WORKFLOWLLM: ENHANCING WORKFLOW ORCHESTRATION CAPABILITY OF LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advancements in large language models (LLMs) have driven a revolutionary paradigm shift in process automation from Robotic Process Automation to Agentic Process Automation by automating the workflow orchestration procedure based on LLMs. However, existing LLMs (even the advanced OpenAI GPT-4o) are confined to achieving satisfactory capability in workflow orchestration. To address this limitation, we present WorkflowLLM, a data-centric framework elaborately designed to enhance the capability of LLMs in workflow orchestration. It first constructs a large-scale fine-tuning dataset WorkflowBench with $106,763$ samples, covering $1,503$ APIs from 83 applications across 28 categories. Specifically, the construction process can be divided into three phases: (1) Data Collection: we collect real-world workflow data from Apple Shortcuts and RoutineHub, transcribing them into Python-style code. We further equip them with generated hierarchical thought via ChatGPT. (2) Query Expansion: we prompt ChatGPT to generate more task queries to enrich the diversity and complexity of workflows. (3) Workflow Generation: we leverage an annotator model trained on collected data to generate workflows for synthesized queries. Finally, we merge the synthetic samples that pass quality confirmation with the collected samples to obtain the WorkflowBench. Based on WorkflowBench, we fine-tune Llama-3.1-8B to obtain WorkflowLlama. Our experiments show that WorkflowLlama demonstrates a strong capacity to orchestrate complex workflows, while also achieving notable generalization performance on previously unseen APIs. Additionally, WorkflowBench exhibits robust zero-shot generalization capabilities on an out-of-distribution task planning dataset, T-Eval.

1 INTRODUCTION

Process Automation (PA) (Cichocki et al., 1997), as a long-standing pursuit of the human race, aims to automate repetitive tasks to minimize human labor and improve efficiency. Tracing back to the agricultural era, humanity has employed waterwheels and oxen to automate farming practices. Robotic Process Automation (RPA), the current predominant PA technique, abstracts the repetitive task into a workflow (i.e., a program that can execute automatically) by orchestrating various actions (e.g., functions or APIs) (Ivančić et al., 2019; Hofmann et al., 2020; Wewerka & Reichert, 2020; Agostinelli et al., 2020; Ferreira et al., 2020). While RPA successfully reduces the human labor via automated workflow execution, the process of orchestrating workflows still requires substantial manual effort. Recently, large language models (LLMs) (OpenAI, 2022; 2023; Touvron et al., 2023a;b; Dubey et al., 2024) have achieved remarkable performance beyond natural language processing (Ahn et al., 2022; Cheng et al., 2023; Qian et al., 2024). The emergence of LLMs has unveiled a paradigm shift trend, moving from Robotic Process Automation to Agentic Process Automation (APA) (Ye et al., 2023; Zeng et al., 2023; Huang et al., 2024; Wornow et al., 2024; Li et al., 2024) which automates the workflow orchestration process by utilizing LLMs to build the workflow.

However, such a paradigm shift trend is constrained by **the limited ability of LLMs to orchestrate complex workflows**, which in turn leads to two crucial limitations in current APA methods: (1) **Constrained Action Scale**: Current LLMs can only orchestrate small-scale workflows with a limited number of actions. The most advanced OpenAI GPT-4 is capable of managing workflows with an average of only 6.1 actions, even when equipped with advanced decision-making mechanisms (Ye et al., 2023). This falls short of the complexity required to meet real-world demands.

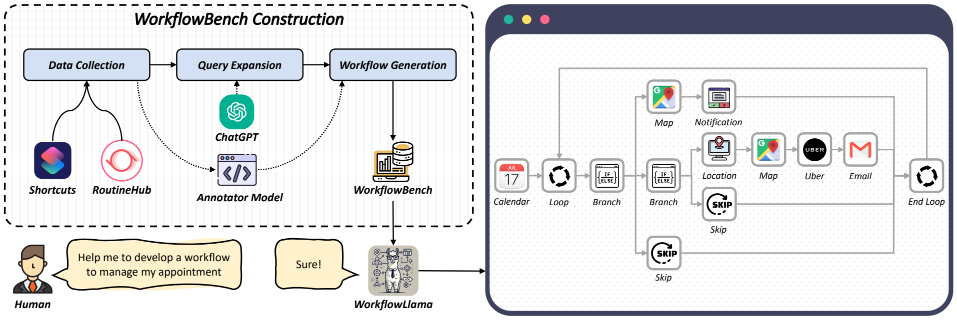


Figure 1: Overview of WorkflowLLM. It first constructs WorkflowBench through a three-phase pipeline and fine-tunes WorkflowLlama, which can generate workflows based on the user's query (appointment management in this case).

For instance, as a widely-used representative, Apple Shortcuts (Apple, 2024) involves an average of 70.4 actions. (2) **Simple Logical Structure**: Currently, most existing work mainly focuses on generating sequential actions (Yao et al., 2022; Qin et al., 2024; Chen et al., 2024) while workflows of the real-world applications usually involve intricate logical structures such as branches and loops. For example, Apple Shortcuts averages 2.6 nested branch/loop logical structures. As a result, **there is an urgent need to unlock the workflow orchestration capability of LLMs to expedite the paradigm shift in process automation.**

To address these challenges, we propose **WorkflowLLM**, a data-centric framework including dataset construction, model training, and evaluation to enhance LLMs' workflow orchestration capabilities (shown in Figure 1). Specifically, we first construct WorkflowBench, which consists of 106,763 supervised fine-tuning instances, encompassing 1,503 APIs across 83 applications, structured through three primary phases:

- **Data Collection**: We select shortcuts from RoutineHub as high-quality data sources because they represent a robust RPA application with numerous expert-developed workflows available. We curate 14,771 human-annotated, high-quality shortcuts spanning 28 diverse categories (e.g., Business, Health & Fitness, Productivity), alongside associated metadata including titles, functionality descriptions, and API documentations. As the raw workflow data is not directly suitable for LLMs to process, and considering that Python allows more convenient parameter passing and control logic (Ye et al., 2023; Wang et al., 2024b), we transcribe the shortcut source code into Python-like code. Subsequently, we prompt ChatGPT to generate comments, task plans, and task queries at varying levels of granularity—from fine-grained to coarse-grained—to enrich the data with detailed thought processes and enhance the learning efficacy of LLMs (Wei et al., 2023).
- **Query Expansion**: To enrich the diversity and complexity of workflows, we utilize ChatGPT to generate additional task queries. Specifically, we first sample applications with diverse functionalities and select their APIs, along with built-in APIs, to prompt ChatGPT to generate task queries that leverage these sampled APIs to accomplish specific tasks. To further ensure workflow complexity, we also sample real-world workflow examples as demonstrations to guide ChatGPT in generating similar workflows.
- **Workflow Generation**: As existing LLMs even GPT-4o still struggle in workflow generation, we first train a workflow annotator model based on the collected real-world shortcuts. Then we utilize the trained annotator to generate workflows for the expanded task queries. To prevent low-quality workflows generated by the annotator model from affecting subsequent training, we perform quality confirmation to ensure dataset integrity. We first utilize ChatGPT to refine the generated workflows to fix existing minor bugs in them and then use rule-based filtering to remove workflows with logical errors.

To evaluate the capability of LLMs in workflow orchestration, we employ two metrics: the reference-code-based metric **CodeBLEU** and the model-based metric **Pass Rate**. Experimental results demonstrate that WorkflowLlama consistently and significantly outperforms all baselines, including GPT-4o even with the in-context learning technique, across both metrics under unseen instructions and unseen APIs settings. Furthermore, WorkflowBench demonstrates strong generaliza-

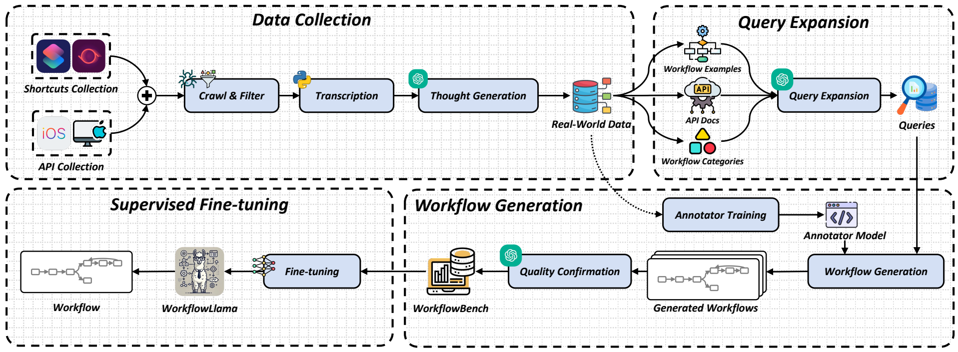


Figure 2: Illustration of our WorkflowLLM which contains three phases to construct WorkflowBench, followed by the supervised fine-tuning phase to derive WorkflowLlama.

tion capabilities in out-of-distribution (OOD) scenarios, particularly on the T-Eval benchmark (Chen et al., 2024), where it achieves an F1 plan score of **77.5**%.

2 Related Work

Process Automation RPA has gained considerable attention for automating repetitive tasks in various productivity scenarios (Ivančić et al., 2019; Hofmann et al., 2020; Wewerka & Reichert, 2020; Agostinelli et al., 2020; Ferreira et al., 2020). RPA predominantly relies on handcrafted workflows (e.g., programming, recording human behavior), making them highly suitable for automating well-structured, routine processes (Herm et al., 2020). However, such approaches require substantial efforts and in-depth domain expertise, resulting in high setup costs and limited adaptability. Recent advancements in LLMs have spurred interest in integrating these models into RPA to enhance flexibility and reduce dependency on manual workflow creation. Ye et al. (2023) introduced the concept of APA, which utilizes LLMs to autonomously orchestrate workflows based on human instructions. Subsequently, several studies have sought to apply APA in various domains, including travel planning (Xie et al., 2024), smartphone applications (Huang et al., 2024), enterprise automation (Wornow et al., 2024), financial question answering (Zeng et al., 2023), and data analysis (Li et al., 2024). Despite relying on advanced LLMs (e.g., GPT-4), these approaches have often exhibited suboptimal performance, highlighting challenges faced by existing LLMs in workflow orchestration. While Li et al. (2024) made an effort to fine-tune Mixtral-8$\times$7B (Jiang et al., 2024), it could only orchestrate sequential workflows with an average of 15.6 actions, remaining insufficient for real-world requirements. This work addresses a critical gap by proposing WorkflowLLM framework to enhance the workflow orchestration capabilities of LLMs to meet real-world demands.

Tool Learning Workflow orchestration driven by LLMs frequently depends on external tools, such as APIs, to extend their operational capabilities. Recent studies have demonstrated that LLMs can effectively acquire and utilize external tools by learning from their documentation, thereby solving complex tasks that would otherwise be beyond the model's native capabilities (Wu et al., 2023; Schick et al., 2024; Qin et al., 2023b; 2024). This integration enables LLMs to access real-time knowledge and perform specialized operations, particularly for executing intricate processes (Yang et al., 2023; Nakano et al., 2021; Qin et al., 2023a; Wang et al., 2024c; Gao et al., 2023). To further enhance this capability, several efforts have introduced datasets specifically designed to fine-tune LLMs for tool interactions (Zhuang et al., 2024; Qin et al., 2024; Wang et al., 2024a). However, these datasets are often constrained to limited actions scale, thus limiting their effectiveness for managing complex, real-world workflows. Compared to tool learning scenarios, orchestrating workflows demands more sophisticated planning and reasoning that current LLMs have yet to fully realize. In response to these limitations, we present WorkflowLLM to significantly improve LLMs' capabilities in workflow orchestration. Besides, Shen et al. (2024) also used Apple's Shortcuts but aimed to assess LLMs' tool utilization ability. In contrast, we emphasize a different scenario, workflow orchestration and aim to enhance the workflow orchestration ability rather than evaluation alone.

3 WorkflowLLM

As Figure 2 shows, WorkflowLLM introduces a data-centric framework to enhance the capability of LLMs in workflow orchestration by constructing a high-quality supervised fine-tuning dataset WorkflowBench. In this section, we outline the dataset construction process, which is carried out in three distinct phases: Data Collection, Query Expansion, and Workflow Generation.

3.1 Data Collection

We first give the introduction to Apple Shortcuts and RoutineHub, and describe how we crawl and filter to get high-quality data. We then convert the shortcuts into Python-style workflow code. Inspired by Chain-of-Thought (Wei et al., 2022; Chen et al., 2023), we prompt ChatGPT to generate hierarchical thoughts, including comments, task plans, and task queries, progressing from fine-grained to coarse-grained details for each shortcut.

Apple Shortcuts and RoutineHub Apple Shortcuts, as a representative application of RPA, is developed by Apple Inc. This tool facilitates the automation of a series of actions, enabling users to efficiently perform a diverse range of tasks. The actions within Shortcuts are APIs provided by both built-in Apple applications, such as *Safari*, and third-party applications like *OpenAI*. Each application may provide multiple actions. For instance, *OpenAI* provides APIs that facilitate voice conversations and text interactions with ChatGPT. Through a simple drag-and-drop interface, users can construct complex workflows, such as navigating to the nearest coffee shop or downloading watermark-free images from TikTok.

RoutineHub[1] is a prominent community for sharing shortcuts, with a collection of thousands of shortcuts across both iOS and macOS platforms. All shortcuts on RoutineHub are categorized into 28 workflow categories (e.g., Business, Health & Fitness, Productivity, etc). RoutineHub records the metadata of each shortcut (e.g., title, description, iCloud URL), providing valuable information.

Crawling and Filtering For each shortcut, we crawl the title, developer-provided description, and iCloud URL linked to Apple. As RoutineHub does not provide the source code for these shortcuts, we further crawl it from their iCloud URLs. Besides, we merge shortcuts collected by Shortcuts-Bench (Shen et al., 2024), sourced from platforms like ShareShortcuts[2] and MacStories[3], to further expand the scale of our dataset. However, the source code of these shortcuts lacks detailed information about the involved actions, such as API metadata. Inspired by ShortcutsBench (Shen et al., 2024), we extract action information from macOS's built-in definition files and third-party application interface definition files. For each API, we record its name, description, parameter names, parameter types, default values, return value types, and return value name, which provides a valuable resource for LLMs to efficiently interpret and utilize these APIs, even in zero-shot scenarios.

To ensure compatibility between the crawled shortcuts and the action interfaces, we implement a stringent filtering mechanism to verify that all API calls are executed correctly. During this process, we identify that some shortcuts contain non-interpretable binary sequences as API parameters, potentially disrupting the training process of language models. To maintain data quality, we remove these samples from the dataset. As a result, we curate a final set of $14,771$ high-quality shortcuts, ensuring the reliability of the dataset for subsequent data expansion and model training.

Shortcuts Transcription The original shortcut source codes are written in property lists format (Hummert & Humphries, 2022), which sequentially encodes logical constructs like branches and loops. This encoding is notably different from the types of data commonly used in the pre-training of LLMs. To address this gap, we convert the shortcuts into abstract syntax trees (ASTs), apply pre-order traversal to transform them into Python code, with further algorithmic details provided in Appendix A. Furthermore, the original shortcuts use hexadecimal strings as variable names, leading to reduced semantic clarity. To improve interpretability, we use ChatGPT to automatically reassign these variables with more contextually meaningful names, thereby enhancing the overall

[1]https://routinehub.co/

[2]https://shareshortcuts.com

[3]https://www.macstories.net/shortcuts

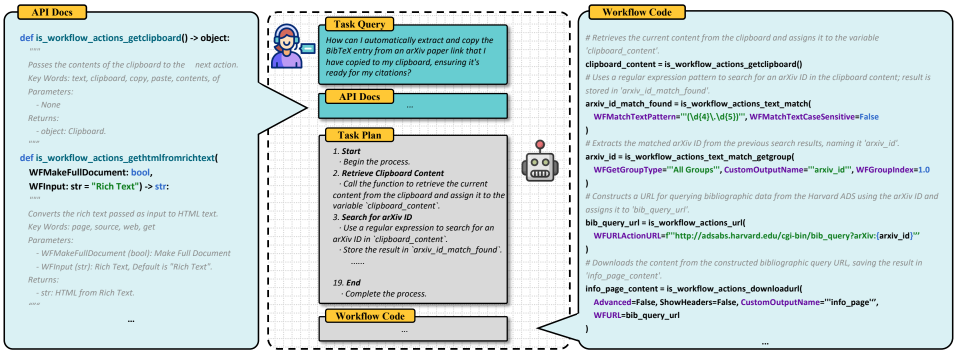


Figure 3: Illustration of data field composition in WorkflowBench comprising *Task Query*, *API documentations*, *Task Plan*, and *Workflow code* with *Comments*.

readability and utility of the code for further language model training. A typical comparison between property lists and Python code can be found in Appendix C.

Thought Generation To provide informative guidance for LLMs in orchestrating workflows, we design a three-level thought hierarchy from fine-grained to coarse-grained: (1) **Low-level comments** are intended to clarify the purpose of each action within the workflow. (2) **Median-level plans** represent an abstraction over a sequence of actions, outlining the collective goal of these steps. (3) **High-level queries** reflect the user's requirements, specifying the intended outcome without prescribing specific methods to achieve it. These three levels of thought are generated through a bottom-up approach. Specifically, given the transcribed workflow w, let the set of actions in the workflow w be denoted as $\mathcal{A}$, where each action $a_i \in \mathcal{A}$ corresponds to a function calling in the Python code. For each action a_i, we generate a corresponding comment c_i by prompting ChatGPT. Subsequently, given the action set $\mathcal{A} = \{a_i\}$ and comments $\mathcal{C} = \{c_i\}$ of workflow w, we prompt ChatGPT to generate the corresponding task plan $\mathcal{P}$. We combine the task plan $\mathcal{P}$, the comments $\mathcal{C}$, and the action set $\mathcal{A}$ of the workflow w to generate the high-level task query $\mathcal{Q}$. This bottom-up manner is analogous to the summarization task, effectively ensuring content reliability and minimizing the risk of hallucination.

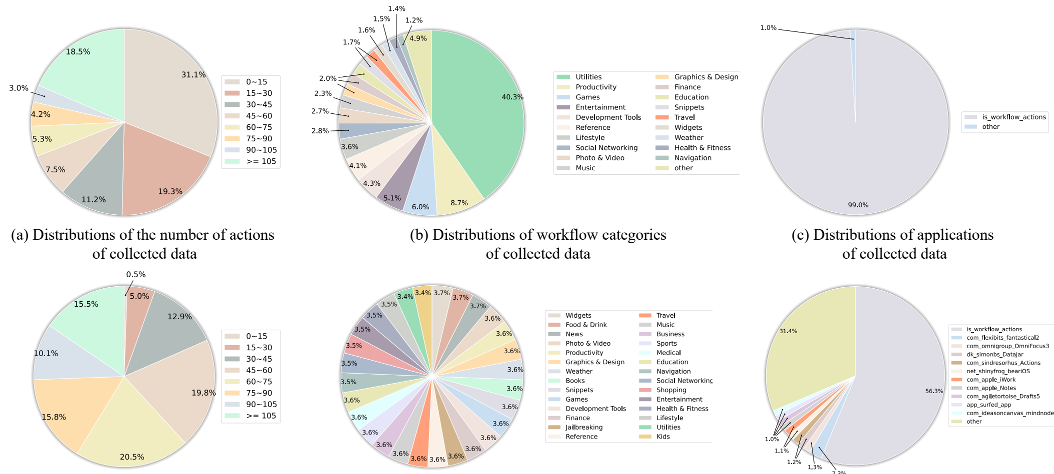
Finally, as Figure 3 shows, each workflow w is represented as: $w = \{\mathcal{Q}, \mathcal{D}, \mathcal{P}, \mathcal{A}\}$, where the workflow w consists of the task query $\mathcal{Q}$, action documentation $\mathcal{D}$ for all involved actions, the task plan $\mathcal{P}$, and all actions represented as annotated Python code $\mathcal{A}$. An example from WorkflowBench can be found in Appendix D.

3.2 Query Expansion

After performing a comprehensive statistical analysis on the collected data, we find that the data exhibits significant complexity, with an average of 70.4 actions and 12 branches, surpassing the complexity of existing workflow-related benchmarks. However, the diversity of the data is relatively low. Specifically, 40.3% of the workflows fall under the `Utilities` category, and over 99% of the APIs used are Apple's built-in APIs (i.e., those classified as `is_workflow_actions` APP).

Therefore, we intend to expand the dataset by focusing on two key aspects: (1) **Diversity**: making up for the lack of diversity in real data and covering a broad range of APIs and workflow categories to enhance the model's utility and robustness; (2) **Complexity**: matching the action scale and logical complexity of the real-world data to ensure that they can effectively represent real-world problems and orchestrate nodes accordingly. To this end, we sample APIs from diverse applications and multiple workflows with representative logical structures (e.g., whether they contain branches or loops) to synthesize additional queries.

To ensure that the number of APIs in the synthesized dataset aligns with real-world usage, we sample n APIs based on real-world distributions. Approximately $\lfloor n/2 \rfloor$ are drawn from Apple's built-in API set (e.g., *openurl* or *sendemail*), with the remainder from third-party applications (e.g., *OpenAI*). The total number of built-in and external APIs is thus n.



(a) Distributions of the number of actions of collected data

(b) Distributions of workflow categories of collected data

(c) Distributions of applications of collected data

(d) Distributions of the number of actions of final data

(e) Distributions of workflow categories of final data

(f) Distributions of applications of final data

Figure 4: Comparison of the distributions across workflow categories, APPs, and action counts in the collected data and the final dataset. The upper section shows the original data collected from Apple Shortcuts and RoutineHub, while the lower section presents the expanded dataset distributions.

To ensure that the sampled APIs can interact coherently, we do not sample directly from the entire API set. Instead, we first randomly select 1-5 applications and then choose all APIs from these selected applications. This method ensures that the selected APIs are functionally compatible and capable of representing real-world workflows.

The prompt used for ChatGPT to synthesize queries consists of four components: (1) a general prompt to describe the task query generation task, (2) documentations for the sampled APIs, (3) in-context examples from the collected data for reference, and (4) the workflow category to which the query belongs to. By controlling the workflow category and in-context examples, we can ensure the diversity and complexity of the generated data. As seen from Figure 4, the synthesized query has a more balanced category distribution and uses more third-party APIs. Although most of the APIs used are still built-in APIs, this is reasonable considering that they carry necessary operations.

3.3 Workflow Generation

To annotate the corresponding workflows of the synthesized queries effectively, we train an annotator model based on the collected shortcuts data to support more diverse applications and categories, while ensuring consistency with the real-world data as much as possible.

Table 1: Detailed statistics of WorkflowBench. *Seed.* refers to the collected data from Shortcuts. *Train.* and *Test.* refers to the training set and the test set of WorkflowBench respectively.

Statistics	Seed.	Train.	Test.
Num. of Instances	14,771	105,573	1,190
Num. of APPs	71	83	31
Num. of APIs	584	1,503	324
Num. of Categories	28	28	28
Avg. Action	70.4	78.5	41.7
Avg. IF	12.0	7.4	7.9
Avg. LOOP	0.7	0.5	0.5
Avg. Nested Depth	2.6	2.7	2.1

Annotator Training First, we construct the supervised fine-tuning (SFT) dataset based on the collected human-labeled shortcuts. Specifically, each workflow data point comprises a query $\mathcal{Q}$, the corresponding action documentation $\mathcal{D}$, the task plan $\mathcal{P}$, and the workflow represented as annotated Python code $\mathcal{A}_{\text{commented}}$. During the SFT process, as shown in Figure 3, we take the query $\mathcal{Q}$, the corresponding action documentation $\mathcal{D}$ as the input to guide the model to generate a task plan $\mathcal{P}$, followed by the step-by-step generation of the current thought (i.e., the comment c_i) and the corresponding action a_i, which includes the action name and its associated parameters. We use the trained annotator to generate workflows $\mathcal{A}'$ from synthesized queries.

Quality Confirmation Due to the limited accuracy of the annotator model, the generated workflows may contain errors to some extent. For example, we identify issues in $\mathcal{A}'$ (e.g., extraneous branches not relevant to the query and incorrect function call formats). To enhance the overall quality, we prompt ChatGPT with in-context samples to refine both $\mathcal{A}'_{\text{commented}}$ and $\mathcal{P}'$, ensuring that the workflow accurately addresses the query. Then, we use rule-based filtering to remove workflows with fundamental errors. Specifically, we remove samples that don't incorporate code, don't utilize the given APIs, or violate parameter constraints associated with those APIs.

Finally, we derive a synthesized dataset of $91,992$ instances, which is combined with the initially collected data to form the final WorkflowBench. It contains $106,763$ instances with $1,503$ APIs across 83 applications, which are used to train WorkflowLlama. The statistics of WorkflowBench are listed in Table 1 and the distribution comparisons of workflow categories, APPs, and the number of actions between the collected data and final data are demonstrated in Figure 4. From the statistical results, we can see that the synthetic data maintains complexity while expanding diversity.

4 Experiments

4.1 Experimental Setup

Training Details We fine-tune the annotator and WorkflowLlama on LLaMA-3.1-8B (Dubey et al., 2024) for 3 epochs using the AdamW optimizer (Loshchilov & Hutter, 2019). A linear learning rate scheduler is used with a peak learning rate of 2×10^{-5} and a warm-up ratio of 0.1. Each mini-batch contains 32 examples, and the maximum sequence length is set as $8,192$ tokens.

Baselines To provide a comprehensive comparison, we select several representative LLMs as baselines for our experiments. These baselines include proprietary models such as GPT-4o-mini and GPT-4o, as well as open-source models like Qwen2-7B (qwe, 2024), Llama-3.1-8B, and Llama-3.1-70B (Dubey et al., 2024). Additionally, we apply in-context learning (ICL) (Dong et al., 2022) with one random-sampled instance to these baselines to better adapt them for workflow orchestration.

Metrics In the main experiments, we use both reference-code-based metrics and a model-based evaluation to comprehensively evaluate the quality of the generated workflows. For reference-based metrics, we apply **CodeBLEU** (Ren et al., 2020) with four components:

- **BLEU** measures N-gram overlap for token-level similarity.
- **Weighted N-Gram Match** assigns higher weights to critical code tokens like keywords.
- **Syntactic AST Match** compares the Abstract Syntax Trees (ASTs) to assess syntactic accuracy.
- **Semantic Data-Flow Match** evaluates logical correctness by comparing data-flow relationships between variables.

Together, these components provide a comprehensive evaluation of both syntactic and semantic aspects of the workflows. We follow Ren et al. (2020), setting the four components to 0.1, 0.1, 0.4, and 0.4, respectively, and calculate a weighted sum to obtain the CodeBLEU score. For model-based evaluation, we elaborately prompt ChatGPT as the automatic evaluator to evaluate the **Pass Rate** of the generated workflows.

4.2 Effectiveness of Evaluator

To validate the reliability of the ChatGPT evaluator in terms of Pass Rate, we sample 30 instruction-response pairs (i.e., task queries and their corresponding workflow codes) for each model in Table 2, forming a human-evaluated dataset of 330 instances ($30 \times 11 = 330$). First, we use GPT-4o-mini to label whether each instance could complete the given tasks only using the provided APIs. Then, human evaluators re-label the sampled data according to the same criteria. Ultimately, 268 instances are labeled consistently by both the ChatGPT evaluator and human evaluators, achieving an agreement rate of **81.2**%, demonstrating the reliability and effectiveness of the evaluator.

4.3 Main Experiments

Settings The main experiments are conducted using the test set of WorkflowBench. Ideally, by scaling both the quantity and diversity of instructions and unique tools within the training data,

Table 2: Performance comparison of various models on the test set of WorkflowBench under the **unseen instructions (ID)** and **unseen APIs (OOD)** settings (%).

Model	CodeBLEU										Pass Rate	
	BLEU		Weighted N-Gram		AST		Data-Flow		Overall			
	ID	OOD	ID	OOD	ID	OOD	ID	OOD	ID	OOD	ID	OOD
Proprietary Models												
GPT-4o-mini	0.4	0.4	1.5	1.6	29.5	29.5	37.0	36.3	26.8	26.5	54.8	47.5
w/ ICL	0.5	0.5	1.7	1.8	35.3	34.4	35.1	34.2	28.3	27.7	66.0	57.7
GPT-4o	0.5	0.4	1.8	1.7	33.5	31.8	37.3	36.9	28.5	27.7	56.6	47.5
w/ ICL	0.5	0.5	1.8	1.8	37.1	35.3	38.0	36.6	30.2	30.0	67.5	57.6
Open-Source Models												
Qwen2-7B	0.4	0.4	1.2	1.3	27.2	27.7	33.2	33.1	24.4	24.5	25.6	22.6
w/ ICL	0.5	0.5	1.2	1.3	30.2	29.8	32.4	32.9	25.2	25.3	28.2	26.4
Llama-3.1-8B	0.6	0.7	1.2	1.4	31.0	29.6	30.0	30.8	24.6	24.3	33.0	24.5
w/ ICL	0.7	0.7	1.3	1.4	34.0	32.4	32.6	32.4	25.3	25.2	40.2	32.7
Llama-3.1-70B	0.4	0.4	1.4	1.5	29.9	30.0	37.8	37.6	27.3	27.2	55.4	42.3
w/ ICL	0.4	0.4	1.6	1.5	34.1	32.9	**39.1**	**38.4**	29.5	28.7	67.6	61.4
WorkflowLlama (8B)	**9.4**	**7.0**	**11.09**	**8.3**	**55.1**	**48.8**	38.0	35.3	**39.3**	**35.1**	**76.9**	**70.4**

WorkflowLlama is expected to generalize to novel instructions and APIs that are not seen during training. This is particularly important because it enables users to define custom APIs and allows WorkflowLlama to adapt based solely on the provided documentation. To evaluate this capability, we assess WorkflowLlama's generalization performance at two levels: (1) **Unseen Instructions**, considers an **In-Distribution (ID)** setting, which involves using the same set of APIs as those in the training data, and (2) **Unseen APIs**, considers an **Out-Of-Distribution (OOD)** setting, involving only 50 common APIs required to construct workflows and APIs that are absent from the training data. Since WorkflowBench contains a comprehensive set of APIs, which poses a substantial challenge for LLMs in terms of API comprehension and selection, we provide the correct APIs directly as input. It allows us to focus on the workflow orchestration, bypassing the issue of API selection.

Main Results The results are placed in Table 2, from which we derive that:

1. Although multiple workflows can successfully complete a query, there is a positive correlation between the reference-free Pass Rate metric and the reference-based CodeBLEU metric. Given that the Pass Rate metric derived from ChatGPT aligns with human evaluations over 80% of the time, CodeBLEU serves as a reliable proxy for evaluating workflow orchestration capabilities.
2. All models demonstrate a certain capacity for workflow orchestration. This may stem from their inherent instruction-following and code-generation capabilities. We find that models like GPT-4o and Llama-3.1-70B, which perform better on generic tasks, also excel in workflow orchestration. In addition, prompting with in-context samples significantly enhances the models' performance.
3. We find that scores on text overlap metrics such as BLEU and weighted N-gram are low for all models. Even the fine-tuned WorkflowLlama only achieves 8.2% and 9.7% on these two metrics. This is because the reference codes consist mainly of workflows with function names and arguments, and contain few Python-related keywords, making exact matching challenging. In contrast, models achieve better scores on syntactic AST match and semantic data-flow match.
4. After fine-tuning, WorkflowLlama shows a significant improvement in its ability to orchestrate actions. The performance of WorkflowLlama even outperforms powerful closed-source models GPT-4o with ICL by a large margin. Specifically, WorkflowLlama achieves a **39.3**% score on CodeBLEU and a **76.9**% Pass Rate under ID settings, demonstrating the validity of our proposed WorkflowLLM framework and WorkflowBench dataset.
5. WorkflowLlama demonstrates strong generalization capabilities. Even though it has not been trained on the same instructions or APIs, it still significantly outperforms the vanilla Llama-3.1 on all metrics, ahead of or close to the more powerful foundation models. Notably, our method achieves **35.1**% in CodeBLEU and **70.4**% in Pass Rate, outperforming all strong baselines.

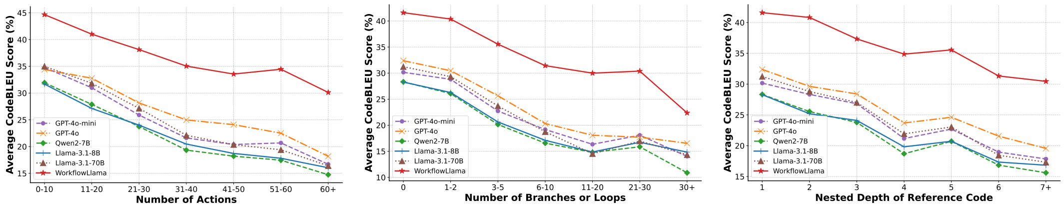


Figure 5: Performance comparisons based on the number of actions, the number of Branch & Loop, and the nested depth of the reference code.

4.4 Analysis of Workflow Complexity

To evaluate the models' ability to generate workflows of varying complexity, we break down the performance of CodeBLEU according to the total number of actions, the number of branches and loops, and the nested depth of the reference code. As shown in Figure 5, the performance of all models deteriorates as the number of actions or the logical complexity increases, indicating the challenge of orchestrating complex workflows. However, across all levels of complexity, WorkflowLlama significantly outperforms all other models. Moreover, the relative performance of WorkflowLlama improves as the complexity of the workflow increases, which demonstrates fine-tuning with WorkflowBench significantly enhances the model's ability to handle more complex workflows.

4.5 Out-of-distribution Generalization to T-Eval (Chen et al., 2024)

Settings To further evaluate the generalization capability of WorkflowLlama, we conduct experiments using an OOD benchmark, T-Eval, a widely-used benchmark to evaluate the multi-step decision-making capability of LLMs to utilize APIs. The original data format in T-Eval is based on JSON or strings, which differ significantly from the Python-based format employed in WorkflowBench. To ensure the evaluation metrics' consistency between ours and the original paper, we convert WorkflowBench into JSON format while preserving the metadata of workflows and the specifics of queries. Subsequently, we retrain WorkflowLlama on the transformed dataset. We employ the **F1 Score** proposed in the original paper to measure the alignment with the reference API sequences.

Results The results are shown in Table 3. As observed, WorkflowLlama demonstrates strong OOD generalization performance on the T-Eval benchmark, despite being trained on different domains and tasks using different APIs. Notably, WorkflowLlama significantly outperforms the vanilla Llama3.1-8B as well as larger open-source models like Llama-2-70B and Qwen-72B, highlighting that fine-tuning with WorkflowBench enhances the model's out-of-distribution planning ability.

Table 3: Comparisons of F1 scores on the **PLAN** task of T-Eval. (**Bold** denotes the best score among models of the same category.)

Model	F1
Proprietary Models	
Claude2	84.9
GPT-3.5	86.6
GPT-4	**86.7**
Open-Source Models	
Qwen-7B	63.1
Mistral-7B	64.9
Llama-3.1-8B	68.2
Qwen-14B	69.7
Llama-2-13B	65.1
Vicuna-13B	54.0
Baichuan2-13B	52.1
WizardLM-70B	42.7
Llama-2-70B	63.1
Qwen-72B	73.4
WorkflowLlama (8B)	**77.5**

4.6 Ablation Study

Settings To assess the efficacy of WorflowBench's components, we conduct an ablation study under the settings of unseen instructions (i.e., the ID setting).

Table 4: Ablation study results of Natural Language Thoughts on Workflow Orchestration (%).

Model	CodeBLEU				
	BLEU	**Weighted N-Gram**	**AST**	**Data-Flow**	**Overall**
WorkflowLlama	9.4	11.1	55.1	38.0	39.3
w/o Task Plan	9.1	10.7	53.9	36.6	38.2
w/o Comment	9.1	10.8	54.9	35.3	38.1
w/o Task Plan & Comment	8.8	10.2	53.7	35.1	37.4
w/o Synthetic Data	7.8	9.4	53.5	35.4	37.3

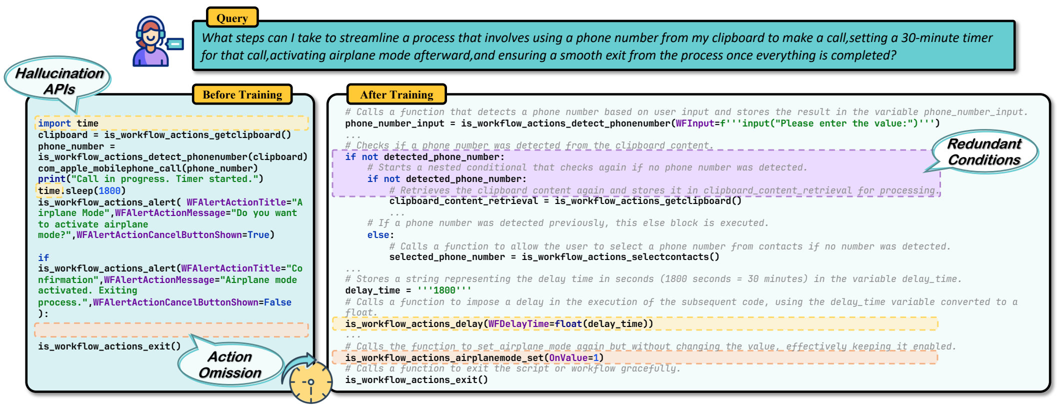


Figure 6: Case study of generated code between vanilla Llama-3.1-8B and WorkflowLlama.

Results Table 4 presents the performance results when the model is trained under different conditions: without synthetic data, without the task plan $\mathcal{P}$, without action-level comments $\mathcal{C}$, and without both $\mathcal{C}$ and $\mathcal{P}$. The experimental results reveal two key findings. **First**, the two types of natural language thoughts enhance the reasoning capabilities of the model. Removing either type of thought leads to a decline in CodeBLEU performance. **Second**, training on large-scale synthetic data further improves performance, highlighting the effectiveness of the WorkflowBench expansion process.

4.7 Case Study

To further illustrate the effect of fine-tuning on WorkflowBench, we present a typical example in Figure 6. In this case, the vanilla Llama-3.1 model exhibits two types of errors. **First**, the model does not adhere to the given instructions for workflow orchestration, using APIs outside the provided list, i.e., hallucination APIs. Specifically, it uses the `time.sleep()` function instead of `is_workflow_actions_delay()` to set a timer. **Second**, due to its relatively weak workflow orchestration capabilities, the model fails to complete all user instructions. Specifically, it does not activate airplane mode using the `is_workflow_actions_airplanemode_set()` function. Fine-tuning on WorkflowBench effectively alleviates these two issues. However, we observe that fine-tuning also introduces redundant actions. For instance, WorkflowLlama repeats the parsing check of the clipboard's content. We will address this redundancy problem in future work.

5 Conclusion

In this paper, we present WorkflowLLM to enhance the capability of large language models in workflow orchestration. In WorkflowLLM, WorkflowBench is constructed covering $106,763$ workflows with $1,503$ APIs across 83 applications through a three-phase pipeline. By fine-tuning Llama-3.1-8B on WorkflowBench, we derive WorkflowLlama which can achieve superior performance on the workflow orchestration task exceeding all comparable baselines including the most advanced OpenAI GPT-4o. Moreover, we adapt our WorkflowLlama on the T-Eval dataset and the experimental results reveal the generalization ability of our constructed WorkflowBench. We believe that our constructed dataset has the potential to contribute to advancements in APA.

REFERENCES

Qwen2 technical report. 2024.

Simone Agostinelli, Andrea Marrella, and Massimo Mecella. Towards intelligent robotic process automation for bpmers. *arXiv preprint arXiv:2001.00804*, 2020.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Apple. Shortcuts app, 2024. URL `https://apps.apple.com/us/app/shortcuts/id915249334`. Accessed: 2024-05-09.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL `https://openreview.net/forum?id=YfZ4ZPt8zd`.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, et al. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9510–9529, 2024.

Liying Cheng, Xingxuan Li, and Lidong Bing. Is gpt-4 a good data analyst? *arXiv preprint arXiv:2305.15038*, 2023.

Andrzej Cichocki, Helal A Ansari, Marek Rusinkiewicz, and Darrell Woelk. *Workflow and process automation: concepts and technology*, volume 432. Springer Science & Business Media, 1997.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

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

Deborah Ferreira, Julia Rozanova, Krishna Dubba, Dell Zhang, and Andre Freitas. On the evaluation of intelligent process automation. *arXiv preprint arXiv:2001.02639*, 2020.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: program-aided language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 2023. URL `https://proceedings.mlr.press/v202/gao23f.html`.

Lukas-Valentin Herm, Christian Janiesch, Alexander Helm, Florian Imgrund, Kevin Fuchs, Adrian Hofmann, and Axel Winkelmann. A consolidated framework for implementing robotic process automation projects. In *Business Process Management: 18th International Conference, BPM 2020, Seville, Spain, September 13–18, 2020, Proceedings 18*, pp. 471–488. Springer, 2020.

Peter Hofmann, Caroline Samp, and Nils Urbach. Robotic process automation. *Electronic markets*, 30(1):99–106, 2020.

Tian Huang, Chun Yu, Weinan Shi, Zijian Peng, David Yang, Weiqi Sun, and Yuanchun Shi. Promptrpa: Generating robotic process automation on smartphones from textual prompts. *arXiv preprint arXiv:2404.02475*, 2024.

Christian Hummert and Georgina Louise Humphries. *Property Lists*, pp. 157–165. Springer International Publishing, Cham, 2022. ISBN 978-3-030-98467-0. doi: 10.1007/978-3-030-98467-0_6. URL `https://doi.org/10.1007/978-3-030-98467-0_6`.

Lucija Ivančić, Dalia Suša Vugec, and Vesna Bosilj Vukšić. Robotic process automation: systematic literature review. In *Business Process Management: Blockchain and Central and Eastern Europe Forum: BPM 2019 Blockchain and CEE Forum, Vienna, Austria, September 1–6, 2019, Proceedings 17*, pp. 280–295. Springer, 2019.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

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

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *ArXiv preprint*, abs/2112.09332, 2021.

OpenAI. OpenAI: Introducing ChatGPT, 2022. URL https://openai.com/blog/chatgpt.

OpenAI. Gpt-4 technical report, 2023.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 15174–15186. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.acl-long.810.

Yujia Qin, Zihan Cai, Dian Jin, Lan Yan, Shihao Liang, Kunlun Zhu, Yankai Lin, Xu Han, Ning Ding, Huadong Wang, et al. Webcpm: Interactive web search for chinese long-form question answering. *arXiv preprint arXiv:2305.06849*, 2023a.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, et al. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*, 2023b.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*, 2020.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2024.

Haiyang Shen, Yue Li, Desong Meng, Dongqi Cai, Sheng Qi, Li Zhang, Mengwei Xu, and Yun Ma. Shortcutsbench: A large-scale real-world benchmark for api-based agents. *arXiv preprint arXiv:2407.00132*, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. *CoRR*, abs/2402.01030, 2024a. doi: 10.48550/ARXIV.2402.01030. URL `https://doi.org/10.48550/arXiv.2402.01030`.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*, 2024b.

Zilong Wang, Yuedong Cui, Li Zhong, Zimin Zhang, Da Yin, Bill Yuchen Lin, and Jingbo Shang. Officebench: Benchmarking language agents across multiple applications for office automation. *CoRR*, abs/2407.19056, 2024c. doi: 10.48550/ARXIV.2407.19056. URL `https://doi.org/10.48550/arXiv.2407.19056`.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

Judith Wewerka and Manfred Reichert. Robotic process automation–a systematic literature review and assessment framework. *arXiv preprint arXiv:2012.11951*, 2020.

Michael Wornow, Avanika Narayan, Krista Opsahl-Ong, Quinn McIntyre, Nigam H Shah, and Christopher Re. Automating the enterprise with foundation models. *arXiv preprint arXiv:2405.03710*, 2024.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*, 2023.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*, 2024.

Linyao Yang, Hongyang Chen, Zhao Li, Xiao Ding, and Xindong Wu. Chatgpt is not enough: Enhancing large language models with knowledge graphs for fact-aware language modeling. *arXiv preprint arXiv:2306.11489*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *ArXiv preprint*, abs/2210.03629, 2022.

Yining Ye, Xin Cong, Shizuo Tian, Jiannan Cao, Hao Wang, Yujia Qin, Yaxi Lu, Heyang Yu, Huadong Wang, Yankai Lin, et al. Proagent: From robotic process automation to agentic process automation. *arXiv preprint arXiv:2311.10751*, 2023.

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

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36, 2024.

A ALGORITHM OF TRANSCRIBING SHORTCUTS

Algorithm 1: Recursive Parsing of Property List to Construct Abstract Syntax Tree

```
Data: Shortcut file to be transcribed
Result: Abstract syntax tree of the actions
Initialize an empty tree with a root node and set current_node to root
foreach action in action list do
    Determine action_type and mode from action
    if action_type is Conditional then
        HandleConditional(mode, action)
    else if action_type is RepeatEach then
        HandleLoop(mode, action)
    else if action_type is RepeatCount then
        HandleLoop(mode, action)
    else if action_type is ChooseFromMenu then
        HandleMatchCase(mode, action)
    else
        HandleDefault(action)
Function HandleConditional(mode, action):
    if mode == 0 (start if) then
        AddNode(action)
        Set current_node to new node
    else if mode == 1 (else) then
        Move current_node to parent node
        AddNode(action)
        Set current_node to new node
    else if mode == 2 (end if) then
        Move current_node to parent node
Function HandleLoop(mode, action):
    if mode == 0 (start loop) then
        AddNode(action)
        Set current_node to new node
    else if mode == 2 (end loop) then
        Move current_node to parent node
Function HandleMatchCase(mode, action):
    if mode == 0 (start match) then
        AddNode(action)
        Set current_node to new node
    else if mode == 1 (start case) then
        if current_node is match node then
            AddNode(action)
            Set current_node to new node
        else
            Move current_node to parent match node
            AddNode(action)
            Set current_node to new node
    else if mode == 2 (end match) then
        Move current_node to parent node
Function HandleDefault(action):
    AddNode(action)
Function AddNode(action):
    Create new node with action
    Append new node to current_node.children
    Set parent of new node to current_node
```

B Prompt Design

B.1 Workflow Orchestration Prompt

~~~
You are a very helpful AI assistant who can write corresponding
   Python main code based on user's query and usable Python
   function interface.

Please generate python main code based on the following query :
 {query}
You can start by using natural language to plan your tool call
   strategy, and then generate the complete code. For example, `
   Thought:
<tool call strategy>

Code:
```python
<main code>
````.
Note that your output should always include `Code:
```python
<main code>
````, formatted accordingly.
Here are some useful function interface you may use:
 {apis_docs}
~~~

B.2 Evaluator Prompt

~~~
You are a kindly code reviewer, I will provide you with a query, a
    list of allowed apis and a piece of code to be reviewed, you
   help me to check if the code to be reviewed is compliant with
   our specifications.
The requirements are as follows:
1. You **should return True even if the code implements additional
    functionality not required in the query**, as long as it
   roughly implements the requirements in the query.
2. We don't impose any requirements on code readability or naming
   conventions. You **should return True as long as the reviewed
   code doesn't use disallowed functions and reasonably
   accomplishes what is asked in the query in general terms**.
   There's no need to get strictly hung up on the details.
3. Return False if the code fails to fulfill the requirement in
   the query. e.g. if it is proposed in the query to turn down
   the battery level of the phone and the brightness of the
   screen, it is a failure to fulfill only any one of the
   functions.
4. Built-in python syntax such as `if`, `loop`, `input()`, and `
   print()` are allowed.  Return False if the code uses **any
   external functions or apis** not in allowed apis list and not
   a built-in function such as input(), print(). For example, if
   I provide the is_workflow_openurl function, this should be
   used. Any use of any other library like requests etc. is a
   False.
query:{query}
list of allowed apis: {apis}
code to review: {code}

Your answer: [True or False with interpretation]
~~~

B.3 COMMENT GENERATION PROMPT

````
A Shortcut is a sequence of actons, where each action is an API call, to execute user-provided queries.
As a user-friendly and patient assistant, your task is to provide a set of description of each line of the code scrippet. To save time, I have retrieved all the lines exclusive of blank lines of the code snippet and listed as a dictionary below the code.

Your answer should be in the json format as follows:
```json
{
    "line x": "<description-of-line-x>",
    "line x+1": "<description-of-line-x+1>",
    "...": "...",
    "line x+n": "<description-of-line-x+n>"
}```

The code is :
{code}
The lines are {lines}
````

B.4 TASK PLAN GENERATION PROMPT

```
Based on this line by line description of the code, generate a flowchart of a workflow by natural language.
This is the code:
{code}
```

B.5 TASK QUERY GENERATION PROMPT

```
As a helpful assistant, please help me craft a query. This query, formatted as a question, should describe the task a user wants to complete and adhere to the following criteria:
1. One of the solution to the task described in the query could be the python code below.
2. It should be close to real-world problems or requests.
3. It should include major parts of the code.
4. The query should not specify python.

For example, the code is:
{ICL_code}
And the expected output query should be similar to:
{ICL_query}

Please craft a query based on the examples and the following code:
{code}
```

B.6 QUERY EXPANSION PROMPT

````
You are exceptionally skilled at crafting real-world user queries given some apis. Here are examples:{examples}. Please gain inspiration from the following api docs to create a high-quality realworld query.
Api docs for inspiration:
```python
````

```
{apis_string}
```
```
Please refer to the above examples and craft a new one!
Requirements: API name is strictly prohibited from appearing in
    the generated query. Each query should be complicated enough
    and can be solved using all apis above. The query **should be
    centered around {category} theme** and should not be spread
    out into unrelated pieces.
```

B.7 QUALITY CONFIRMATION PROMPT

```
You are exceptionally skilled at polishing tool calling plan (i.e
    ., thought) and python code given a task.

Given task:
{query}


Old tool calling plan:
{thought}

 Old code:
{code}

 Used API doc:
{apis}

Here are examples for you to refer:{ICL_context}.
Please make sure the code is logically correct and operational.

Requirements:
[1] Ensure that both plan and code respond correctly to the task
    and that code calls match the plan, which you can do by
    tweaking, embellishing, and modifying both plan and code.
Plan does not have to be one-to-one correspondence of code; plan
    can be abbreviated.
[2] Please ensure that the code conforms to python syntax. Ensure
    that all python code is complete and runnable. You can add
    code when necessary.
[3] Every line of code should be preceded by a comment marked with
     a "#". When modifying the code, please modify the in-line
    comments accordingly.
[4] Ensure that all function parameter calls are correct and you
    can change the code in case of errors.
[5] Thought and code should be as concise while keeping the
    meaning intact.
[6] If there are cases including invalid binary code, replace them
     with reasonable text, delete them, or replace them with a
    reading operation on a file (especially when the binary code
    is an encoded image).
Respond strictly with JSON.
```

B.8 VARIABLE RENAME PROMPT

```
You are a helpful assistant for renaming variable names in a code
    snippet.
The following code snippet is a part of a program, and variables
    are named in format 'variablex_'.
```

```
Your task is to rename these variables so that they conform to the
     programming specification and have some semantic meaning,
    which can be infered by relative function calls
And your output should only be a dictionary containing the old
    name-new name key value pair
The definition of some functions are not included, and you shouldn
    't modify them.
Following the code, there's a dictionary that contains short
    description of the uuid-named variable. And you can take it as
     reference.
Note that while the description might be the same, but the actual
    meaning is different across different variables. So you should
     not just copy the short description. Instead you'd better
    conprehensively consider the description, names of called
    functions, and the general logic.
The code is as follows:
{code}
The dictionary is as follows:
{description}
To save time, I have retrieved all the variables that requires to
    be renamed:
{variables}
```

C Case Study of Shortcuts

We provide a real-world shortcut example, which includes the following three presentation forms: the rwa property list configuration file, the Python code after transcription and variable renaming, and the visual interface on MacOS.

The raw property list configuration file is presented below. For the sake of brevity, we have omitted the middle portion containing the actions.


```
{
  "WFWorkflowClientVersion": "754",
  "WFWorkflowClientRelease": "2.1.2",
  "WFWorkflowMinimumClientVersion": 411,
  "WFWorkflowIcon": {
    "WFWorkflowIconStartColor": 4274264319,
    "WFWorkflowIconImageData": "b''",
    "WFWorkflowIconGlyphNumber": 59672
  },
  "WFWorkflowImportQuestions": [],
  "WFWorkflowTypes": ["WatchKit", "ActionExtension"],
  "WFWorkflowInputContentItemClasses": ["WFURLContentItem"],
  "WFWorkflowActions": [
    {
      "WFWorkflowActionIdentifier": "is.workflow.actions.count",
      "WFWorkflowActionParameters": {
        "WFCountType": "Items",
        "UUID": "F292DD85-A8D2-4EBF-93E8-AC45F1C38310"
      }
    },
    {
      "WFWorkflowActionIdentifier": "is.workflow.actions.conditional",
      "WFWorkflowActionParameters": {
        "WFControlFlowMode": 0,
        "WFConditionalActionString": "0",
        "GroupingIdentifier": "51B09BBE-EF2D-4635-B820-412BADC6D64C",
        "WFCondition": "Equals"
      }
```

```
    },
    ...
    {
      "WFWorkflowActionIdentifier": "is.workflow.actions.conditional",
      "WFWorkflowActionParameters": {
        "GroupingIdentifier": "05DA8CFC-73E5-47EC-BBF6-7A23BD4D6C27",
        "WFControlFlowMode": 2
      }
    }
  ]
}
```

It can be observed that this configuration file employs non-semantic hexadecimal strings to represent variables and uses keywords such as `is.workflow.actions.conditional` and `GroupingIdentifier` to implement logic controls like conditions, making it inherently difficult to read and comprehend. Consequently, we have converted it into a Python-like code format. The Python code, after transcription, variable renaming, and commenting, is shown as follows:

```python
# This line calls the function is_workflow_actions_count with a parameter
↪ of WFCountType set to 'Items', which checks the count of workflow
↪ actions related to items and assigns the result to
↪ workflow_action_count.
workflow_action_count = is_workflow_actions_count(
↪ WFCountType='''Items''')
# This line checks if the workflow_action_count is equal to '0', which
↪ means there are no available actions for items.
if workflow_action_count == '''0''':
    # If there are no actions, this line calls the function
    ↪ is_workflow_actions_url with a parameter of WFURLActionURL set to
    ↪ a specific Amazon URL to get the URL for the workflow actions and
    ↪ assigns it to workflow_action_url.
    workflow_action_url = is_workflow_actions_url(
    ↪ WFURLActionURL='''https://www.amazon.com/gp/history''')
    # This line displays the webpage defined by workflow_action_url by
    ↪ calling the is_workflow_actions_showwebpage function.
    is_workflow_actions_showwebpage( WFURL=workflow_action_url)
# This line starts the else clause that executes if 'UpdateKit' is not
↪ found in my_workflows.
else:
    # In this line, the code prompts the user for input with 'Please
    ↪ enter the value:', captures it, and calls the function
    ↪ is_workflow_actions_getvariable to get a corresponding variable
    ↪ and assigns the result to user_input_value.
    user_input_value = is_workflow_actions_getvariable(
    ↪ WFVariable=f'{input("Please enter the value:")}')
    # This line processes the user_input_value by calling the function
    ↪ is_workflow_actions_detect_link, which extracts a link from the
    ↪ user's input, and assigns the detected link to detected_link.
    detected_link = is_workflow_actions_detect_link(
    ↪ WFInput=user_input_value)
    # Here, the detected_link is used as input for the function
    ↪ is_workflow_actions_getitemfromlist to retrieve an item from a
    ↪ list and assigns the result to item_from_list.
    item_from_list = is_workflow_actions_getitemfromlist(
    ↪ WFInput=detected_link)
    # Finally, this line displays the webpage associated with the
    ↪ retrieved item from item_from_list by calling
    ↪ is_workflow_actions_showwebpage.
    is_workflow_actions_showwebpage( WFURL=item_from_list)
# This line retrieves the user's workflows by calling the function
↪ is_workflow_actions_getmyworkflows and assigns the result to
↪ my_workflows.
my_workflows = is_workflow_actions_getmyworkflows()
# This line checks if 'UpdateKit' exists in the user's workflows.
```

```
if '''UpdateKit''' in my_workflows:
    # If 'UpdateKit' is found, this line creates a dictionary named
    ↪  updatekit_details that contains the details for the update kit,
    ↪  including its name, version, and RoutineHub ID.
    updatekit_details = {'''Shortcut Name''': '''Buy Kindle Book''',
    ↪  '''Current Version''': '''1.0''', '''RoutineHub ID''':
    ↪  '''1360'''}
    # This line calls the function is_workflow_actions_runworkflow to
    ↪  execute the workflow named 'UpdateKit' with the parameters
    ↪  WFShowWorkflow set to False and WFInput set to the details from
    ↪  updatekit_details.
    is_workflow_actions_runworkflow( WFWorkflowName='''UpdateKit''',
    ↪  WFShowWorkflow=False, WFInput=updatekit_details)
    # This line contains the pass statement, indicating that if
    ↪  'UpdateKit' is not found, the program will do nothing.
    pass
```

We can clearly see that the transcribed Python code is of high quality, with strong readability, making it well-suited for training LLMs that have been pre-trained on extensive Python code.

For easy understanding, we also present the visual interface of this shortcut in Figure 7, which corresponds to the transcribed Python code on a line-by-line basis.

D Case Study of WorkflowBench

In this section, we provide a workflow example of WorkflowBench based on the shortcut listed in Appendix C. We omit the annotated workflow code that has been shown in Appendix C.

The query $\mathcal{Q}$ is:

```
Could you guide me on how to design a solution that manages workflow
    actions? I need it to check for existing items, retrieve purchase
    history from a specific source if no items are present, and also
    provide the option to input a specific item for detailed retrieval if
     items are available. Additionally, I want the solution to trigger a
    specific workflow if it is available within my existing processes.
```

The list of provided APIs is:

```
[
    'is.workflow.actions.showwebpage',
    'is.workflow.actions.getitemfromlist',
    'is.workflow.actions.getvariable',
    'is.workflow.actions.url',
    'is.workflow.actions.getmyworkflows',
    'is.workflow.actions.count',
    'is.workflow.actions.runworkflow',
    'is.workflow.actions.detect.link'
]
```

The task plan $\mathcal{P}$ is:

```
1. **Start**
   - The process begins.
2. **Retrieve Workflow Action Count**
   - Call the function `is_workflow_actions_count` with parameter `
     WFCountType` set to 'Items'.
   - Store the result in `workflow_action_count`.
3. **Check Workflow Action Count**
   - **Decision**: Is `workflow_action_count` equal to '0'?
     - **Yes**: Proceed to step 4.
     - **No**: Proceed to step 10.
4. **Get Workflow Action URL**
   - Call the function `is_workflow_actions_url` with parameter `
     WFURLActionURL` set to 'https://www.amazon.com/gp/history'.
   - Store the result in `workflow_action_url`.
```

```
5. **Display Webpage**
   - Call the function `is_workflow_actions_showwebpage` with `WFURL` set
      to `workflow_action_url`.
6. **End Workflow Action Check**
   - End the process of checking workflow actions if the count is '0'.
7. **Retrieve My Workflows**
   - Call the function `is_workflow_actions_getmyworkflows`.
   - Store the result in `my_workflows`.
8. **Check for UpdateKit**
   - **Decision**: Is 'UpdateKit' in `my_workflows`?
     - **Yes**: Proceed to step 9.
     - **No**: Proceed to step 11.
9. **Handle UpdateKit**
   - Create a dictionary `updatekit_details` with:
     - 'Shortcut Name': 'Buy Kindle Book'
     - 'Current Version': '1.0'
     - 'RoutineHub ID': '1360'
   - Call the function `is_workflow_actions_runworkflow` with:
     - `WFWorkflowName` set to 'UpdateKit'
     - `WFShowWorkflow` set to False
     - `WFInput` set to `updatekit_details`.
10. **Receive User Input (if no update kit)**
   - Prompt user: "Please enter the value:".
   - Capture input and call the function `is_workflow_actions_getvariable
     `.
   - Store the result in `user_input_value`.
11. **Detect Link from Input**
   - Call the function `is_workflow_actions_detect_link` with `WFInput`
     set to `user_input_value`.
   - Store the result in `detected_link`.
12. **Get Item from List**
   - Call the function `is_workflow_actions_getitemfromlist` with `
     WFInput` set to `detected_link`.
   - Store the result in `item_from_list`.
13. **Show Item Webpage**
   - Call the function `is_workflow_actions_showwebpage` with `WFURL` set
      to `item_from_list`.
14. **End Process**
   - The process concludes after performing the respective actions based
     on the conditional checks.
```

E LIMITATIONS

While the framework proposed in this paper represents notable progress in workflow orchestration, it also has certain limitations that warrant discussion. First, the APIs used in our work are exclusively derived from Apple Shortcuts application, resulting in a lack of coverage across more diverse fields and potentially limiting the generalizability of the dataset to broader application contexts. Second, our approach lacks evaluation through actual execution. This limitation arises due to the complexities involved in executing workflows, such as the need for intricate user registration and permission acquisition. Moreover, the APIs are subject to frequent changes as applications continue to evolve, making it challenging to implement a consistent executable evaluation. Consequently, our evaluation is limited to static analysis.

F ETHICAL STATEMENT

In this study, the dataset construction process was fully automated using LLMs and algorithms for data annotation, eliminating the need for human annotators and thereby avoiding concerns related to annotator compensation and working conditions. The data utilized was collected through web scraping from publicly accessible sources, with strict adherence to the Terms of Service (ToS) of the respective websites. Scraping was avoided on platforms where such activity is explicitly prohibited, ensuring compliance with ethical standards. Additionally, no personally identifiable information (PII) or private user data was collected at any stage of the research process. All data was anonymized to protect privacy and mitigate any potential ethical concerns related to user information.

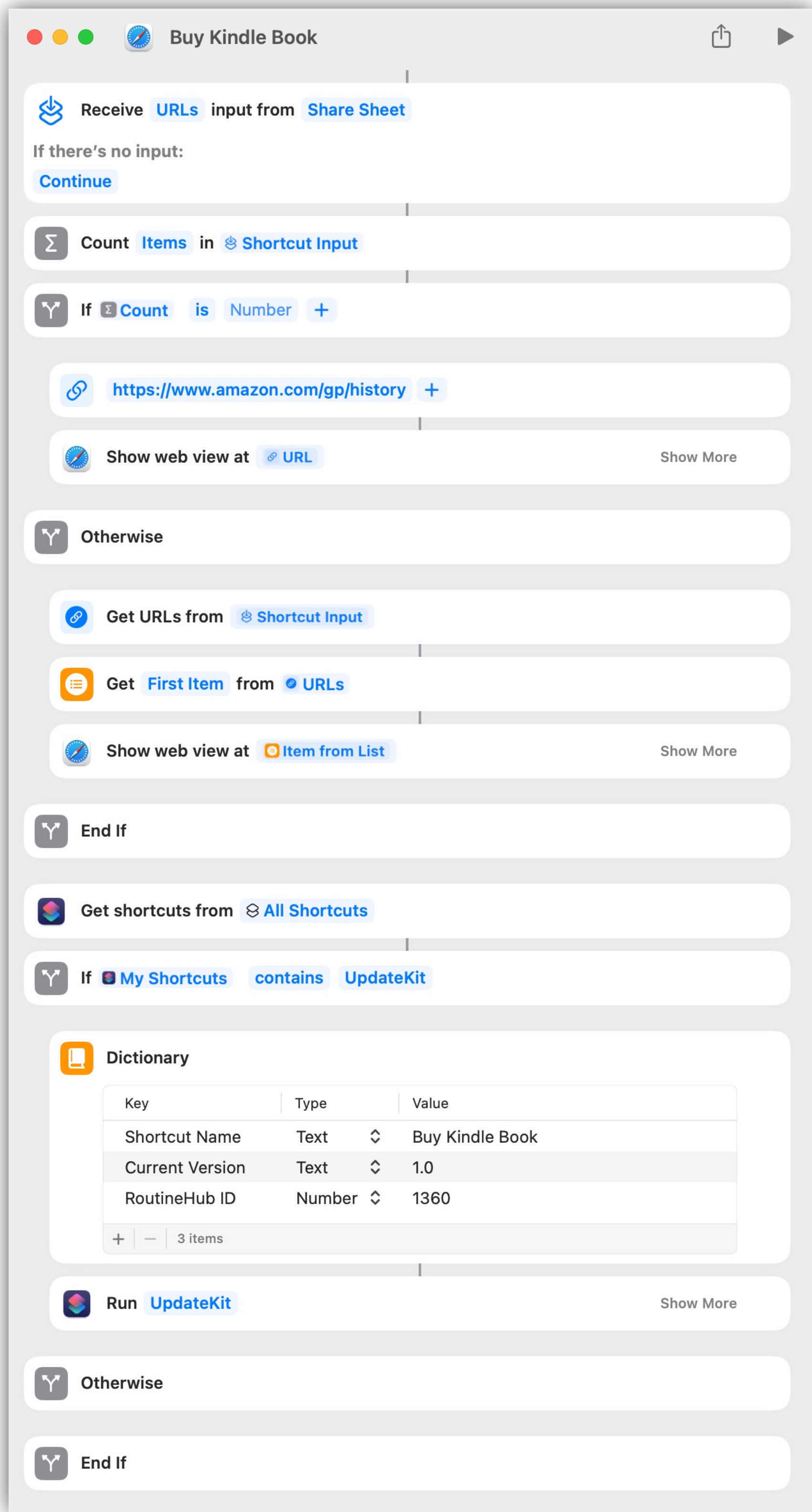


Figure 7: The visual interface of the shortcut `RoutineHub • Buy Kindle Book`.