems, 2021. *URL https://arxiv. org/abs/2110.14168*, 2021. 6, 15

[7] Hongwei Du, Jiamin Wang, Jian Hui, Lanting Zhang, and Hong Wang. Densegnn: universal and scalable deeper graph neural networks for high-performance property prediction in crystals and molecules. *npj Computational Materials*, 10(1):292, 2024. 2, 3

[8] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023. 2, 3

[9] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019. 2, 3

[10] T Guo, X Chen, Y Wang, R Chang, S Pei, NV Chawla, O Wiest, and X Zhang. Large language model based multi-agents: A survey of progress and challenges. In *33rd International Joint Conference on Artificial Intelligence (IJCAI 2024)*. IJCAI; Cornell arxiv, 2024. 3

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. 3

[12] Junda He, Christoph Treude, and David Lo. Llm-based multi-agent systems for software engineering: Literature review, vision and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 2025. 1, 3

[13] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020. 6, 16

[14] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021. 6, 15

[15] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023. 1

[16] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685, 2021. 7

[17] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024. 1, 3, 4

[18] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020. 2, 3

[19] Yue Hu, Yuzhu Cai, Yaxin Du, Xinyu Zhu, Xiangrui Liu, Zijie Yu, Yuchen Hou, Shuo Tang, and Siheng Chen. Self-evolving multi-agent collaboration networks for software development. *arXiv preprint arXiv:2410.16946*, 2024. 1, 2, 3, 4

[20] Yizhe Huang, Xingbo Wang, Hao Liu, Fanqi Kong, Aoyang Qin, Min Tang, Xiaoxi Wang, Song-Chun Zhu, Mingjie Bi, Siyuan Qi, et al. Adasociety: An adaptive environment with social structures for multi-agent decision-making. *arXiv preprint arXiv:2411.03865*, 2024. 1

[21] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 7

[22] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 3, 7

[23] Lin Lan, Pinghui Wang, Xuefeng Du, Kaikai Song, Jing Tao, and Xiaohong Guan. Node classification on graphs with few-shot novel labels via meta transformed network embedding. *Advances in Neural Information Processing Systems*, 33:16520–16531, 2020. 10

[24] Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008, 2023. 1

[25] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024. 7

[26] Hao Liu and Jiarui Feng. Lecheng kong, ningyue liang, dacheng tao, yixin chen, and muhan zhang. one for all: Towards training one graph model for all classification tasks. *arXiv preprint arXiv:2310.00149*, pages 1–2, 2023. 7

[27] Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*, 2023. 1, 2, 3

[28] Siyuan Qi, Shuo Chen, Yexin Li, Xiangyu Kong, Junqi Wang, Bangcheng Yang, Pring Wong, Yifan Zhong, Xiaoyuan Zhang, Zhaowei Zhang, et al. Civrealm: A learning and reasoning odyssey in civilization for decision-making agents. *arXiv preprint arXiv:2401.10568*, 2024. 1

[29] Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Benchmarking agentic workflow generation. *arXiv preprint arXiv:2410.07869*, 2024. 3

[30] Luana Ruiz, Luiz Chamon, and Alejandro Ribeiro. Graphon neural networks and the transferability of graph neural networks. *Advances in Neural Information Processing Systems*, 33:1702–1712, 2020. 10

[31] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020. 7

[32] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 7

[33] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 3, 7

[34] Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, et al. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *arXiv preprint arXiv:2401.04398*, 2024. 1, 3

[35] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992. 3

[36] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023. 1, 3

[37] Xing Wu, Hongye Wang, Yifei Gong, Dong Fan, Peng Ding, Qian Li, and Quan Qian. Graph neural networks for molecular and materials representation. *Journal of Materials Informatics*, 3(2), 2023. 2, 3

[38] Yiran Wu, Feiran Jia, Shaokun Zhang, Hangyu Li, Erkang Zhu, Yue Wang, Yin Tat Lee, Richard Peng, Qingyun Wu, and Chi Wang. Mathchat: Converse to tackle challenging math problems with llm agents. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024. 1, 3

[39] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020. 2, 3

[40] Wei Xiong, Chengshuai Shi, Jiaming Shen, Aviv Rosenberg, Zhen Qin, Daniele Calandriello, Misha Khalman, Rishabh Joshi, Bilal Piot, Mohammad Saleh, et al. Building math agents with multi-turn iterative preference learning. *arXiv preprint arXiv:2409.02392*, 2024. 1

[41] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. 5

[42] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024. 2

[43] Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*, 2024. 3, 4, 6, 14

[44] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024. 1, 2, 3, 6, 8, 15

[45] Qihuang Zhong, Kang Wang, Ziyang Xu, Juhua Liu, Liang Ding, Bo Du, and Dacheng Tao. Achieving> 97% on gsm8k: Deeply understanding the problems makes llms perfect reasoners. *arXiv preprint arXiv:2404.14963*, 2024. 3

[46] Qi Zhu, Natalia Ponomareva, Jiawei Han, and Bryan Perozzi. Shift-robust gnns: Overcoming the limitations of localized graph training data. *Advances in Neural Information Processing Systems*, 34:27965–27977, 2021. 10

[47] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024. 2, 3, 4

A Details of Benchmark Construction

A.1 Benchmark Construction Pipeline

In fig. 5, we show the pipeline of FLORA-Bench construction.

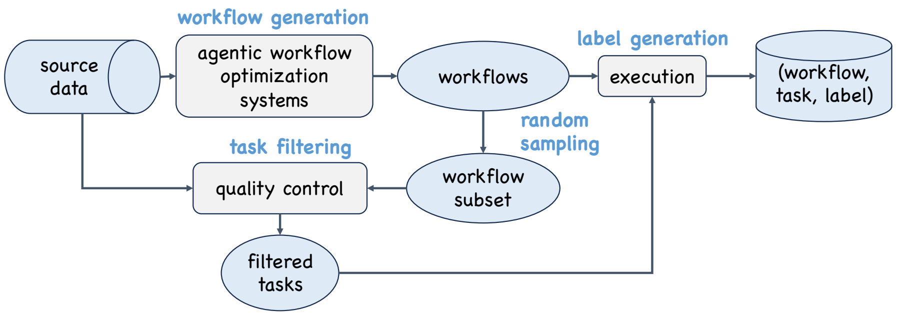
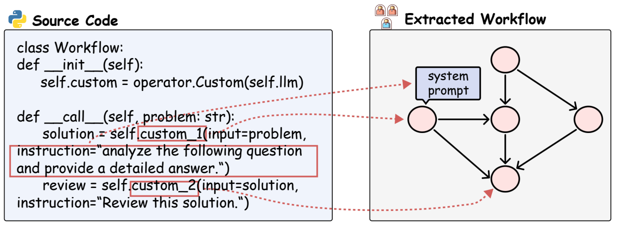


Figure 5: Pipeline of benchmark construction.

A.2 Workflow Extraction

In GD system, we directly extract the connections between agents from Directed Acyclic Graphs (DAGs). This approach ensures that the inherent structural relationships within the DAGs are faithfully captured and represented in the workflow. Moreover, in AF system, we parse the Python source code into an Abstract Syntax Tree (AST). This enables us to capture the information transmission logic of agents through variable propagation. Specifically, each new instance of an agent call is treated as a distinct code unit. When the output variables of one agent instance are used as inputs for the next agent instance, an edge is created between the corresponding nodes in the workflow. Additionally, the specific instructions passed to agents in the source code are integrated with the agent system prompt and embedded as node attributes within the workflow.

For example, consider the workflow generated from a Python class Workflow in fig. 6. The first node corresponds to the first call to Custom_1 agent and the second node corresponds to the call to Custom_2, which reviews the solution generated by Custom_1. The edges between these nodes represent the flow of the solution from Custom_1 to Custom_2. The attributes of the first node include the instruction "analyze the following question and provide a detailed answer".



```
class Workflow:
def __init__(self):
    self.custom = operator.Custom(self.llm)

def __call__(self, problem: str):
    solution = self.custom_1(input=problem,
instruction="analyze the following question
and provide a detailed answer.")
    review = self.custom_2(input=solution,
instruction="Review this solution.")
```

Figure 6: Workflow Extraction from AF in FLORA-Bench.

A.3 Task Filter

To achieve a more uniform distribution of tasks within our dataset, we implemented a filtering mechanism designed to exclude tasks that are either too complex or too straightforward. Tasks that fall into these categories often provide limited value for training and evaluating predictive models, as they do not offer a balanced knowledge.

First, We sampled a smaller workflow subset from a specific dataset to capture a wide range of agent profiles. The workflows generated by GD [43] system are predefined with specific types and numbers of agents. Therefore, we sampled workflow subsets separately from each unique agent configuration

Table 4: Task filtering of FLORA-Bench.

Data Set	Original Task	Workflow Subset	Success Rate Range	Filtered Task
HumanEval	161	26	[0.1, 0.9]	31
MBPP	427	30	[0.2, 0.8]	202
MATH	605	150	[0.2, 0.8]	178
GSM8K	1319	42	[0.3, 0.9]	404
MMLU	14k+	20	[0.25, 0.75]	2400

Table 5: Task filtering results of FLORA-Bench.

Data Set	Original Task	Filtered Task	Eval Method
HumanEval	161	31	pass@1
MBPP	427	202	pass@1
MATH	605	178	accuracy
GSM8K	1319	404	accuracy
MMLU	14k+	2400	accuracy

to ensure diversity in agent types and interactions. The workflows generated by AF [44] system are designed with agent system prompts but feature unpredictable numbers of agents. This is due to the inherent constraints of the optimization framework, specifically the Monte Carlo Tree Search (MCTS) process used by AF, which limits the generation of large-scale workflows. As a result, we included all generated workflows in our sample pool to ensure comprehensive coverage within the scope of workflows that are feasible with the AF. This approach ensures that our dataset encompasses a variety of small-scale workflows, providing a comprehensive view of potential agent interactions.

To achieve balanced performance across tasks, we evaluated all tasks in the current dataset through the workflow subset three times using GPT-4o-mini, with the temperature parameter set to 0. This setup allows for consistent evaluations of task performance.

Then, we calculated the success rate for each task across the workflow subset within the current dataset. Tasks were retained if their success rates fell within a predefined range, as detailed in table 4. This range was carefully chosen to avoid including tasks that are either trivially easy (all-success) or intractably difficult (all-fail), thus preserving a challenging yet solvable benchmark.

For an example, we selected 26 workflows generated by GD for HumanEval[4] dataset as our subset. These workflows varied in the number of agents, ranging from 4 to 8, and included diverse agent types such as "Programming Expert", "Project Manager", "Test Analyst", "Bug Fixer" and "Algorithm Designer". After evaluating all 161 tasks from the original HumanEval [4] dataset across this workflow subset, we calculated the success rate for each task. Tasks were retained if their success rates fell within a predefined range of 0.1 to 0.9. This range was chosen to exclude tasks that were either trivially easy (success rate >0.9) or intractably difficult (success rate <0.1). This resulted in a selection of 31 tasks that demonstrated diverse performance across the workflow subset. These tasks were capable of distinguishing performance differences among various workflows, providing a balanced and challenging benchmark for our predictive models.

A.4 Details of Criterion.

The methods for generating binary labels vary depending on the nature and requirements of each dataset. Here, we provide a detailed overview of how binary labels are obtained from several key datasets used in our research.

- **HumanEval**[4] and **MBPP**[2]: For these datasets, the primary focus is on evaluating the functionality of the model's output code. Specifically, the code generated by the model is tested to determine whether it can pass predefined unit tests (pass@1). If the code successfully passes the unit tests, it is assigned a binary label of "1" (indicating correctness); otherwise, it receives a binary label of "0" (indicating failure).
- **MATH**[14] and **GSM8K**[6]: In these datasets, the model's output is evaluated based on the accuracy of the answers. The final answers in the Math dataset are wrapped and delimited with the `\boxed` command. For GSM8K, the model's output answer is compared with the ground

Table 6: Overview of 6 domains of our FLORA-Bench.

Domain	Coding-GD	Coding-AF	Math-GD	Math-AF	Reason-GD	Reason-AF
Num. of workflows	739	38	300	42	189	30
Avg. of nodes	5.96	6.11	6.06	5.49	5.97	6.58
Num. of tasks	233	233	782	782	2400	2400
Num. of samples	30683	7362	12561	4059	453600	72000

truth answer within a tolerance level of 1e-6. If the model's answer falls within the acceptable range of the ground truth answer, it is assigned a binary label of "1"; otherwise, it is labeled as "0".

- **MMLU**[13]: The binary label is determined by whether the model's output option matches the correct answer option. Each question in MMLU is a multiple-choice question with four options (A, B, C, and D). The model's performance is evaluated based on its ability to accurately identify the correct answer among the provided choices. If the model selects the correct option, it is labeled as "1"; otherwise, it is labeled as "0".

To obtain stable performance, the agentic workflows are inferred 3 times during the label generation in our benchmark construction.

A.5 Benchmark Statistics

B Visualization of Agentic Workflows

In fig. 7, the coding workflow starts with task initialization (Agent 1). Then Agent 1 connects to Agents 2, 3, and 4, who each fill different CodeBlocks independently. These components flow into Agent 5, which aggregates the generated answers into a cohesive solution. From there, Agent 6 reviews the solution for correctness before passing it to Agent 7 for final validation. Finally, Agent 8 make final response by receiving the task and the suggestions from Agent 7.

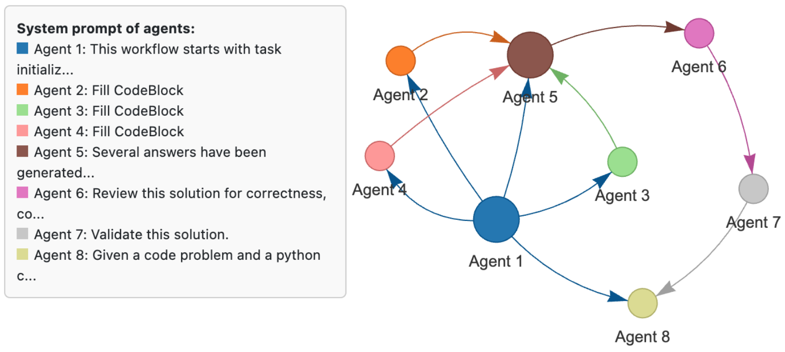
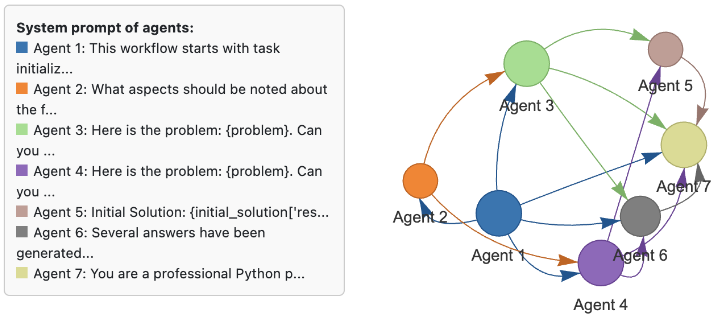

System prompt of agents:
Agent 1: This workflow starts with task initializ...
Agent 2: Fill CodeBlock
Agent 3: Fill CodeBlock
Agent 4: Fill CodeBlock
Agent 5: Several answers have been generated...
Agent 6: Review this solution for correctness, co...
Agent 7: Validate this solution.
Agent 8: Given a code problem and a python c...
Agent 1
Agent 2
Agent 3
Agent 4
Agent 5
Agent 6
Agent 7
Agent 8

Figure 7: Example of a extracted coding workflow from AF.

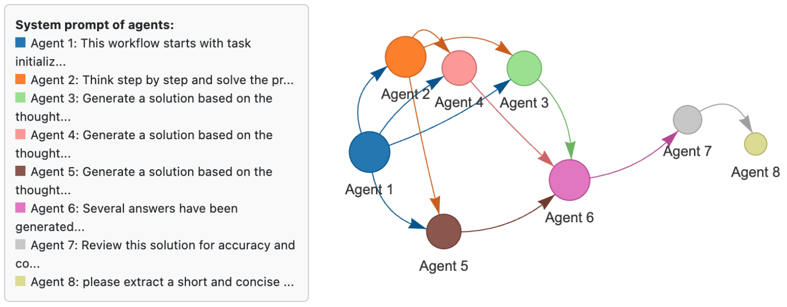
In fig. 8, the math workflow starts with task initialization (Agent 1). Then Agent 2 make some notes about solving this problem. After this, Agent 3 and 4 receive the notes and attempt to answer this problem. Their answers are passed to Agent 5, which reviews the answers and analyses the difference. Agent 6 receive the problem statement and the answers generated by Agent 3 and 4 to generate additional answer. Finally, the responses of Agent 1, 3, 4, 5, 6 are passed to Agent 7 to make final decision.

In fig. 9, Agent 1 initiates the process and delegates to multiple subsequent agents. Agent 2 serves as the analytical thinker who works step-by-step through the problem and feeds insights to Agents 3, 4, and 5, each of which generates different solution variations based on this thinking. These solutions flow into Agent 6, which aggregates and synthesizes the various answers. The workflow then moves to Agent 7, which reviews the consolidated solution for accuracy and correctness, before concluding with Agent 8, which extracts a short, concise final version.



System prompt of agents:
🟦 Agent 1: This workflow starts with task initializ...
🟧 Agent 2: What aspects should be noted about the f...
🟩 Agent 3: Here is the problem: {problem}. Can you ...
🟪 Agent 4: Here is the problem: {problem}. Can you ...
🟫 Agent 5: Initial Solution: {initial_solution['res...
⬛ Agent 6: Several answers have been generated...
🟨 Agent 7: You are a professional Python p...

Figure 8: Example of a extracted math workflow from AF.



System prompt of agents:
🟦 Agent 1: This workflow starts with task initializ...
🟧 Agent 2: Think step by step and solve the pr...
🟩 Agent 3: Generate a solution based on the thought...
🟥 Agent 4: Generate a solution based on the thought...
🟫 Agent 5: Generate a solution based on the thought...
🟪 Agent 6: Several answers have been generated...
⬜ Agent 7: Review this solution for accuracy and co...
🟨 Agent 8: please extract a short and concise ...

Figure 9: Example of a extracted Reason workflow from AF.

C Observation of Agentic Workflow Performances

To analyze the relation between number of nodes and their performances. We group agentic workflows according to their number of nodes and calculate their average success rate in the collected dataset. The statistics is shown in fig. 10. Results show that when the number of nodes is small, the workflows tend to perform poorly. Within the range from 2 to 5, the success rate increases with the number of nodes. However, as the number of nodes continues to increase, the rate of increase in success rate slows down until it saturates, and the success rate even starts to decrease as the graph complexity increases.

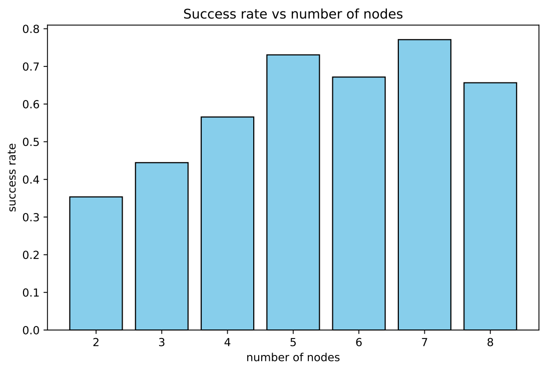


Figure 10: Success rate of workflows with different number of nodes.

D Details of Experiments.

D.1 Efficiency Improvement

We provide more detailed results of fig. 3 in table 7. It can be observed that GCN significantly accelerates refinement cycles by 125 times while the performance loss is 0.1 in average. And GCN outperforms the 'random' baseline by 0.11 performance gain because GCN correctly predicts performance. These results support GNNs' potential to advance an efficient and effective paradigm for agentic workflow optimization.

Table 7: Efficiency improvement in terms of **Time (s)** and **Cost ($)**, and the associated performance loss (**Score**). Results are reported on HumanEval, MMLU, and MBPP benchmarks, as well as their average.

Method	**HumanEval**			**MMLU**			**MBPP**			**Avg.**		
	Score	Time	Cost	Score	Time	Cost	Score	Time	Cost	Score	Time	Cost
Ground Truth	0.92	728	0.13	0.73	3073	6.23	0.67	3180	0.48	0.77	2327	2.28
Random	0.78	14	0.01	0.42	23	0.01	0.47	22	0.01	0.56	19.67	0.01
MLP	0.84	20	0.01	0.50	12	0.01	0.49	15	0.01	0.61	8	0.01
GCN	0.89	23	0.01	0.55	15	0.01	0.56	18	0.01	0.67	15.67	0.01

D.2 Detialed results of RQ2: Are the GNNs' prediction performances robust when LLM driving the agents is different?

See table 8.

Table 8: Prediction performance of GNNs across datasets generated by different LLMs. Results demonstrate the robustness and consistency of GNN predictors.

Model	**GPT-4o-mini**		**DeepSeek**		**Qwen-7B**		**Mistral-7B**	
	acc.	*util.*	*acc.*	*util.*	*acc.*	*util.*	*acc.*	*util.*
GCN	0.8294	**0.8040**	**0.8656**	**0.7569**	0.8671	<u>0.8448</u>	0.9258	0.8848
GAT	<u>0.8303</u>	<u>0.8025</u>	0.8442	0.7518	**0.8698**	0.8426	<u>0.9262</u>	<u>0.8872</u>
GCNII	0.8281	0.7948	0.8434	<u>0.7568</u>	0.8517	0.8271	0.9094	0.8589
GT	**0.8342**	0.7983	<u>0.8634</u>	0.7306	<u>0.8676</u>	**0.8465**	**0.9280**	**0.8887**
OFA	0.8124	0.7192	0.8473	0.7323	0.8451	0.8042	0.8913	0.8506

D.3 Comparison with Alternative Predictors

More detailed results of fig. 4 are shown in table 9. It indicates GNNs are simple yet efficient predictors compared to alternatives.

Table 9: Comparison with alternative predictors. Results are reported on the AF domains.

Model	**Coding-AF**		**Math-AF**		**Reason-AF**		**Avg.**	
	acc.	*util.*	*acc.*	*util.*	*acc.*	*util.*	*acc.*	*util.*
MLP	0.7748	0.6960	0.7494	0.6888	0.7760	**0.9299**	0.7667	0.7716
GCN	<u>0.8345</u>	0.6716	**0.8010**	**0.7491**	0.8674	0.8909	**0.8343**	0.7705
GAT	0.8223	0.6941	0.7985	<u>0.7400</u>	0.8668	0.8966	0.8292	**0.7769**
GCNII	0.8290	0.7054	0.7936	0.7100	**0.8708**	<u>0.9066</u>	0.8311	<u>0.7740</u>
GT	0.8250	0.6379	<u>0.8000</u>	0.7267	<u>0.8692</u>	0.8647	<u>0.8314</u>	0.7431
OFA	<u>0.8345</u>	<u>0.7104</u>	0.7985	0.6686	0.8142	0.9064	0.8157	0.7618
Llama	**0.8453**	**0.7512**	0.7346	0.6705	0.8011	0.8360	0.7937	0.7526
DeepSeek	0.5020	0.4267	0.6412	0.5097	0.4536	0.5036	0.5323	0.4800

Table 10: Expected calibration error (ECE) of predictions across different AF domains.

Model	Coding-AF	Math-AF	Reason-AF
GCN	0.0196	0.0505	0.0322
GAT	0.0201	0.0522	0.0271
GCNII	0.0179	0.0765	0.0279
GT	0.0238	0.0498	0.0295
OFA	0.0245	0.0568	0.0205

D.4 Analysis of Prediction Confidence Calibration

In table 10, we report the Expected Calibration Error (ECE), which measures the deviation between the model's predicted confidence and actual accuracy, reflecting its calibration quality. The ideal ECE is 0, indicating perfect alignment. Empirically, all GNNs are close to this in most scenarios, demonstrating strong calibration.