WorkTeam: Constructing Workflows from Natural Language with Multi-Agents

Hanchao Liu[†] **and Rongjun Li**[†] **and Weimin Xiong**[‡] **and Ziyu Zhou**[†] **and Wei Peng**[†]
[†]IT Innovation and Research Center, Huawei Technologies
[‡]National Key Laboratory for Multimedia Information Processing,
School of Computer Science, Peking University
{liuhanchao2, lirongjun3, zhouziyu8, peng.wei1}@huawei.com
wmxiong@pku.edu.cn

Abstract

Workflows play a crucial role in enhancing enterprise efficiency by orchestrating complex processes with multiple tools or components. However, hand-crafted workflow construction requires expert knowledge, presenting significant technical barriers. Recent advancements in Large Language Models (LLMs) have improved the generation of workflows from natural language instructions (aka NL2Workflow), yet existing single LLM agent-based methods face performance degradation on complex tasks due to the need for specialized knowledge and the strain of task-switching. To tackle these challenges, we propose WorkTeam, a multi-agent NL2Workflow framework comprising a supervisor, orchestrator, and filler agent, each with distinct roles that collaboratively enhance the conversion process. As there are currently no publicly available NL2Workflow benchmarks, we also introduce the HW-NL2Workflow dataset, which includes 3,695 real-world business samples for training and evaluation. Experimental results show that our approach significantly increases the success rate of workflow construction, providing a novel and effective solution for enterprise NL2Workflow services.

1 Introduction

Workflows, comprising reusable processes that integrate multiple tools or components in a specific logic sequence, can significantly enhance enterprise efficiency (Ayala and Bechard, 2024). Traditional workflow construction methods require numerous manual steps to orchestrate components, demanding specialized expertise (Chi et al., 1981, 2014; Faloughi et al., 2014). In contrast, automated commercial systems can directly convert natural language instructions into workflows, offering a more convenient and technically accessible approach.

With the rapid development of Large Language Models (LLMs) (Achiam et al., 2023; Dubey et al., 2024) and LLM agents (Xiong et al., 2024), researchers have begun to utilize them as backbones to develop Natural Language to Workflows (NL2Workflow) systems. Zeng et al. (2023) directly prompted a LLM to generate workflows, while Ayala and Bechard (2024) improved this process by adopting a Retrieval-Augmented Generation (RAG) approach to enhance the quality of the generated workflows. Although they can produce workflows for simple scenarios, a significant gap remains compared to human performance in handling complex real-world instructions.

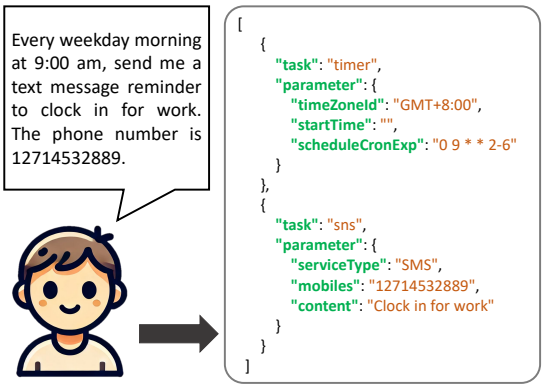

Figure 1: An example of generating workflows (JSON format) from text instruction.

Crafting a workflow (Figure 1) for real-life scenarios involves coordinating several tasks, from comprehending human intent, selecting appropriate components, to orchestrating the task flow and accurately configuring each component's parameters (Wang et al., 2024). It's quite challenging to rely on a single LLM agent to handle the entire process, as different tasks may require specialized knowledge and skills. The need to switch between multiple tasks could potentially affect its performance on any individual task (Gabriel, 2020).

To address this challenge, we draw inspiration from software development, where requires collaboration among multiple team members with diverse

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Industry Track), pages 20–35

skill sets is essential (Basili, 1989; Sawyer and Guinan, 1998). Specifically, we propose **WorkTeam**, a multi-agent framework that integrates multiple agents to collaboratively accomplish the NL2Workflow task. WorkTeam consists of three agents with distinct roles: the supervisor, the orchestrator and the filler (Figure 2). The supervisor agent is responsible for understanding the user's intent and coordinating the orchestrator agent and the filler agent. Upon receiving the user intent parsed by the supervisor agent, the orchestrator agent selects the appropriate components and arranges them into a suitable workflow schema. The filler agent then retrieves the documentation for relevant components and fills in accurate parameters, turning it into a fully operational workflow. Our framework enables different agents to perform their respective tasks accurately and communicate efficiently, thereby effectively constructing workflows. Moreover, since no publicly available NL2Workflow benchmarks exist, we construct the **HW-NL2Workflow** dataset from real production scenarios, comprising 3,695 entries for training and evaluation. Extensive experiments show that WorkTeam significantly improves workflow construction accuracy compared to existing methods, and further analysis validates the effectiveness of our framework.

Our contributions are summarized as follows:

- For the first time, we introduced a multi-agent framework into the NL2Workflow task, effectively enhancing the automation of workflow construction.
- We construct the HW-NL2Workflow dataset, comprising 3,695 entries of real-world enterprise business data for training and evaluation.
- Extensive experimental results on HW-NL2Workflow demonstrate the superior performance of our method and the effectiveness of each framework component.

2 Related Work

2.1 Natural Language to Workflow

Recent advancements in LLMs have enabled the conversion of natural language instructions into logical outputs, such as code (Xiong et al., 2023; Hong et al., 2024; Jiang et al., 2024) and SQL (Fu et al., 2023; Lian et al., 2024), making it increasingly viable for commercial applications. Workflows, which serve as a structured form of task orchestration, automate repetitive activities across various industrial applications, such as data entry and invoice processing (Villar and Khan, 2021). To reduce technical barriers and expand commercial adoption, researchers are now focusing on generating workflows directly from natural language instructions. For example, Microsoft (El Hattami and Pal, 2023) and ServiceNow (Gorroño et al., 2023) have patented systems that apply a machine learning model to transfer user-input text instructions into executable workflows. Zeng et al. (2023) developed FlowMind, a system that employs LLMs to automatically generate workflows from user queries, enhancing automation in financial services while maintaining data security. To improve the quality of generated workflows, Ayala and Bechard (2024) proposed an RAG-based method for NL2Workflow conversion. Upon receiving user instructions, their approach first retrieves relevant components and then generates workflows based on these components, effectively reducing hallucination issues. Although these methods have shown some success, single LLM-based approaches often suffer performance degradation in real-world commercial applications due to a lack of specialized knowledge and the strain of task-switching when handling complex instructions.

2.2 Multi-Agents

Recently, LLM agents have been developed to understand and execute complex instructions, leading to improved interaction and more informed decision-making across various environments (Xi et al., 2023; Ruan et al., 2023; Wu et al., 2024). Along this line, multi-agent systems enhance functionality by utilizing the collective intelligence and specialized skills of multiple LLM agents, assigning distinct roles and facilitating interactions to better simulate complex real-world scenarios.

Hong et al. (2024) introduced MetaGPT, a multi-agent collaborative framework for programming featuring six role-specific agents. This design, combined with Standardized Operating Procedures (SOPs), led to notable performance improvements in programming. In robotics, Kannan et al. (2023) proposed SMART-LLM, a multi-agent framework for robot task planning. SMART-LLM decomposes user instructions into sub-tasks, assigns them to robots based on their skills, and coordinates execution to optimize task completion. In scientific experimentation, Zheng et al. (2023) implemented a multi-agent framework with agents specializing in

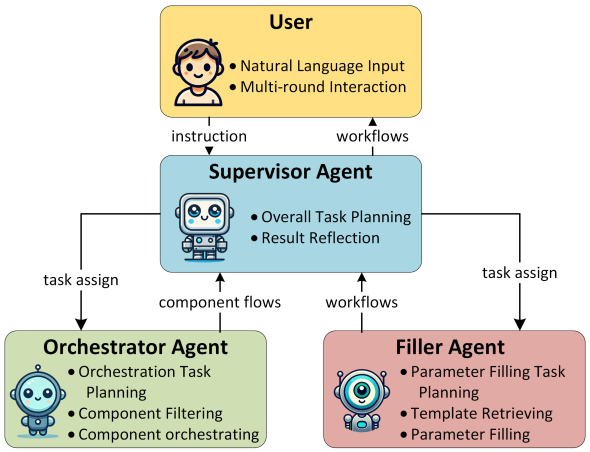

Figure 2: The overall architecture of the proposed WorkTeam framework.

areas like strategic planning, literature search, and coding. These agents collaborate with human researchers to improve the synthesis of complex materials. However, existing multi-agent approaches are generally designed for specific tasks and are not directly applicable to NL2Workflow.

In this paper, we propose a multi-agent approach to enhance NL2Workflow tasks, where agents with distinct roles and specialized skills collaborate to significantly boost workflow generation accuracy.

3 Methods

The WorkTeam framework comprises three agents: the supervisor agent, the orchestrator agent, and the filler agent. The overall structure of the framework is shown in Figure 2. Upon receiving an end user's prompt, the supervisor agent initiates a task planning phase, decomposing the tasks into subtasks and invoking the orchestrator and filler agents in a coordinated manner to execute them. The orchestrator and filler agents handle component orchestration and parameter filling, respectively, using appropriate tools to complete these tasks. To further elucidate the functionality of WorkTeam, Figure 6 in Appendix B provides an operational example. The design and functionality of these agents are detailed as following.

3.1 The Supervisor Agent

The supervisor agent, as depicted in Figure 2, is responsible for two primary functions: task planning and result reflection. The task planning function allows the supervisor agent to dynamically plan based on user instructions. For instance, when receiving a workflow creation instruction, the agent first calls the orchestrator agent for component orchestration, followed by the filler agent to populate the necessary parameters. In contrast, for workflow modification instructions, the agent may invoke only the orchestrator or the filler agent. This flexibility enables WorkTeam to efficiently execute user instructions. Upon completion of task planning, the supervisor agent assigns tasks to either the orchestrator agent or the filler agent based on the planning results, to ensure the objectives are achieved. After completing their tasks, the orchestrator and filler agents return the results to the supervisor agent for result reflection. The next steps proceed only if the supervisor agent confirms the results are correct. Otherwise, tasks are redirected to the appropriate agents for re-execution.

3.2 The Orchestrator Agent

The orchestrator agent selects appropriate components from the component set based on user instructions and arranges them in a logical order as implied by the instructions. To accomplish this, similar to the supervisor agent, the orchestrator agent first undertakes a dynamic planning process based on the input instructions inst_O, which encompass user directives and, if available, feedback from the supervisor agent. Subsequently, to ensure accurate orchestration results, the agent leverages two tools: the component filtering and the component orchestration tool, to finish the orchestration process based on the planning results. Next, we provide an overview of these two tools.

Component Filtering Tool The primary objective of the component filtering tool is to select candidate components from the component set that are most relevant to the orchestrator agent's input instructions. These selected components serve as input for subsequent orchestration. Specifically, we use the SentenceBERT model (Reimers and Gurevych, 2019) to extract embeddings for the orchestrator agent's input instructions inst_O and the descriptions desc_i for each component t_i, then compute the cosine similarity between the instruction and component embeddings to evaluate their relevance, as shown in Equations (1)

$$s_i = \text{Similarity}(\mathbf{e}_{inst}, \mathbf{e}^i_{desc}) \quad (1)$$

$\mathbf{e}_{inst}$ and $\mathbf{e}^i_{desc}$ represent their corresponding sentence embeddings for the input instructions and descriptions, Similarity is the cosine function, and

s_i is the similarity between $\mathbf{e}_{inst}$ and $\mathbf{e}^i_{desc}$. Components with higher similarity scores are considered more relevant to the input instructions and prioritized as candidate components. We select the $top\text{-}k$ components based on descending similarity scores:

$$C_{filtered} = \mathrm{TopK}(\langle t_1, s_1\rangle, \langle t_2, s_2\rangle, \ldots, \langle t_n, s_n\rangle) \tag{2}$$

Component Orchestration Tool The primary objective of the component orchestration tool is to select and arrange a subset of components from the candidate components provided by the component filtering tool, based on the logic embedded in the input from the orchestrator agent, thereby generating a component flow. Given that the orchestration logic is embedded within the natural language instructions provided by the user, this process demands a high level of text comprehension. To address this challenge, we employ a large language model (LLM) as the component orchestration tool. The LLM can directly generate a component flow that incorporates the specified orchestration logic based on inputs of the orchestrator agent. The arranged component flow can be represented by:

$$F_C = \mathrm{Tool}_O(\mathrm{inst}_O, C_{filtered}) \tag{3}$$

where Tool_O represents the component orchestration tool and F_C is the generated component flow.

3.3 The Filler Agent

The filler agent populates parameters for each component in the given component flow F_C, transforming it into a complete workflow. Generally, the input of the filler agent inst_P comprises three main parts: the user textual instructions, the component flow provided by the orchestrator agent, and the feedback from the supervisor agent, with the latter two being optional. Similar to the supervisor agent and the orchestrator agent, the filler agent performs dynamic task planning upon receiving input. It decomposes the parameter filling task and then utilizes the template lookup tool and the parameter filling tool to ensure the accuracy and stability of the parameterization results. A detailed introduction to these two tools will be provided next.

Template Lookup Tool The template lookup tool retrieves the parameter description d_i and the blank parameter template p_i associated with each component t_i in F_C. The parameter description provides detailed information for each parameter, including its meaning, type, and allowable values. In contrast, the blank parameter template encompasses all parameters of the component, assigning a default value to each. By utilizing the prepopulated blank parameter template, only essential modifications to the component's parameters are required, significantly reducing the complexity of the parameter filling task.

Parameter Filling Tool The parameter filling process begins once the tool has acquired three key elements: the orchestrated component flow F_C, the parameter description templates d_i and the blank parameter templates p_i for each component. With these in hand, the parameter filling tool's initial task is to analyze the input instructions, extracting all relevant information necessary for accurate parameter instructions. Then, it need to populate the specified parameters in the blank templates based on their intended meanings, resulting in a complete workflow. Due to the complexity of this task, in this paper, we employ a LLM as the backbone for parameter filling tool. By providing the LLM with the input instructions inst_P, component flow F_C, the looked-up parameter description templates $D = \{d_1, d_2, ..., d_m\}$, and the looked-up blank parameter templates $P = \{p_1, p_2, ..., p_m\}$ as prompts, the model is able to populate the parameters for each component in the stream, resulting in the generation of a complete workflow. The whole process can be represented by:

$$F_W = \mathrm{Tool}_P(\mathrm{inst}_F, F_C, D, P) \tag{4}$$

where Tool_P represents the parameter filling tool and F_W is the generated workflow.

4 HW-NL2Workflow

Given the limited availability of publicly accessible datasets for NL2Workflow tasks and our focus on real-world commercial applications, we have developed HW-NL2Workflow, a novel dataset specifically designed to meet these needs. This dataset consists of 3,695 real-world enterprise workflows, making it suitable for both performance evaluation and tool training.

4.1 Data Statistics

The HW-NL2Workflow dataset was created by collecting 3,695 workflows from our enterprise platform, each annotated by domain experts with natural language instructions. It is divided into training and testing sets, with detailed statistics provided in Table 1. Specifically, the dataset comprises 3,380

Split	Type	Size	# Comp	# Param
Train	Creation	2818	13993	45696
	Modification	562	2819	9187
	All	3380	16812	54883
Test	Creation	263	1269	4244
	Modification	52	252	838
	All	315	1521	5082

Table 1: Composition of HW-NL2Workflow. # Comp and # Param represent the number of components and parameters, respectively.

training samples and 315 testing samples. On average, each workflow in the training set consists of 5.02 components, with each component having 3.26 parameters. In the testing set, workflows contain an average of 4.83 components and 3.34 parameters per component. Additionally, the dataset encompasses both workflow creation and modification tasks, ensuring that WorkTeam can adapt to more flexible requirements.

4.2 Component Resources

In addition to data samples, the HW-NL2Workflow also provides comprehensive component resource information, including a component set C, a component parameter description set T_{desc}, and a blank parameter template set T_{blank}. These resource details provide sufficient component information to support workflow generation. Appendix A illustrates a few examples of the component resources of HW-NL2Workflow.

4.3 Metrics

We systematically evaluated the generated workflows from three perspectives:

Exact Match Rate (EMR) Exact matching occurs when the generated workflow fully aligns with the ground truth, including both component sequence and parameter values. The exact match rate is calculated as $E_{acc} = N_{em}/N_{total}$, where N_{em} and N_{total} represent the exact matches and total test samples, respectively.

Arrangement Accuracy (AA) Correct arrangement refers to the correctness of the sequence of components within the workflow generated by the model, irrespective of the correctness of the filled parameters. This metric primarily assesses the capability of the system to comprehend logical constructs in user instructions. Similarly, the arrangement accuracy is computed as $A_{acc} = N_{am}/N_{total}$, where N_{am} represents the number of samples with accurate arrangement.

Parameter Accuracy (PA) The parameter accuracy evaluates whether the parameters of the components in the generated workflow are consistent with those of the corresponding components in the ground truth. It is computed as $P_{acc} = N_{pm}/N_p$, where N_{pm} and N_p represent the number of matched parameters and the total number of parameters in the test set, respectively.

5 Experiments

5.1 Configurations

Model Configurations WorkTeam is a multi-agent framework that supports implementation with various models. This subsection only focuses on the model configurations used in our experiments. All agents in our experiments are built on Qwen2.5-72B-Instruct (Yang et al., 2024). The prompt for all these agents are illustrated in Figure 7 to Figure 9 in Appendix B. The component orchestration tool and the parameter filling tool are implemented with LLaMA3-8B-Instruct (Dubey et al., 2024), fine-tuned on the HW-NL2Workflow dataset. Similarly, the component filtering tool is built using the SentenceBERT model, which has been fine-tuned with data from the HW-NL2Workflow dataset.

Training Data Configurations The component filtering tool is built using the SentenceBERT model, trained with contrastive learning from paired text instructions and corresponding components. The training data is directly derived from the HW-NL2Workflow dataset, with positive samples comprising text instructions and their relevant components, and negative samples comprising text instructions with unrelated components.

In our experiments, both the component orchestration and parameter filling tools are developed by finetuning a LLM. The training data for the component orchestration tool includes the agent's input instruction, denoted as inst_O, along with descriptions of the selected $top\text{-}k$ candidate components. The model's output is a workflow that consists solely of the names of these components. For the parameter filling tool, the training data comprises the agent's input instruction inst_P, the component flow F_C, the corresponding component parameter descriptions D, and blank parameter templates P, with the model's output being a complete workflow.

Baselines Our experiments use a single LLM-based agent as the baseline, utilizing GPT-4o, Qwen2.5-72B-Instruct, Qwen2.5-7B-Instruct, and LLaMA3-8B-Instruct as backbone models. These models generate workflows directly based on the input use instructions and in-context examples. The prompts utilized for these approaches are detailed in Appendix C. We also incorporate a RAG NL2Workflow method from (Ayala and Bechard, 2024) as an additional baseline. Due to the unavailability of the original source code, we implement our version using SentenceBERT as the retriever and LLaMA3-8B-Instruct as the generator, both trained on HW-NL2Workflow.

5.2 Experiment Results

Methods	EMR (%)	AA (%)	PA (%)
GPT-4o	18.1	71.4	56.3
Qwen2.5-72B-Instruct	12.7	66.9	51.5
Qwen2.5-7B-Instruct	3.5	25.4	19.9
LLaMA3-8B-Instruct	1.6	19.4	16.6
RAG (Ayala and Bechard, 2024)	24.1	77.8	60.3
WorkTeam (ours)	**52.7**	**88.9**	**73.2**

Table 2: Comparison of experiment results of the baselines and our methods.

Table 2 presents the performance comparison between WorkTeam and baseline methods on the HW-NL2Workflow test set. In our experiments, the single LLM agent approach generates workflows end-to-end by directly inputting all component information and user instructions. The prompts for this method are shown in Figure 10. Table 2 shows that the NL2Workflow task is highly challenging for single LLM-based method. Top models like GPT-4o and Qwen2.5-72B-Instruct achieve only 18.1% and 12.7% EMR respectively, while smaller models such as Qwen2.5-7B-Instruct and LLaMA3-8B-Instruct are nearly ineffective, with EMRs of just 3.5% and 1.6%. The RAG NL2Workflow method improves workflow construction accuracy compared to the single LLM agent approach, but EMR performance remains unsatisfactory. In contrast, WorkTeam achieve an EMR of 52.7%, an AA of 88.9%, and a PA of 73.2% on the HW-NL2Workflow test set, representing a comprehensive and significant improvement over baseline methods.

We attribute the performance enhancement of WorkTeam to task specialization and collaboration among multiple agents. The orchestrator and filler agents concentrate on their specific tasks, improving execution stability and accuracy, while the supervisor agent, responsible for task planning and result reflection, enhances robustness and flexibility. Ablation studies, detailed in Table 3, further illustrate each agent's contribution.

Supervisor Agent	Orchestrator Agent	Filler Agent	EMR (%)	AA (%)	PA (%)
✓	✗	✗	-	-	-
✗	✓	✗	-	85.7	-
✗	✗	✓	-	-	-
✗	✓	✓	49.8	85.7	72.8
✓	✓	✓	**52.7**	**88.9**	**73.2**

Table 3: Results of the ablation experiments for different agents.'-' represents the task cannot be completed.

The results in Table 3 demonstrates that both the orchestrator agent and the filler agent are essential for workflow generation, as the absence of either leads to task failure. Although the workflow can still be generated without the supervisor agent, the accuracy decreases from 52.7% to 49.8% compared to the complete WorkTeam. This indicates that the task planning and result reflection functions of the supervisor effectively facilitates collaboration between the orchestrator and filler agents, thereby enhancing workflow generation accuracy.

To better illustrate the roles of WorkTeam's agents and its NL2Workflow process, we present a real-world case in Figure 11 of the Appendix D. Additionally, we developed a commercial NL2Workflow system based on WorkTeam that effectively meets business requirements, as shown in Figure 12 of the same appendix.

6 Conclusion

In this paper, we present WorkTeam, a novel multi-agent framework designated to enhance workflow automation in enterprise environments. Three specialized agents — supervisor, orchestrator, and filler agents — collaborate to overcome the limitations of a traditional LLM agent-based method, resulting in substantial improvements to workflow generation accuracy. Experimental results on the HW-NL2Workflow dataset confirm the effectiveness of WorkTeam. To address the lack of publicly available NL2Workflow benchmarks, we develop the HW-NL2Workflow dataset, comprising 3,695 real-world business samples, to support research in this area. Future work will focus on refining the framework to support more complex workflows and integrate it with a wider range of enterprise tools to further enhance automation.

References

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

Orlando Ayala and Patrice Bechard. 2024. Reducing hallucination in structured outputs via retrieval-augmented generation. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pages 228–238.

Victor R Basili. 1989. Software development: A paradigm for the future. In *[1989] Proceedings of the Thirteenth Annual International Computer Software & Applications Conference*, pages 471–485. IEEE.

Michelene TH Chi, Paul J Feltovich, and Robert Glaser. 1981. Categorization and representation of physics problems by experts and novices. *Cognitive science*, 5(2):121–152.

Michelene TH Chi, Robert Glaser, and Marshall J Farr. 2014. *The nature of expertise*. Psychology Press.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Amine El Hattami and Christopher Joseph Pal. 2023. Automatic flow implementation from text input. US Patent App. 17/752,564.

Mazen Faloughi, Wissam Bechara, Joy Chamoun, and Farook Hamzeh. 2014. Simplean: an effective tool for optimizing construction workflow. In *Proceedings for the 22nd Annual Conference of the International Group for Lean Construction*, pages 281–292.

Han Fu, Chang Liu, Bin Wu, Feifei Li, Jian Tan, and Jianling Sun. 2023. Catsql: Towards real world natural language to sql applications. *Proceedings of the VLDB Endowment*, 16(6):1534–1547.

Iason Gabriel. 2020. Artificial intelligence, values, and alignment. *Minds and machines*, 30(3):411–437.

José Luis Fernández Gorroño, Lan Li, Cédric Thierry Michel Bignon, Nicolas Chao Wei Ding, Cédric Bernard Jean Golmard, Anand Mourouguessin, Jaime Enrique Reyes Salazar, JAIN Shuktika, Dimitrios Leventis, Yu Hu, et al. 2023. Generating automations via natural language processing. US Patent App. 17/847,972.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Shyam Sundar Kannan, Vishnunandan LN Venkatesh, and Byung-Cheol Min. 2023. Smart-llm: Smart multi-agent robot task planning using large language models. *arXiv preprint arXiv:2309.10062*.

Jinqing Lian, Xinyi Liu, Yingxia Shao, Yang Dong, Ming Wang, Zhang Wei, Tianqi Wan, Ming Dong, and Hailin Yan. 2024. Chatbi: Towards natural language to complex business intelligence sql. *arXiv preprint arXiv:2405.00527*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3980–3990. Association for Computational Linguistics.

Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Xingyu Zeng, and Rui Zhao. 2023. Tptu: Task planning and tool usage of large language model-based ai agents. *arXiv preprint arXiv:2308.03427*.

Steve Sawyer and Patricia J. Guinan. 1998. Software development: Processes and performance. *IBM systems journal*, 37(4):552–569.

Alice Saldanha Villar and Nawaz Khan. 2021. Robotic process automation in banking industry: a case study on deutsche bank. *Journal of Banking and Financial Technology*, 5(1):71–86.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024. Agent workflow memory. *arXiv preprint arXiv:2409.07429*.

Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. 2024. Chateda: A large language model powered autonomous agent for eda. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.

Weimin Xiong, Yiwen Guo, and Hao Chen. 2023. The program testing ability of large language models for code. *arXiv preprint arXiv:2310.05727*.

Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024. Watch every step! llm agent learning via iterative step-level process refinement. *arXiv preprint arXiv:2406.11176*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Zhen Zeng, William Watson, Nicole Cho, Saba Rahimi, Shayleen Reynolds, Tucker Balch, and Manuela Veloso. 2023. Flowmind: automatic workflow generation with llms. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, pages 73–81.

Zhiling Zheng, Oufan Zhang, Ha L Nguyen, Nakul Rampal, Ali H Alawadhi, Zichao Rong, Teresa Head-Gordon, Christian Borgs, Jennifer T Chayes, and Omar M Yaghi. 2023. Chatgpt research group for optimizing the crystallinity of mofs and cofs. *ACS Central Science*, 9(11):2161–2170.

A Component Resource Examples

Figure 3 presents two component examples from the HW-NL2Workflow component set C. Each component includes a name and a functional description. When using the component filtering tool, the SentenceBERT model within the tool computes the similarity between the user input instructions and the description of each component. It selects the top-k components with the highest similarity as candidate components for use by the component orchestration tool.

Figure 4 illustrates two examples from the parameter description set of the HW-NL2Workflow, detailing all parameters required for each component, along with comprehensive descriptions of their functions. Figure 5 presents examples of the blank parameter template. When the parameter filling tool, invoked by the filler agent, is used, it receives the parameter description information of the component and the blank parameter template, subsequently filling in the parameters according to the template.

B Details of WorkTeam

Figure 6 illustrates a typical working process of WorkTeam. As previously mentioned, the supervisor agent acts as the primary agent, facilitating multi-turn interactions with the user and performing dynamic task planning. It invokes the orchestrator and filler agents to carry out component orchestration and parameter filling. Furthermore, the supervisor agent can evaluate the results provided by the orchestrator and filler agents. These capabilities contribute to the flexibility and stability of WorkTeam's operation.

Figure 7, 8, and 9 shows the prompts used in the supervisor agent, the orchestrator agent and the filler agent, respectively.

```
[
  {
    "task": "timer",
    "description": "Trigger component, timed trigger process."
  },
  ...
  {
    "task": "sns",
    "description": "Users can use this component to invoke
SNS services to send text messages, voice messages, or
verification codes."
  }
]
```

Figure 3: Examples in the component set C of HW-NL2Workflow.

```
[
  {
    "task": "timer",
    "parameter": [
      {
        "id": "scheduleCronExp",
        "description": "Trigger time, in cron format, used to
describe the frequency of the trigger"
      },
      {
        "id": "startTime",
        "description": "Start time"
      },
      {
        "id": "timeZoneId",
        "description": "Time zone identifier of the start time,
values range from GMT-12:00 to GMT+12:00;"
      }
    ]
  },
  ...
  {
    "task": "sns",
    "parameter": [
      {
        "id": "serviceType",
        "description": "Types of SNS notifications available,
only the following three options are available: SMS, Voice,
Captcha, representing text message, voice message, and
verification code respectively"
      },
      {
        "id": "mobiles",
        "description": "Mobile numbers to receive the
message"
      },
      {
        "id": "content",
        "description": "Content to be sent"
      }
    ]
  }
]
```

Figure 4: Examples in the component parameter description set T_{desc} of HW-NL2Workflow.

```
[
  {
    "task": "timer",
    "parameter": {
      "scheduleCronExp": "",
      "startTime": "",
      "timeZoneId": ""
    }
  },
  ...
  {
    "task": "sns",
    "parameter": {
      "serviceType": "",
      "mobiles": "",
      "content": ""
    }
  }
]
```

Figure 5: Examples in the blank parameter template set T_{blank} of HW-NL2Workflow.

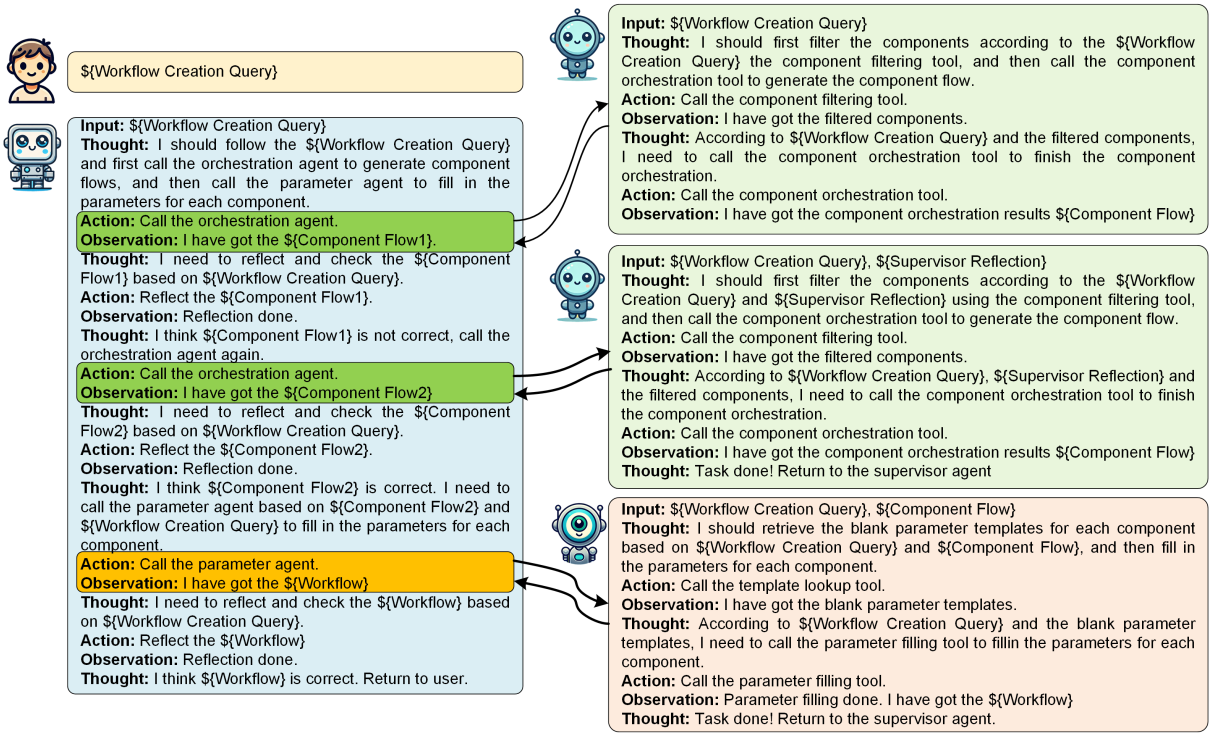

Figure 6: An illustration of a typical example for WorkTeam's overall working process.

C Details of the Baselines

For single LLM-based methods, we use prompts to guide the LLMs to generate workflows based on user instructions through in-context learning. The prompts utilized are illustrated in Figure 10.

Since the source code of the RAG method in (Ayala and Bechard, 2024) has not been released. We implemented our version. In our experiments, we trained a SentenceBERT model using contrastive learning with the training data in HW-NL2Workflow as the retriever. Actually, the retriever is same as the component filtering tool used in our orchestrator agent. For the generator, we fine-tuned a LLaMA3-8B-Instruct with the training data in HW-NL2Workflow. The generator aims to generate the workflow end-to-end according to the selected components by the retriever and the user instruction.

D Case Study and Enterprise System

Here, we provide a NL2Workflow case by the WorkTeam framework in Figure 11. Based on this case, we can see how the WorkTeam works for the NL2Workflow task. It can be seen that the supervisor agent can effectively plan the steps needed to complete the task, and accurately invoke the orchestrator agent and filler agent to complete orchestration and parameter filling tasks, and can reflect after receiving the return results from the orchestrator agent and filler agent. The orchestrator agent and filler agent can respectively plan for component orchestration and parameter filling tasks and call the corresponding tools to complete the tasks. Through the task decomposition and collaboration of multiple agents, WorkTeam can correctly and stably complete the NL2Workflow task.

Furthermore, the objective of developing WorkTeam is to provide more effective NL2Workflow services for enterprise business applications. Figure 12 presents the interface of the commercial NL2Workflow service system developed based on WorkTeam.

Prompt for Supervisor Agent

You are the supervisor agent in the NL2Workflow system, capable of directly interacting with users and automatically calling two agents based on user instructions: the orchestrator agent and the filler agent.

Your job is to receive messages from users:
1. First, you need to judge the user's instructions and plan tasks flexibly, for example:
(1) If the user's intention is to generate workflows from natural language, then first call the orchestrator agent to get the orchestration result, and then call the filler agent to get the final result, and return it to the user;
(2) If the user's intention is to modify the structure of the workflow, then you may need to call the orchestrator agent to make modifications to the workflow;
(3) If the user's intention is to modify the parameters in the workflow, then you may directly call the filler agent.
2. Determine if the results returned by the orchestrator agent/the filler agent have any issues. If there are problems with the results, you need to call the orchestrator agent/the filler agent again. (Please note that even after parameter filling, it is normal for some components to have no parameters or incomplete parameters, and there is no need to call again in such cases.)
3. Determine if the user instruction has been solved. If it has been solved, return the final result to the user.

Notice:
1. Do not create/modify workflows on your own; just call agents according to user intent.
2. Keep replies concise.

Your output should be in JSON: {"analysis" : xxxx, " action" : xxxx}
where the 'analysis' field is for your problem analysis process or reply to the user, and the 'action' field includes three actions: None (no call), <orchestrator_agent> (call the orchestrator agent), <filler_agent> (call the filler angent), <end> (end operation).
Note that you can only output a single such JSON content at a time, and it is not allowed to output multiple at once!

Figure 7: Prompt for the supervisor agent in WorkTeam. Notice that the initial prompt is in Chinese, we translate it to English for better reading in this paper.

Prompt for Orchestrator Agent

You are the orchestrator agent in the NL2Workflow system, and you can call two tools: the component filtering tool and the component orchestration tool.

You need to judge the user's instructions and plan tasks flexibly, for example:
1. If the user's intent is to generate a component flow based on their instructions, you should first call the component filtering tool to filter components from the component set, and then call the component orchestration tool to generate the component flow;
2. If the user's intent is to modify the component flow, you should first call the component filtering tool to filter out candidate components, and then use your own capabilities to modify the component flow provided by the user;
3. For other intents, respond according to your own capabilities.

Notice:
1. Do not orchestrate on your own ability! Determine when to call the component filtering tool and

the component orchestration tool and initiate the calls.
2. Keep replies concise.

Your output should be in JSON: {"analysis": xxxx, "action": xxxx}
where the 'analysis' field is for your problem analysis process or reply to the user, and the 'action' field includes four actions: None (no call), <call_selector>(call the component filter tool), <call_arrange>(call the component orchestration tool), , <end>(end operation).
Note that you can only output a single such JSON content at a time, and it is not allowed to output multiple at once!

Figure 8: Prompt for the orchestrator agent in WorkTeam. Notice that the initial prompt is in Chinese, we translate it to English for better reading in this paper.

Prompt for Filter Agent

You are the filler agent in the NL2Workflow system. Your role is to fill in parameters for each component in the component flows according to user instructions and the generated workflows. You can call two tools: the blank parameter template lookup tool and the parameter filling tool.

You need to judge the user's instructions and plan tasks flexibly, for example:
1. If the user's intent is to fill in parameters based on user instructions and the component flow, you need to first call the blank parameter template lookup tool to find the blank parameter templates corresponding to the components, and then call the parameter filling tool to fill in parameters for each component in the component flow.
2. If the user's intent is to modify the parameters in an existing workflow, you need to call the parameter filling tool to modify the parameters.
3. For other intents, respond according to your own capabilities.

Notice:

1. Do not fill the parameters on your own ability! Determine when to call the blank parameter template lookup tool and the parameter filling tool and initiate the calls.
2. Keep replies concise.

Your output should be in JSON: {"analysis": xxxx, "action": xxxx}
where the 'analysis' field is for your problem analysis process or reply to the user, and the 'action' field includes four actions: None (no call), <call_lookup>(call the blank parameter template lookup tool), <call_filling>(call the parameter filling tool), <end>(end operation).
Note that you can only output a single such JSON content at a time, and it is not allowed to output multiple at once!

Figure 9: Prompt for the filler agent in WorkTeam. Notice that the initial prompt is in Chinese, we translate it to English for better reading in this paper.

Prompt for Baseline Methods

You are a workflow generation expert. I will provide you with a textual instruction and descriptions of all candidate components, including their functionalities and detailed parameter information. Please select the appropriate components based on the instruction, arrange them according to the logical flow specified in the instruction, and finally populate the parameters of the selected components as indicated by the instruction.

Component Information:
{component_information}

Examples:
———
Instruction: {example_instruction1}
Output Workflow: {example_workflow1}
———

Instruction: {example_instruction2}
Output Workflow: {example_workflow2}
———
Instruction: {example_instruction3}
Output Workflow: {example_workflow3}

Now the input is:
Instruction: {input_instruction}
Output Workflow:

Figure 10: Prompt for baseline methods. {component_information} refers to the description of the component and its corresponding parameter descriptions. {example_instruction} and {example_workflow} are the given NL2Workflow examples for in-context-learning.

Case Study

User Instruction: Monitor the mailbox with account 98234 and password pass56789. When an email with the subject "Payment Confirmation" is received, automatically process the payment information using Python to obtain the result pythonRes and update the financial information through the post API at his.huawei.com/payment via the API gateway.

Supervisor Agent: {"analysis": "The user wants to set up a workflow to monitor emails for a specific subject, process payment information with Python, and then update financial information through an API. The first step will be to generate the workflow structure, followed by filling in the specific details.", "action": "<orchestrator_agent>"}

Orchestrator Agent: {"analysis": "The user wants to create a workflow where an email with a specific subject triggers a series of automated actions involving processing payments and updating financial information via an API. This requires identifying relevant components from the available set, and then arranging them into a coherent workflow.", "action": "<call_selector>"}

Candidate Components: ["public-email", "sns", "data-mapper", "selenium", "file-processing", "api-gateway", "http-request", "mqs-produce", "edm", "mqs-consumer"]

Orchestrator Agent: {"analysis": "Given the user's instruction and candidate components, I should arrange them into a component flow", "action": "<call_arrange>"}

Component Flow: [{"task": "public-email"}, {"task": "file-processing"}, {"task": "api-center"}]

Orchestrator Agent: {"analysis": "According to the user input and the component flow, I have finished the work.", "action": "<end>"}

Supervisor Agent: {"analysis": "I have received the component flow from the orchestrator agent. It seems rights. I should filling in the parameters.", "action": "<filler_agent>"}

Filler Agent: {"analysis": "The user wants to set up a workflow that monitors emails for a specific subject, processes the payment information using Python, and updates financial data via an API. First, I will call the blank parameter template lookup tool to get the required parameter templates for the 'public-email', 'file-processing', and 'api-center' components.", "action": "<call_lookup>"}

Blank Templates: [{"task": "public-email", "parameter": { "account": "", "password": "", "receiveType": "", "sender": "", "subject": ""}}, {"task": "file-processing", "parameter": {"inputParams": {}, "script": ""}}, {"task": "api-center", "parameter": {"url": "", "queryParams": {}, "headers": {}, "body": "", "method": ""}}]

Filler Agent: {"analysis": "Based on the user's instructions and the given component flow, I will now fill in the parameters using the provided blank templates.", "action": "<call_filling>"}

Parameter filling result: [{"task": "public-email", "parameter": {"account": "98234", "password": "pass56789", "receiveType": "", "sender": "", "subject": "Payment Confirmation"}}, {"task": "file-processing", "parameter": { "inputParams": {}, "script": ""}}, {"task": "api-gateway", "parameter": {"url": "his.huawei.com/payment", "method": "POST", "queryParams": {}, "body": "{"parameter": ${pythonRes}}"}}]

Filler Agent: {"analysis": "I have filled the parameters. My work is done.", "action": "<end>"}

Supervisor Agent: {"analysis": "I have received the workflow, and I think the result is correct. Return to the user.", "action": "<end>"}

Figure 11: A real case of the WorkTeam performing NL2Workflow task. The original text in the example is in Chinese, and we translate it into English for better readability.

Figure 12: The enterprise NL2Workflow system based on WorkTeam.