Automating Complex Document Workflows via Stepwise and Rollback-Enabled Operation Orchestration

Yanbin Zhang, Hanhui Ye, Yue Bai, Qiming Zhang, Liao Xiang, Wu Mianzhi, Renjun Hu*

East China Normal University

ybzhang@dase.ecnu.edu.cn, 51265903057@stu.ecnu.edu.cn, baiyue@cc.ecnu.edu.cn, qmzhang@stu.ecnu.edu.cn, 51285903046@stu.ecnu.edu.cn, mianzhiwu@stu.ecnu.edu.cn, rjhu@dase.ecnu.edu.cn

Abstract

Workflow automation promises substantial productivity gains in everyday document-related tasks. While prior agentic systems can execute isolated instructions, they struggle with automating multi-step, session-level workflows due to limited control over the operational process. To this end, we introduce **AutoDW**, a novel execution framework that enables stepwise, rollback-enabled operation orchestration. AutoDW incrementally plans API actions conditioned on user instructions, intent-filtered API candidates, and the evolving states of the document. It further employs robust rollback mechanisms at both the argument and API levels, enabling dynamic correction and fault tolerance. These designs together ensure that the execution trajectory of AutoDW remains aligned with user intent and document context across long-horizon workflows. To assess its effectiveness, we construct a comprehensive benchmark of 250 sessions and 1,708 human-annotated instructions, reflecting realistic document processing scenarios with interdependent instructions. AutoDW achieves 90% and 62% completion rates on instruction- and session-level tasks, respectively, outperforming strong baselines by 40% and 76%. Moreover, AutoDW also remains robust for the decision of backbone LLMs and on tasks with varying difficulty.

Code — https://github.com/YJett/AutoDW

Introduction

Automating document-related tasks, which constitute a substantial portion of the intellectual workload, remains a persist challenge.With the advent of large language models (LLMs) (Brown et al. 2020), significant progress has been made in language understanding, instruction following, and multi-step planning (Comanici et al. 2025; Guo et al. 2025). This has opened new possibilities for workflow automation that involve interpreting instructions, maintaining context, and translating intent into structured actions. Explorations in code generation (Ishibashi and Nishimura 2024), data science (Hong et al. 2025), and web-based tasks (Yang et al. 2025) have demonstrated encouraging results. These advances highlight the promise of extending LLM-powered automation to complex document workflows for productivity improvement.

Several lines of research could apply for this purpose. General tool use or workflow orchestration models (Schick and Schütze 2023; Fan et al. 2024) have been developed to facilitate effective API selection. However, they are not tailored for precise document state management, especially in long-horizon tasks. Other document-oriented agents adopt a human-in-the-loop paradigm, sacrificing automation for alignment (Mathur et al. 2024; Liang et al. 2025). More closely related to our setting are recent agentic systems designed for instruction-based automation (Guo et al. 2024). Leveraging the native planning ability of LLMs (Yao et al. 2023; Rawat et al. 2025), these systems have demonstrated promising performance in isolated instruction execution. **Nonetheless**, such agents typically employ predetermined plans without context-sensitive adjustments, struggling to precisely align flexible and, sometimes, ambiguous user instructions with evolving document states. **Moreover**, they generally lack error recovery mechanisms, *i.e.*, directly proceeding without validating outcomes. Consequently, even a single erroneous API call can propagate silently, limiting effectiveness in managing session-level multi-step workflows (Wang et al. 2024b; Yao 2023).

These challenges underscore the necessity of more structured and precisely controlled operation orchestration for automating complex document workflows. To address this, we introduce AutoDW, a novel execution framework that provides a general and robust orchestration paradigm through the integration of stepwise planning and adaptive rollback. Unlike previous methods that treat the task as a monolithic process, AutoDW incrementally decomposes workflows into atomic operations. Each operation is executed and verified individually, allowing errors or misalignments with user intent to be detected early and corrected promptly.

Specifically, AutoDW incrementally plans API actions one at a time, conditioning each choice on the user instruction, intent-filtered candidate APIs, and the latest document state. Within a Python runtime environment, AutoDW executes the selected API and extracts the updated document state. It then invokes an LLM to summarizes the state change before and after execution. In the adaptive rollback stage, an LLM-based validator assesses the alignment of the state change with user intent, providing a confidence score and detailed explanation. Based on this feedback, AutoDW chooses to accept the API or employs two distinct rollback

*Corresponding author

strategies: argument-level rollback, which revises only the API arguments while preserving the API selection, and API-level rollback, which completely re-generates an alternative API. By integrating precise incremental planning with adaptive rollback, AutoDW effectively maintains alignment with user intent throughout complex sessions, significantly reducing the risk of cascading errors.

To rigorously evaluate this robust orchestration capability, we construct a new benchmark **DWBench** designed to reflect realistic document processing scenarios involving highly interdependent and long-horizon instructions. DWBench consists of 250 multi-turn sessions (1,708 instructions total). To support these workflows, we implement 74 APIs and hired graduate-level volunteers to annotate each instruction with a feasible API sequence. Notably, most instructions require multiple API calls, with each session averaging 34.8 APIs (min 15, max 75), demonstrating the non-triviality and depth of DWBench for automation.

For evaluation, we adopt an operational correctness metric inspired by recent agent benchmarks (Zhou et al. 2024; Yao et al. 2025).Specifically, we ask an LLM-based judge to compare the programatically-extracted document state after each executed instruction against the ground-truth state obtained by replaying the annotated API sequence. A task is considered successfully completed if the LLM determines the two states are semantically equivalent.To ensure reliability, we conduct a specialized analysis to verify the judge's high agreement with human verification on a sampled subset of tasks.

With DWBench, we find that AutoDW achieves 90% instruction-level and 62% session-level completion rates, outperforming strong baselines by 40% and 76%, respectively. The latter improvement is achieved by using an extra of 25.6% tokens, demonstrating significant gains in end-to-end workflow completion. Our robustness study also verifies that AutoDW generalizes well across different backbone LLMs (DeepSeek-v3, Qwen-Plus, Gemini-2.5-Pro, and GPT-4.1) and varying instruction difficulty. Moreover, the performance on hard instructions (*i.e.*, those require > 6 APIs to fulfill) is only 4.4% lower than the overall suggesting its practical applicability in real-world usage scenarios. Finally, our ablation study convincingly confirms the rationale of our rollback strategies, which could strike a good balance between performance gain and extra cost with single-round dual-level rollback.

In summary, this paper makes the following contributions:

- We introduce AutoDW, which integrates an innovative combination of stepwise planning and adaptive rollback for complex document workflow automation.
- We construct DWBench for evaluation, which features 250 sessions of 1,708 human-annotated instructions, 74 APIs, and an operational correctness metric.
- With DWBench, we empirically verify that AutoDW obtains the state-of-the-art performance the the task, as well as its robustness and rationale in model design.

Related Work

LLM-based document workflow automation. Recent advances in LLMs have enabled new possibilities for automating document-related tasks through natural language interfaces. Early work demonstrated their potential for basic text understanding and generation (Brown et al. 2020), but subsequent evaluations revealed limitations in reliably executing multi-step document operations (Bang et al. 2023; Borji 2023). Specialized architectures (e.g., LayoutLM (Xu et al. 2020), DocLLM (Wang et al. 2024a)) have advanced document understanding, but focus on comprehension rather than execution reliability.

General LLM-based agents (e.g., ReAct (Yao et al. 2023), Toolformer (Schick and Schütze 2023)) provide effective frameworks for planning and tool use, but are not designed for the document domain.

Existing document automation approaches typically fall into three paradigms. Retrieval-based methods rely on semantic similarity to select relevant APIs (Karpukhin et al. 2020), but they often misalign with user intent due to the gap between natural language and function semantics. Reasoning-only approaches employ LLMs to generate executable actions directly (Wei et al. 2023), offering flexibility but suffering from brittleness and inconsistency in long-horizon workflows. Hybrid approaches (Guo et al. 2024; Mathur et al. 2024) combine planning, selection, verification, and execution, but their predetermined planning lacks adaptability and error recovery is generally neglected.In contrast, AutoDW directly addresses these challenges with state-aware stepwise planning and rollback mechanisms.

Execution reliability and fault recovery in agents. Reliability is a critical requirement for automation systems, especially in high-stakes or productivity-focused environments (Vendrow et al. 2025). Traditional error handling strategies have evolved into self-healing (Lu et al. 2024) and multi-level validation (Zheng, Lapata, and Pan 2024). However, these domain-agnostic systems struggle with document-specific concerns where subtle changes propagate unintended effects.

Efforts to improve robustness in sequential tasks include stepwise execution in fields like program synthesis (Wang, Jha, and Jha 2024) and robot planning (Paxton et al. 2019). These techniques operate in deterministic environments, which are not directly transferable to document workflow automation characterized by natural language ambiguity inherent in natural language instructions. AutoDW extends these ideas by introducing intent-conditioned validation and rollback that accommodate natural language variability.

Document automation benchmarking. Despite growing interest in document automation, evaluation frameworks have lagged behind in assessing real-world task reliability. Previous benchmarks emphasize static analysis tasks such as information extraction (Mathew, Karatzas, and Jawahar 2021). DocBench (Zou et al. 2024) focuses on document analysis but does not evaluate execution reliability. PPTC (Guo et al. 2024) introduced a benchmark for multi-turn PowerPoint automation, where GPT-4 achieved only 6% session completion. Similarly, we construct DWBench, a bilingual benchmark for Word automation (250 work-

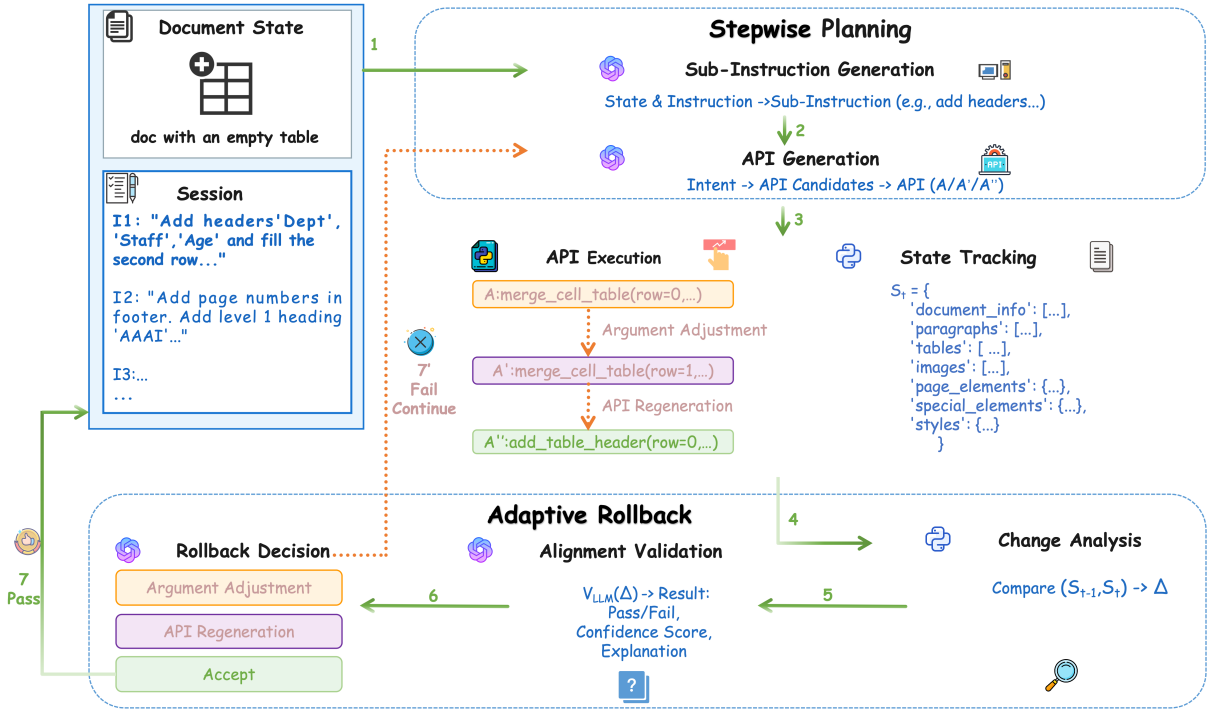

Figure 1: Overview of the AutoDW framework, which comprises three core modules: stepwise planning, API execution & state tracking, and adaptive rollback. The overview also includes an illustrative example showing how AutoDW selects one API call at a time and corrects its mistakes (*i.e.*, APIs A and A') through rollback.

flows), featuring human-annotated API sequences and operational correctness evaluation, offering a realistic measure for production-grade agents.

Each instruction is annotated by human experts with a feasible API sequence, and it adopts operational correct for evaluation. Together, these features make DWBench a more realistic measure of readiness for production-grade document automation agentic systems.

Methodology

In this section, we present our proposed framework, AutoDW, which performs stepwise and rollback-enabled operation orchestration for complex document workflow automation. An overview of the system architecture is shown in Fig. 1 Given an initial document state $\mathcal{S}_0$ (*i.e.*, a Word file) and a session of natural language instructions $\mathcal{I} = \{I_1, I_2, \dots\}$, AutoDW plans and executes a sequence of API calls $\mathcal{P} = [A_1, A_2, \dots]$ that incrementally transform $\mathcal{S}_0$ in alignment with the user intent expressed in $\mathcal{I}$. Each API A_i is selected from a predefined API set $\mathcal{A}$, which AutoDW has full access to. At a high level, AutoDW comprises three core modules: a stepwise planning module for incrementally selecting the next API, a Python runtime environment for executing APIs and tracking document states, and an adaptive rollback module for validating and potentially reverting previously executed operations. We now describe each of these components in detail.

Stepwise Planning

Traditional instruction decomposition approaches generate all atomic operations upfront, which often leads to execution failures as the document state evolves. In contrast, our stepwise planning mitigates this limitation by generating API calls one at a time, conditioned on real-time document state, ensuring that each operation adapts to the current execution context. While this is a locally guided approach, the comprehensive and up-to-date document state encoding (as detailed in Table 1) provides sufficient context, effectively preventing the system from falling into low-efficiency or dead-end local optima during complex, long-horizon sessions.Specifically, we adopt a two-stage planning strategy: first generating a sub-instruction, and then the corresponding API action.

Sub-instruction generation. Each sub-instruction corresponds to a single atomic document operation that can be completed with one API call. The use of sub-instructions serves two key purposes. First, it bridges the semantic gap between potentially ambiguous natural language instructions and concrete API functionalities. Second, it enables intent classification, which narrows down the API search space during API generation by identifying the likely user intent behind the atomic operation. Sub-instruction generation is performed by an LLM prompted with a comprehensive template that includes the task objective, the original user instruction, previously completed API calls, the current document state (as captured in the state tracking module), detailed guidelines, output format specifications in JSON,

and in-context examples.[1]

API generation. Once a sub-instruction is generated, AutoDW classifies its underlying intent. Given that document-related intents are relatively fixed and well-defined, we fine-tune a 178M BERT model (Devlin et al. 2019) for 8-way intent classification: *content creation*, *content modification*, *table operation*, *image operation*, *chart operations*, *format/style editing*, *document structure update*, and *document lifecycle update*. The classifier is fine-tuned on 3,315 instruction–intent pairs (with no overlap with the DWBench benchmark), and achieves a test accuracy of 98%. This choice, instead of using the LLM for classification, is primarily motivated by efficiency and cost: for a fixed and well-defined set of 8 intents, BERT maintains high accuracy while significantly reducing inference latency and computational overhead, optimizing the overall resource consumption of the framework."

To improve robustness to ambiguous instructions, we retain the top-3 predicted intents instead of relying solely on the top-1. This allows AutoDW to better capture the true intent behind diverse user inputs. The APIs associated with the top-3 detected intents are then retrieved and, along with the sub-instruction and other prompt engineering elements, fed into an LLM to generate the final API action, *i.e.*, an API and its full arguments. Despite the framework's multi-stage reliance on LLMs, our design incorporates a subsequent robust validation and rollback mechanism that systematically mitigates the risk of LLM errors accumulating and propagating across different modules in the execution chain."

For further details on intent categories, associated APIs, and the BERT fine-tuning procedure, please refer to the supplementary material.

API Execution and State Tracking

$$\mathcal{S}_t = (\mathcal{D}_t^{\text{doc}}, \mathcal{D}_t^{\text{para}}, \mathcal{D}_t^{\text{table}}, \mathcal{D}_t^{\text{image}}, \mathcal{D}_t^{\text{page}}, \mathcal{D}_t^{\text{int}}, \mathcal{D}_t^{\text{style}}), \quad (1)$$

where each $\mathcal{D}$ is a list of dict objects that describes all elements of a specific type in $\mathcal{S}_t$. We summarize the detailed attributes (*i.e.*, dict keys) to track for each type of components in Table 1.

Robustness against State Tracking Failures: We address this by treating a state parsing failure as an invalid execution outcome. If Python Runtime fails to extract a valid document state, the subsequent adaptive rollback module recognizes this as a critical misalignment with expected user intent, triggering an API-level rollback. This mechanism prevents the system from generating subsequent plans based on unreliable or erroneous document states.

Adaptive Rollback

The rollback module in AutoDW is responsible for handling fault recovery during execution. It achieves this by summarizing the document state change after an API call and validating whether the change aligns with the user's intent. AutoDW supports two types of rollback mechanisms: argument-level and API-level. The use of rollback is adaptively determined by an alignment validator.

[1]All prompt templates used by AutoDW are provided in the supplementary material.

Change analysis. We implement an `analyze_change` function which employs a multi-layered approach to detecting changes between document states. The analysis operates on state pairs $(\mathcal{S}_{t-1}, \mathcal{S}_t)$ and outputs:

$$\Delta = (\Delta_S, \Delta_C, \Delta_F, \Delta_{St}, \Delta_T, \Delta_H), \quad (2)$$

that represent structural, content, format, style, table, and hyperlink changes, respectively. These changes are captured by analyzing the seven-component document state: structural analyzer processes document info, paragraphs, tables, images, and page elements for element count changes; content analyzer examines paragraph and table text modifications; format analyzer handles run-level formatting within paragraphs and tables; style analyzer tracks paragraph style changes; table analyzer focuses on table-specific structural and content changes; and hyperlink analyzer processes special elements for link modifications.

The content analyzer uses Python's SequenceMatcher to detect text modifications, generating operation codes $\mathcal{O} = \{(\text{tag}, i_1, i_2, j_1, j_2)\}$ where $\text{tag} \in \{\text{insert}, \text{delete}, \text{replace}, \text{equal}\}$. The structural analyzer compares element counts and positions, while table analysis uses cell signature comparison:

$$\text{sig}_{i,j} = \text{hash}(\text{content} \| \text{structure} \| \text{position} \| \text{merge}) \quad (3)$$

Alignment validation and rollback. To perform validation, the system passes the analyzed state change Δ, along with the sub-instruction, current document state, and previously executed APIs, to an LLM. The validator LLM is prompted to return: (1) a binary decision (pass or fail), (2) a confidence score in the range $[0, 1]$, and (3) a textual explanation supporting its judgment. Based on the result, AutoDW decides whether to accept the action or initiate rollback. Specifically, the action is accepted if the LLM decision is 'pass' with high confidence (we use an empirically tuned threshold of 0.6), or if the confidence score is low. The selection of this threshold is supported by a sensitivity analysis in Ablation Study, demonstrating its optimal balance between false negatives and false positives. In the latter case, low confidence is interpreted as the validator being uncertain, and the action is accepted by default. In all other cases, AutoDW initiates a rollback to avoid potential execution errors.

Rollback follows a two-step routine. First, argument-level rollback is applied: AutoDW regenerates the action using the same API but with updated arguments, taking validator's explanation into consideration. If this revised action also fails validation, the system escalates to API-level rollback, where a new API is selected altogether. These two steps together form a full rollback round. While additional rollback rounds are computationally possible, AutoDW defaults to a single round of rollback. This pragmatic choice is justified by our empirical analysis (see Section 4.4), which shows only marginal performance gains beyond the first attempt, making the single-round strategy the most cost-effective solution. With API-level rollback, regardless of the result, the most recent action is accepted and execution proceeds.

Running example. An example of AutoDW's stepwise planning and self-correction through rollback is illustrated in Fig. 1. Given the instruction *"Add headers*

Component Type	Tracking Attributes
Document Info	total paragraphs count, total tables count, total sections count, has header flag, has footer flag
Paragraph Elements	index, text content, style name, text alignment, text runs[1], spacing, indentation, embedded images[2]
Table Elements	index, row count, column count, cell matrix, table style, row heights, column widths
Image Elements	host paragraph index, host text run index, image sequence index, width, height
Page Layout Elements	headers, footers, page numbers, watermarks, table of contents
Interactive Elements	hyperlinks, bookmarks, line breaks, page breaks
Document Styles	style name, style category, font name, font size, bold flag, italic flag

Table 1: Document state tracking details. [1]Text runs are formatted text segments within a paragraph (e.g., "normal **bold** normal" contains 3 runs). [2]Embedded images are positioned within paragraph text runs, not standalone image objects.

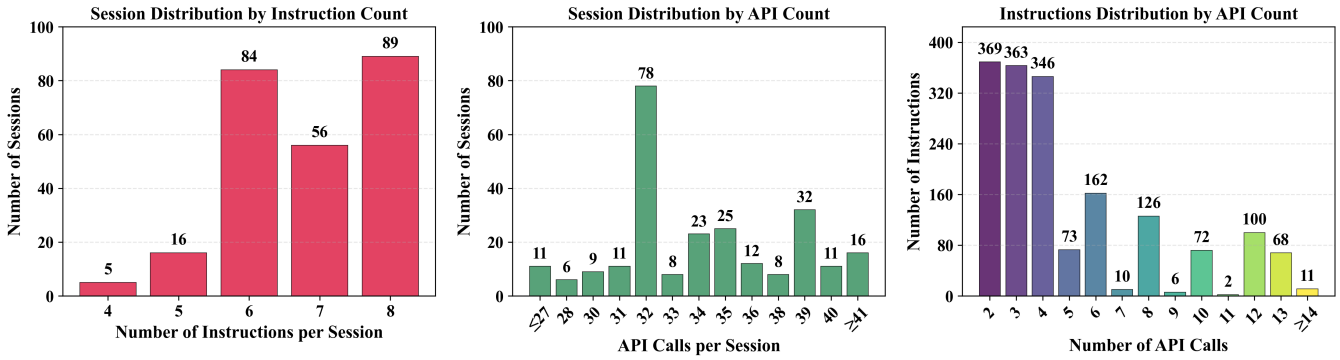

Figure 2: Distributional statistics of DWBench, which includes 250 sessions and 1,708 instructions. **Left**: Number of instructions per session (range: 4–8, mean=6.8), with a peak at 8 (89 sessions, 35.6%). **Middle**: Number of API calls per session (range: 15–75, mean=34.8), peak at 32 (78 sessions, 31.2%). **Right**: Number of API calls per instruction (range: 2–22, mean=5.1), while most instructions require 2-–4 API calls, complex instructions with $ge10$ calls account for 14.8%.

'Dept', 'Staff', 'Age' and fill the second row ...", AutoDW first decomposes the instruction into a sub-instruction *"add headers ..."*. However, the initially generated API is `merge_cell_table(row=0,)`, which is incorrect. After executing this API, updating the document state, and analyzing the resulting state change, the alignment validator rejects the action due to misalignment with the intended operation. AutoDW then initiates rollback. It first attempts argument-level rollback, generating a revised action with `row=1`, but this is still rejected. The system then escalates to API-level rollback, producing a new API: `add_table_header(row=0,)`, which is aligned with the sub-instruction and thus accepted. With this correction, AutoDW resumes planning the next sub-instruction in the instruction.

Experiments

We conduct an extensive evaluation of AutoDW from three perspectives: (1) a comparative analysis of overall performance against representative baselines, (2) a robustness study examining performance across different backbone LLMs and tasks with varying difficulty, and (3) an ablation study investigating the contributions of key components.

Evaluation Setup

We first introduce the experimental setting.

Benchmarking data. To evaluate the effectiveness of AutoDW, we construct a benchmark, DWBench, consisting of 250 sessions and 1,708 instructions. Each instruction is associated with: (i) a turn ID, (ii) a user-issued natural language instruction, (iii) an initial document state file, (iv) a human-annotated sequence of feasible API calls that fulfill the instruction, and (v) the corresponding expected document state obtained by executing the annotated API sequence on the initial state. The distributional statistics of DWBench are visualized in Fig. 2.

Baseline approaches. Document automation agents most relevant to our work include DocPilot (Mathur et al. 2024), TableTalk (Liang et al. 2025), and the systems evaluated in PPTC (Guo et al. 2024). We exclude DocPilot and TableTalk from our comparison, as both rely on human-in-the-loop verification, which is incompatible with our fully automated evaluation setup. Adapting the evaluation protocol established in PPTC, we consider the following three baselines for comparison:

(1) Retrieval-only: This baseline performs semantic matching between the user instruction I and the available APIs in $\mathcal{A}$, followed by naive execution using default arguments. First, both the instruction and each API description are embedded into dense vectors, and cosine similarity is computed. APIs with similarity above a threshold ($\tau = 0.75$) are retained as candidates. These candidates are then ranked by similarity in descending order and instantiated using default

argument values to form the final API call sequence.
(2) Reasoning-only: in this baseline, the full API library $\mathcal{A}$ is included in the LLM's context along with user instruction I. The LLM is responsible for selecting appropriate APIs, determining their execution order, and generating fully parameterized API calls based on its own reasoning capabilities, without any external validation or decomposition steps.
(3) Hybrid: To enable a more thorough comparison, we implement the multi-stage pipeline from PPTC (Guo et al. 2024). At each turn t, the system receives the current instruction I_t, the dialogue history, the current document state, and a reference API list. An instruction-understanding module converts this input into an abstract intent representation $\mathcal{U}_t$. A rule-based mapper then selects and parameterizes the most relevant APIs to construct the API sequence $\mathcal{A}_t$.

Metrics. We evaluate performance using both instruction-level accuracy (**iACC**) and session-level accuracy (s**ACC**). In the instruction-level setting, the agent operates on the initial document state associated with each instruction and executes that instruction in isolation. In contrast, session-level evaluation requires the agent to begin from the session's initial state and sequentially complete all instructions in order. In addition, we report the average number of API calls and average token usage required to complete an instruction or an entire session. These metrics serve as indicators of the agent's automation efficiency.

LLMs and generation parameters. We evaluate AutoDW using four recent LLM: two open-source models DeepSeek-V3 (used by default) and Qwen-Plus, and two proprietary models GPT-4.1 and Gemini 2.5 Pro. For embedding-based retrieval, we use text-embedding-v4. During all experiments, the temperature is fixed at 0.1, and all other generation parameters are kept at their default values.

We next present our findings.

Overall Performance Comparison

To demonstrate the effectiveness of AutoDW, we compare its performance against three baseline approaches introduced earlier. The results are summarized in Table 2, and we highlight several key observations:

- The retrieval-only method performs poorly, achieving just 14% iACC and a mere 4.4% sACC. This confirms that simple semantic retrieval is insufficient for complex, sequential workflows. The method lacks awareness of the document state and cannot dynamically schedule actions, making it incapable of resolving inter-instruction dependencies or disambiguating intent in multi-step tasks.
- The reasoning-only method improves over retrieval, reaching approximately 25% sACC. However, the performance remains limited, highlighting the brittleness of relying solely on in-context LLM reasoning for long-horizon tasks. This baseline struggles with long input contexts (commonly referred to as the "lost in the middle" problem) and lacks structured planning or validation, leading to frequent execution failures.
- Hybrid is the strongest among the baselines, achieving 64% iACC but only 35% sACC. Its primary limitation is that it generates a complete plan upfront and lacks any recovery mechanism. Consequently, a single mid-process error can cascade and derail the entire session, resulting in a significant drop in session-level accuracy.
- AutoDW outperforms all baselines by a large margin on both instruction- and session-level metrics. It achieves over 90% iACC, demonstrating highly reliable instruction execution. More notably, it reaches a session-level accuracy (sACC) of 62%, outperforming the next best baseline by a relative gain of 76%. This improvement is achieved with only 26.5% more tokens, a modest increase given the substantial accuracy boost. Additionally, AutoDW's stepwise planning does not result in excessive API calls; its API usage remains comparable to that of other reasoning-based methods.

These results clearly demonstrate the effectiveness and efficiency of our AutoDW framework. The integration of stepwise planning and adaptive rollback successfully maintains long-horizon alignment with user intent and evolving document state, setting a new state-of-the-art for complex document workflow automation.

Robustness Study

We next conduct a robustness study to evaluate the generalization ability of AutoDW under varying circumstances. Specifically, we examine two scenarios: (1) using different backbone LLMs, and (2) operating on tasks of varying difficulty. To this end, we test AutoDW with four LLMs and further stratify the iACC result by difficulty level (*i.e.*, easy, medium, and hard) based on the number of APIs per instruction. The results are presented in Table 3.

Across all tested LLMs, AutoDW demonstrates strong and consistent performance. With Qwen-Plus, it achieves 82.8% iACC and 53.6% sACC, lower than other LLMs. This is likely due to its relatively weaker capabilities compared to other tested models.[2] Nevertheless, it still significantly outperforms the strongest baseline (Hybrid). The other three LLMs yield similar results, achieving over 90% iACC and 62–67% sACC, with comparable API calls and token usage. Interestingly, we observe greater variation across LLMs (excluding Qwen) at the session level (a 5.2% gap in sACC) than at the instruction level (1.7% gap in iACC). This suggests that AutoDW is capable of leveraging the planning and reasoning strengths of more capable LLMs more effectively in complex, multi-step workflows.

For the fine-grained difficulty-based analysis, we observe the expected trend: accuracy decreases as task difficulty increases. However, AutoDW maintains strong performance even on the hard subset. With the three stronger LLMs, it achieves iACC above 86%, which represents only 4.4% drops from the overall instruction-level accuracy. This mild degradation demonstrates that AutoDW is robust to increasing task complexity, ensuring its practical applicability in real-world usage scenarios.

Ablation Study on Rollback Mechanisms

We conclude our experimental analysis with an ablation study to validate the design choices underlying AutoDW. As

[2]https://lmarena.ai/leaderboard

Approach	Instruction-level			Session-level		
	iACC (%)	#APIs	#Tokens (k)	sACC (%)	#APIs	#Tokens
Retrieval-only	13.84	4.82	29.6k	4.40	15.50	98.7k
Reasoning-only	39.93	5.12	31.6k	25.20	34.98	175.4k
Hybrid	64.46	5.30	36.5k	35.20	36.21	225.2k
AutoDW (ours)	**90.33**	5.21	42.8k	**62.00**	34.45	284.9k
AutoDW vs. Hybrid	+40.1%	-1.7%	+17.4%	+76.1%	-4.9%	+26.5%

Table 2: Overall performance comparison with baselines. Tests were repeated for three times and we report the average.

LLM	Instruction-level			Session-level			iACC (%) by difficulty		
	iACC (%)	#APIs	#Tokens	sACC (%)	#APIs	#Tokens	S	M	H
Qwen-Plus	82.82	6.15	40.1k	53.60	38.60	265.4k	**86.34**	83.13	78.99
DeepSeek-v3	90.33	5.21	42.8k	62.00	34.45	284.9k	**94.54**	90.02	86.33
Gemini-2.5-Pro	91.79	4.92	42.7k	**67.20**	32.88	286.0k	**95.08**	93.46	86.84
GPT-4.1	**92.03**	5.14	43.5k	65.20	34.82	294.7k	**95.77**	92.48	87.85

Table 3: Robustness study of AutoDW in terms of backbone LLMs and task difficulty. We classify instruction into Simple, Mediate, and Hard sets if the number of API calls to tackle the instruction is ≤ 3, within $[4, 6]$, and ≥ 6, respectively.

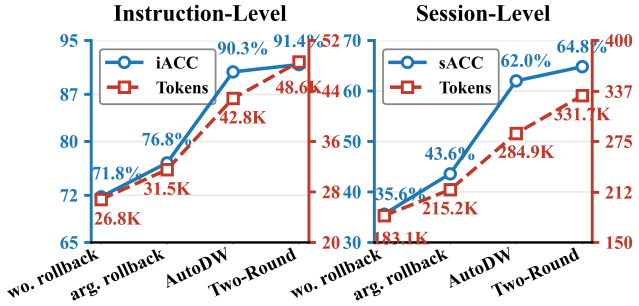

Figure 3: Accuracy and token usage of different rollback strategies. Note that text labels in the figure have been enlarged for better readability.

discussed earlier, AutoDW integrates two key innovations: stepwise planning and adaptive rollback. However, fully ablating stepwise planning is non-trivial, as rollback inherently requires dynamic re-planning. Therefore, in this study, we focus on ablating the rollback mechanism. Specifically, we evaluate three variants of AutoDW: (1) wo. rollback, a version that uses stepwise planning only, (2) arg. rollback which applies a single round of argument-level rollback per action, and (3) two-round, which performs two full rounds of rollback if needed. We compare these variants with the standard AutoDW in terms of instruction-level accuracy (iACC), session-level accuracy (sACC), and token usage. The results are summarized in Fig. 3.

As shown in the figure, both accuracy and token usage increase as more rollback budget is allocated (from left to right). Specifically, the iACC scores for the four configurations are 71.8%, 76.8%, 90.3%, and 91.4%, while the corresponding sACC scores are 35.6%, 43.6%, 62.0%, and 64.8%. Although token usage grows roughly linearly from wo. rollback to two-round, the associated accuracy improvements are non-linear. The most substantial gain occurs between arg. rollback and AutoDW, with an improvement of 13.5% in iACC and 18.4% in sACC. In contrast, further extending to two-round yields only marginal improvements, *i.e.*, an additional 1.1% iACC and 2.8% sACC, representing less than 20% of the previous gain. These results suggest that AutoDW's current rollback strategy, *i.e.*, a single round of dual-level rollback, achieves an effective balance between performance and token cost. It delivers most of the accuracy benefit (*i.e.*, +74% from 35.6% to 62%) while keeping token usage affordable (*i.e.*, +56%), making it a practical design choice for real-world deployment.

Conclusion and Future Work

In this paper, we presented AutoDW, a novel framework that integrates stepwise planning and adaptive dual-level rollback (acting as a dynamic fault-tolerance gate) to automate complex document workflows with robust fault tolerance. AutoDW incrementally plans API actions conditioned on the evolving document state, enabling the system to detect and promptly correct errors before they cascade. To evaluate this capability, we introduced AutoDW, a comprehensive bilingual benchmark of 250 multi-turn sessions with 1,708 human-annotated instructions, designed for rigorous assessment of long-horizon automation where sessions require 15 to 75 API calls. Experimental results demonstrate that AutoDW achieves state-of-the-art performance, outperforming baselines by at least 40% and 76% in instruction and session metrics, respectively. Our ablation study critically reveals the adaptive rollback module alone contributes a 74% relative improvement in session-level success. Looking forward, we envision several promising directions for future research, including exploring more sophisticated planning techniques (e.g., hierarchical or graph-based modeling), extending AutoDW to support a broader range of document types and modalities (e.g., spreadsheets, PDFs), improving rollback efficiency through lightweight validation models, investigating human-in-the-loop strategies to handle instruction ambiguity, and expanding DWBench to include collaborative and multi-agent scenarios.

References

Bang, Y.; Cahyawijaya, S.; Lee, N.; Dai, W.; Su, D.; Wilie, B.; Lovenia, H.; Ji, Z.; Yu, T.; Chung, W.; Do, Q. V.; Xu, Y.; and Fung, P. 2023. A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity. arXiv:2302.04023.

Borji, A. 2023. A Categorical Archive of ChatGPT Failures. arXiv:2302.03494.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D. M.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165.

Comanici, G.; Bieber, E.; Schaekermann, M.; et al. 2025. Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities. arXiv:2507.06261.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, 4171–4186.

Fan, S.; Cong, X.; Fu, Y.; Zhang, Z.; Zhang, S.; Liu, Y.; Wu, Y.; Lin, Y.; Liu, Z.; and Sun, M. 2024. WorkflowLLM: Enhancing Workflow Orchestration Capability of Large Language Models. *arXiv preprint arXiv:2411.05451*.

Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Guo, Y.; Zhang, Z.; Liang, Y.; Zhao, D.; and Duan, N. 2024. PPTC Benchmark: Evaluating Large Language Models for PowerPoint Task Completion.

Hong, S.; Lin, Y.; Liu, B.; Liu, B.; Wu, B.; Zhang, C.; Li, D.; Chen, J.; Zhang, J.; Wang, J.; et al. 2025. Data Interpreter: An LLM Agent for Data Science. In *Findings of the Association for Computational Linguistics: ACL 2025*, 19796–19821.

Ishibashi, Y.; and Nishimura, Y. 2024. Self-organized agents: A llm multi-agent framework toward ultra large-scale code generation and optimization. *arXiv preprint arXiv:2404.02183*.

Karpukhin, V.; Oguz, B.; Min, S.; Lewis, P.; Wu, L.; Edunov, S.; Chen, D.; and Yih, W.-t. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of EMNLP*.

Liang, J. T.; Kumar, A.; Bajpai, Y.; Gulwani, S.; Le, V.; Parnin, C.; Radhakrishna, A.; Tiwari, A.; Murphy-Hill, E.; and Soares, G. 2025. TableTalk: Scaffolding Spreadsheet Development with a Language Agent. arXiv:2502.09787.

Lu, N.; Xie, Q.; Zhang, H.; et al. 2024. Training Overhead Ratio: A Practical Reliability Metric for Large Language Model Training Systems. *arXiv preprint*.

Mathew, M.; Karatzas, D.; and Jawahar, C. V. 2021. DocVQA: A Dataset for VQA on Document Images. arXiv:2007.00398.

Mathur, P.; Siu, A.; Manjunatha, V.; and Sun, T. 2024. DocPilot: Copilot for Automating PDF Edit Workflows in Documents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, 232–246.

Paxton, C.; Raman, V.; Shen, D.; Fox, D.; and Hsu, D. 2019. Prospection: Interpretable plans from language by predicting the future. In *Conference on Robot Learning (CoRL)*.

Rawat, M.; Gupta, A.; Goomer, R.; Bari, A. D.; Gupta, N.; and Pieraccini, R. 2025. Pre-Act: Multi-Step Planning and Reasoning Improves Acting in LLM Agents. arXiv:2505.09970.

Schick, T.; and Schütze, H. 2023. ToolFormer: Language Models Can Teach Themselves to Use Tools. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, 293–304.

Vendrow, J.; Vendrow, E.; Beery, S.; and Madry, A. 2025. Do Large Language Model Benchmarks Test Reliability? *arXiv preprint*.

Wang, D.; Xu, Z.; Ma, Z.; Wang, A. N.; Perot, V.; Liu, S.; Yin, K.; Pfister, T.; Sussman, A.; Ittycheriah, A.; et al. 2024a. DocLLM: A layout-aware generative language model for multimodal document understanding. *arXiv preprint*.

Wang, X.; Jha, S.; and Jha, S. 2024. Incremental Verification of Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 14626–14634.

Wang, Z.; Cui, Y.; Zhong, L.; Zhang, Z.; Yin, D.; Lin, B. Y.; and Shang, J. 2024b. OfficeBench: Benchmarking Language Agents across Multiple Applications for Office Automation. arXiv:2407.19056.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; and Zhou, D. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903.

Xu, Y.; Yang, M.; Liu, L.; Wang, Y.; Cao, F.; and Li, Y. 2020. LayoutLM: Pre-training of Text and Layout for Document Image Understanding. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 1192–1200.

Yang, K.; Liu, Y.; Chaudhary, S.; Fakoor, R.; Chaudhari, P.; Karypis, G.; and Rangwala, H. 2025. AgentOccam: A Simple Yet Strong Baseline for LLM-Based Web Agents. In *The Thirteenth International Conference on Learning Representations*.

Yao, C. 2023. DocXChain: A Powerful Open-Source Toolchain for Document Parsing and Beyond. arXiv:2310.12430.

Yao, S.; Shinn, N.; Razavi, P.; and Narasimhan, K. R. 2025. tau-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *The Thirteenth International Conference on Learning Representations*.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *Proceedings of the International Conference on Learning Representations*.

Zheng, D.; Lapata, M.; and Pan, J. Z. 2024. How Reliable are LLMs as Knowledge Bases? Re-thinking Factuality and Consistency. *arXiv preprint*.

Zhou, S.; Xu, F. F.; Zhu, H.; Zhou, X.; Lo, R.; Sridhar, A.; Cheng, X.; Ou, T.; Bisk, Y.; Fried, D.; Alon, U.; and Neubig, G. 2024. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *The Twelfth International Conference on Learning Representations, ICLR*.

Zou, A.; Yu, W.; Zhang, H.; Ma, K.; Cai, D.; Zhang, Z.; Zhao, H.; and Yu, D. 2024. DOCBENCH: A Benchmark for Evaluating LLM-based Document Reading Systems. arXiv:2407.10701.