Thread: A Logic-Based Data Organization Paradigm for How-To Question Answering with Retrieval Augmented Generation

Anonymous EMNLP submission

Abstract

Current question answering systems leveraging retrieval augmented generation perform well in answering factoid questions but face challenges with non-factoid questions, particularly how-to queries requiring detailed step-by-step instructions and explanations. In this paper, we introduce Thread, a novel data organization paradigm that transforms documents into logic units based on their inter-connectivity. Extensive experiments across open-domain and industrial scenarios demonstrate that Thread outperforms existing data organization paradigms in RAG-based QA systems, significantly improving the handling of how-to questions.

1 Introduction

Dating back to ancient philosophy, Aristotle's Nicomachean Ethics (Crisp, 2014) introduced the concept of "*5Ws and 1H*" questions to comprehensively understand actions and the circumstances surrounding them, where the "5Ws" represent What, Why, When, Where, and Who, and the "1H" stands for How. These questions serve as foundational tools for addressing various aspects of events and human behavior, spanning from philosophical discourse to daily life questions. In our everyday experiences, the 5Ws and 1H questions covers a wide range of questions across different contexts (Kipling, 2018; Wikipedia, 2024). While the 5Ws questions are typically *factoid* questions used to gather factual information about an event or situation[1], the "How" question (or referred to in this paper as "How-To" questions) often delves into processes, methods, or explanations, which may involve more interpretation or analysis rather than straightforward facts, tending to be *non-factoid* questions. Current research has achieved remarkable success in factoid question answering (factoid QA) (Nguyen et al., 2016; Rajpurkar et al., 2018; Jiang et al., 2019; Kwiatkowski et al., 2019; Stelmakh et al., 2022). However, research on non-factoid QAs (Bolotova et al., 2022; Rogers et al., 2023), particularly those involving "How-To" questions, remains relatively underexplored. Existing QA systems often struggle to provide detailed step-by-step answers to how-to questions akin to human comprehension, highlighting a significant gap in current AI capabilities (Krishna et al., 2021; Soleimani et al., 2021).

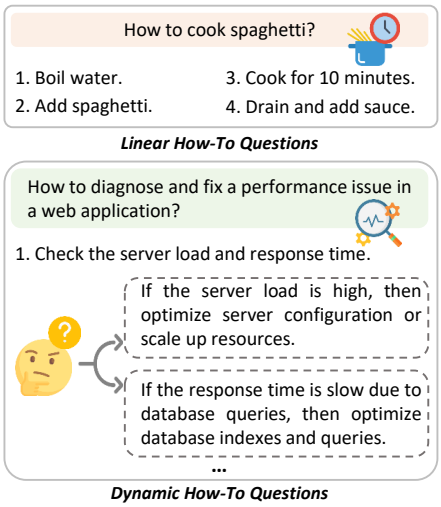

Figure 1: Two types of How-To questions.

In order to better address how-to questions, we explicitly propose the definitions of two types of how-to questions: *Linear How-To Questions* and *Dynamic How-To Questions*. Linear how-to questions involve a fixed sequence of steps that do not require feedback or decision points based on intermediate outcomes. These steps are predefined and must be followed in a specific order to achieve the desired outcome, making them suitable for straightforward, procedural tasks (Merrill, 2012). For example, a question like "*How to cook spaghetti?*" from WikiHow[2] would be answered with a step-by-step procedure as shown in Figure 1. On the other

[1] "Why" question can sometimes delve into explanations and motivations, thus blurring the line between factoid and non-factoid questions.

[2] https://www.wikihow.com/Make-Spaghetti

hand, dynamic how-to questions require a more complex approach where each step may lead to different actions based on the results of previous steps. This type involves conditional decision points and may require iteration, making the process adaptive and responsive to varying circumstances (Newell et al., 1972; Wang and Ruhe, 2007). As shown in Figure 1, initial steps are followed by different paths, where each path requiring different steps and potentially iterative troubleshooting.

The nature of how-to QA inherently follows the decision-making processes central to problem-solving (Polya, 2004) and human learning in cognitive science (Learn, 2000). This requires a structured approach to break down and solve tasks step by step. However, such a natural extension of human decision-making makes how-to questions pose significant challenges for QA systems leveraging Retrieval Augmented Generation (RAG). In the era of Large Language Models (LLMs), RAG represents a powerful technique to enhance the capability of LLMs by incorporating external knowledge. Current RAG systems mainly concentrate on improving the retrieval quality or adaptability (Shao et al., 2023; Trivedi et al., 2023; Jiang et al., 2023; Asai et al., 2023). Despite their success on factoid questions, they still struggle to handle how-to questions, particularly dynamic how-to questions. The root cause of this problem lies in the data organization they use during the retrieval stage. The common practice of splitting documentation into chunks and indexing these chunks for retrieval can disrupt the inherent connections within the text. Although recent works have discussed the granularity of chunks (Chen et al., 2023; Gao et al., 2023), they still adhere to the chunk-based[3] data organization paradigm. This approach can fragment the solution to a how-to question into several chunks, leading to scenarios where the retriever fails to gather all relevant chunks based on semantic or lexical similarity. Consequently, the answers may lack coherence and fail to maintain the logical sequence required for accurately addressing how-to questions.

To facilitate current RAG systems on how-to questions, in this paper we propose a new logic-based data organization paradigm called Thread, which is compatible to RAG systems. Thread explores the inner connections within the documents, splitting the documents into more structured and loosely interconnected logic units (LUs). Given a how-to question, the retriever gets the relevant LUs with its indexed header, and the linker of retrieved LUs will automatically link to other LUs, till obtaining enough information to answer the how-to question. It makes the process of answering how-to questions like "*Pulling on the **thread**, the whole mystery started to unravel like a sweater.*" (Garcia and Stohl, 2011)

[3]"Chunk" refers to a general document splitting paradigm including chunks, sentences, phrases, etc.

2 Related Work

2.1 Data Organization Paradigm in RAG

The data organization process is a critical pre-stage of RAG methods where documents are processed and segmented following certain data organization paradigms. The most common data organization paradigm is splitting documents into retrieval units (Gao et al., 2023). These retrieval units vary in granularity such as phrases, sentences, propositions (Chen et al., 2023), chunks (Kamradt, 2024), etc. Coarser-grained units contain more information but introduce redundant noise, while finer-grained units have lower semantic integrity and often require retrieving more units to gather comprehensive information. However, the chunk-based data organization paradigm ignores the logical and relational connections between chunks, potentially disrupting the inherent logic flow in documents. Another paradigm constructs documents into knowledge graphs (KG), where retrieval units include entities, triplets, etc. (Gaur et al., 2022; Sen et al., 2023; He et al., 2024; Wang et al., 2024; Edge et al., 2024). However, these approaches emphasize semantic/lexical similarities between retrieval units, their success in factoid QA is limited when applied to how-to QA. This limitation arises because how-to QA demands logical connections between retrieval units that extend beyond mere semantic or lexical similarities.

2.2 Information Retrieved by RAG

The effectiveness of RAG methods depends on the generator's ability to utilize retrieved information and the quality and quantity of that information. Insufficient question-relevant information can cause hallucination in LLM-based generators (Li et al., 2023; Zhang et al., 2023), making it crucial to improve the retrieval process. Traditional one-round retrieval methods (Guu et al., 2020; Lewis et al., 2020) often fail to gather all necessary information due to their reliance on the similarity

between query and retrieval units (Gan et al., 2024). Advanced RAG methods use query rewriting and expansion (Peng et al., 2024; Shao et al., 2023; Trivedi et al., 2023; Kim et al., 2023) or iterative retrieval (Shao et al., 2023; Jiang et al., 2023; Asai et al., 2023) to collect more information. However, these approaches still struggle with dynamic how-to questions, which require making next-step decision based on the current retrieved units, unless the current retrieved units contain clues that lead to the next step. The main issue is the lack of connections between retrieval units, which prevents effective retrieval and the gathering of sufficient information for answering how-to questions.

3 Methodology

3.1 Logic Unit: Retrieval Unit of Thread

We propose a new data organization paradigm called ***Thread***. It is a knowledge base composed of discrete but interconnected retrieval units called Logic Units (LUs). Each LU consists of following:

Prerequisite. The prerequisite component acts as an *information supplement*, providing the necessary context to understand the LU. For example, an LU may include domain-specific terminology such as entities or abbreviations. The prerequisite explains these terms and can generate new queries to retrieve LUs with more detailed information. Without this context, passing these LUs to an LLM-based generator could lead to hallucinations. Additionally, the prerequisite can function as an *LU filter*, containing constraints that must be met before the LU is considered in answer generation. This filtering ensures only relevant LUs are retrieved. Tags are a specific example of such prerequisites, quickly filtering out irrelevant LUs in the retrieval stage.

Header. The header summarizes the LU or describes the intention it aims to address, depending on the type of LU (refer to §3.2). For example, the header could be the name of a terminology if LU describes a terminology; if the LU describes actions to resolve a problem, the header describes the intent or the problem LU aims to resolve. The header is used for indexing, similar to traditional RAG processes, and serves as the key for retrieving the LU based on a query.

Body. The body contains the detailed information of the LU, which is the core content fed into the LLM-based generator to generate answers. It includes specific actions or necessary information such as code blocks, detailed instructions, etc. This detailed content helps resolve the query mentioned in the header or provides detailed explanation of the header.

Linker. The linker acts as a bridge between logic units. It is used to generate new queries for the next-round retrieval. In traditional iterative RAG methods, query generation relies on previously retrieval units, but these units often lack direct clues to retrieve new information. In Thread, the linker provides the necessary information for generating these queries. For example, in dynamic how-to questions, the linker specifies multiple possibilities after taking the action in the LU body, guiding the retrieval of the next-step LU. The format of the linker varies depending on the LU type; it can be a query connecting to retrieve other LUs or an entity relationship. The edge of knowledge graph in traditional factoid QA is a special linker enabling navigation between related entities. If no further LUs are connected, the linker remains empty, isolating the current LU.

Meta Data. The meta data includes information about the source document from which the LU is extracted, such as the document title, ID, date, and other relevant details. This meta data is crucial for updating LUs when the source documentation is revised and reprocessed.

3.2 Logic Unit Type

When converting documents into logic units (LUs), there are multiple types of LUs. Below are the common LU types identified in our experiments[4]:

Step. This is the most common LU type for resolving how-to questions. Each LU body represents detailed actions, including code blocks and resolution instructions. The LU prerequisite describes the actions that need to be completed before executing the current actions. The prerequisite is especially crucial for identifying the entry point of a solution. For example, when facing a problem, there exist different ways to resolve it depending on the current situation. The prerequisite serves as a condition in the LU selection stage, filtering out LUs that do not meet the prerequisite.

Terminology. This type provides detailed explanations of domain-specific terminology. For example, terms may share the same name or abbreviation in the LU header but have different meanings in the LU body. The prerequisite in terminology LUs

[4]These LU types are summarized from our practice in experiments with industrial and public datasets. There may be additional LU types depending on the specific scenario.

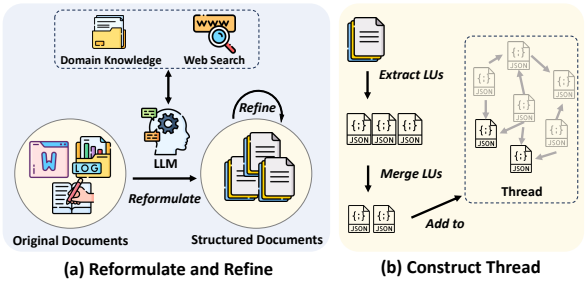

Figure 2: The process of Thread construction.

includes scenarios where the terminology typically appears. Terminology LUs generally have empty linkers unless they refer to extended terminology that depending on it.

FAQ. This type provides frequently asked questions, supplementing the knowledge base. These LUs are typically isolated, with the LU body offering solutions through sequential steps that address linear how-to questions not reliant on dynamic states. They save time by avoiding the need for sequentially retrieving LUs for common questions.

Appendix. This type provides additional information relevant to the scenario of LUs, such as examples, background, lookup tables, etc. These LUs serve as supplementary knowledge for LLMs when generating responses or executable plans.

3.3 Thread: LU-based Knowledge Base

In practice, documentation is often unstructured and varies in format and style. Our approach to converting documentation into Thread involves a two-stage process to obtain LUs. We believe that with advancements in LLMs, this process could be streamlined into a single stage, enabling high-fidelity LU conversion with minimal hallucination.

Documentation Reformulation (Optional). This stage is optional, depending on the quality of the documentation. For example, in software engineering, Troubleshooting Guides (TSGs) often lack readability and detail, negatively impacting productivity and service health (Shetty et al., 2022). Due to varying document styles, where some are clearly outlined and others are disordered, we avoid directly extracting LUs from the original documents. Instead, we first reformulate these documents into structured formats. Leveraging LLMs for this task, we enhance the LLMs' understanding in domain by providing search capabilities and domain-specific context. This is followed by a refinement step to prevent overlooking details or hallucinating information. Figure 2(a) shows the reformulation stage. The reformulation stage is unnecessary for well-written documents like product help docs, which typically follow a linear how-to format. Table 8 in the Appendix shows an example of this process with prompts.

LU Extraction and Merge. After reformulation, multiple LUs of varying types can be extracted from a single structured document (shown in Figure 2(b)). Unlike chunk-based data organization commonly with fixed chunk sizes, LU granularity depends on content. For example, solutions to linear how-to questions typically form a single path from start to completion, with interconnected steps and no multiple execution outcomes encapsulated in one LU, such as an FAQ LU. However, for dynamic how-to questions with multiple possible outcomes, it is better to have one step per LU (Step LU), with Linkers navigating to the next LUs. Note that in dynamic how-to questions, not every step has multiple execution outcomes. If only one next step exists, the LUs can be merged to include both current and subsequent steps. Additionally, LUs with similar Headers and Bodies should be merged, extending the Prerequisite and Linker.

LU update. In industry, documentation is often updated with each product version release. When this happens, we redo the above steps for the updated documentation, identifying LUs in Thread with their Meta Data and replacing outdated LUs.

As we extract and merge LUs, the collection of LUs from all documents forms the knowledge base, *i.e.*, Thread. This LU-based knowledge base serves as an essential component compatible to current RAG-based QA system, and even making it possible for automation.

3.4 Integrate Thread with QA System

To demonstrate how Thread works, we use dynamic how-to questions as an example. Figure 3 shows the QA system incorporating our Thread data organization paradigm. The *Retriever* and *LLM-based Generator* are components from the original RAG-based QA system. LUs are indexed by their Headers. When an initial how-to question is submitted, the Retriever identifies the top-K most relevant LUs based on query-Header similarity. The Selector then checks the prerequisites of these LUs and filters out those that do not meet the current prerequisites, derived from the initial question or any available chat history. If no current prerequisite is provided, the system can prompt the user, *e.g.*, "Before doing ..., have you tried ...?",

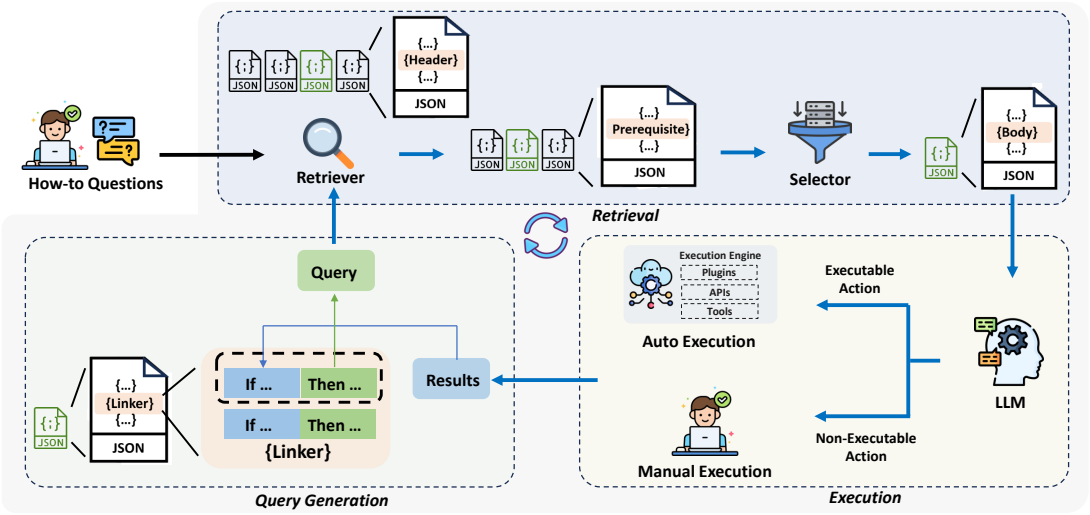

Figure 3: The QA system integrated with Thread. It retrieves relevant LUs based on query-Header similarity and filters out LUs that do not meet the current prerequisites. The selected LUs are passed to LLMs to generate actions for execution. After execution, the Linker matches results and generates a new query for the next retrieval iteration.

to obtain the current prerequisite for LU filtering. After selection, the Body of the LUs is fed into the LLM-based generator to produce an answer. If an execution engine is available, actions can be executed automatically; otherwise, the answer/action is presented to the user for manual execution. Once the action is executed, the Linker matches one of the possible outcomes and generates a new query for the next retrieval round.

Unlike traditional QA systems, the QA system with Thread can potentially be fully or semi-automated if an execution engine is present. Compared to manually designed automation pipelines, Thread enables greater automation and flexibility where updating LUs will automatically update the semi-automated system (see §3.3).

4 Experimental Setup

4.1 Scenarios and Datasets

We evaluate Thread on two open-domain scenarios: Web Navigation (Mind2web (Deng et al., 2023)) and Wikipedia Instructions (WikiHow (Koupaee and Wang, 2018)), and one industrial scenario: Incident Mitigation (IcM (Shetty et al., 2022; An et al., 2024)). Table 7 in Appendix B provides one example of each dataset to better demonstrate the linear and dynamic how-to question. More details about datasets can be found in Appendix D, E, F.

Web Navigation. Mind2web (Deng et al., 2023) is a dataset designed for web agents to perform complex tasks on real-world websites based on language instructions. Each task is a *dynamic how-to question*, with multiple possible outcomes depending on the state of the executed actions.

Wikipedia Instructions. WikiHow[5] is a platform containing numerous articles that offer step-by-step guidance on various procedural tasks, ranging from entertainment to technology. Each article is typically titled with "How to" and includes a brief task description, followed by detailed steps. We use this dataset to demonstrate Thread's performance on *linear how-to questions*.

Incident Mitigation. Incident Mitigation (Shetty et al., 2022; An et al., 2024; Jiang et al., 2024) is essential for operating large-scale cloud services, where engineers use Troubleshooting Guides (TSGs) to address incidents. Each step in incident mitigation can yield different outcomes depending on the state, making it suitable for testing Thread on *dynamic how-to questions*. Unlike the other open-domain datasets, we perform a human evaluation involving twenty on-call engineers (OCEs)[6] responsible for incident mitigation. We collect five incidents[7], classified into two simple and three hard ones based on their mitigation steps in history. Each OCE is tasked with mitigating all five incidents, using only one method per incident to avoid familiarity bias. The QA system initiates automated mitigation for each incident; if it encounters a failure at any step, an OCE intervenes to address the issue before the system resumes automated procedures.

[5] https://www.wikihow.com

[6] Varying across new-hire and experienced OCEs.

[7] Mitigating one incident costs each OCE around 30 minutes to 1 hour, and we collect five incidents to ensure consistency, engagement, and ethical treatment.

4.2 Documents for Retrieval

Since the Mind2web datasets lack relevant documents, we created documents for retrieval to fit the RAG-based QA system. Assuming the same website has help docs applicable to all tasks in the website, we utilized the "Cross-Task" test set and selected examples from the training set to create informative documents for each website, following Wang et al. (2023). This allows the generation of documents for retrieval to effectively accomplish tasks within the RAG-based QA system.

For the WikiHow dataset, we used publicly available Windows Office Support Docs[8] as retrieval documents. We selected around 100 tasks from the WikiHow dataset tagged with Microsoft products like Word, PowerPoint, and Teams, each containing 10 to 40 steps related to Windows operations.

For the IcM dataset, we collected 56 TSGs as documents from an enterprise-level engineering team responsible for a large-scale cloud platform. The selected incidents in the IcM dataset can be resolved using knowledge from these TSGs.

Table 1 shows the dataset statistics. Note that in IcM dataset, one OCE's data was contaminated during the experiment, so we removed that OCE's data. It results in 19 OCEs with 95 tasks.

Dataset	#Docs	#Tasks	#Steps	#Chunks	#LUs
Mind2web	490	252	2094	6210	1089
WikiHow	97	97	2140	4225	774
IcM	56	95	323	413	378

Table 1: The statistics of datasets, including the number of documents, tasks, steps, and derived chunks and LUs.

4.3 Baselines

In the Mind2web dataset, previous work has not treated it as how-to questions. State-of-the-art methods like SYNASE (Zheng et al., 2023) and MINDACT (Deng et al., 2023) either use In-Context Learning (ICL), providing few-shot demonstrations, or Supervised Learning (SL) to finetune a model on the training set. To ensure a fair comparison with our LLM endpoints[9], we re-implemented the MINDACT experiments with the same demonstrations and included chat history as extra context. In our paper, we treat the Mind2web dataset as dynamic how-to questions and use our RAG-based system to solve these tasks. For comparison, we use doc-based (providing the entire document) and chunk-based data organization paradigms as baselines against Thread. This RAG-based QA system with different data organization paradigms also serves as baselines for the WikiHow and IcM datasets.

[8] https://github.com/MicrosoftDocs/OfficeDocs-Support

[9] We use GPT-3.5 and GPT-4 with version 1106-preview.

4.4 Evaluation Metrics

For the Mind2web dataset, each task is treated as a multi-choice question. We adopt the evaluation metrics used in (Deng et al., 2023), which include: *Element Accuracy (Ele. Acc)* to compare the chosen HTML element with all provided elements; *Operation F1 (Op. F1)* to calculate the token-level F1 score for predicted operations such as "CLICK", "TYPE IN", etc.; *Step Success Rate (Step SR)*, where a step is successful if both the selected element and predicted operation are accurate; and *Success Rate (SR)*, where a task is successful only if all steps are successful.

For the WikiHow dataset, which contains ground truth steps, we leverage LLMs to extract "Action Items" from each ground truth step and generated step, and we use the following metrics: *Precision* ($P = \frac{\#matched\_items}{\#total\_generated\_items}$); Recall ($R = \frac{\#matched\_items}{\#total\_groundtruth\_items}$); *F1*; and *Success Rate (SR)* to assess if the generated steps can successfully complete the task, using LLMs to evaluate (Table 16 shows the evaluation prompt).

For the IcM dataset, which involves task execution, we perform evaluations with OCEs (refer to §4.1) using five metrics: *Success Rate (SR)* indicating the percentage of incidents mitigated automatically by the system without human intervention; *Step Success Rate (Step SR)* representing the percentage of successful steps out of all task steps; *Pre-Failure Step Success Rate (P.F. Step SR)* representing the percentage of successful steps before the first failure; *Human Intervention (HI)* measuring the percentage of steps requiring human intervention; and *Average Turns (Turns)* to measure the average interaction turns between OCEs and the system during incident mitigation.

5 Experimental Results

Web Navigation. The overall performance on Mind2web is shown in Table 2, where we compare our method with baselines and both doc-based and chunk-based RAG methods. We observe that providing informative documents (regardless of the data organization paradigm) within RAG methods significantly improves performance. RAG methods

Method	Model	Paradigm	Mind2Web Cross-Task			
			Ele. Acc	Op. F1	Step SR	SR
SYNASE (2023)	w/ GPT-3.5*	ICL	34.00	-	30.60	2.40
MINDACT (2023)	w/ GPT-3.5	ICL	40.69	49.66	33.91	1.59
	w/GPT-4	ICL	62.80	60.37	51.81	10.32
	w/ Flan-T5$_{XL}$*	SL	55.10	**75.70**	52.00	5.20
RAG	w/ GPT-4	Chunk	64.23	65.96	58.45	8.73
	w/ GPT-4	Doc	63.80	65.89	58.36	11.51
	w/ GPT-4	Thread	**68.29**	69.53	**61.94**	**12.30**

Table 2: Experiment results on Mind2web. "*" represents taking results from the original paper.

outperform ICL and SL methods. By incorporating external documents, the doc-based RAG method achieves performance comparable to the best results of MINDACT. Notably, our Thread paradigm is the best among RAG methods, showing improvements of 4.06% in Ele. Acc, 3.49% in Step SR and 3.57% in SR. Note that MINDACT-SL gets the highest Op. F1 due to label distribution imbalance, leading the model to favor generating the most frequent operations.

Wikipedia Instructions. Table 3 presents the performance of various data organization paradigms on WikiHow tasks, experimenting with both single-turn and multi-turn interactions. In the single-turn setting, where the QA system generates the entire plan in one interaction, our Thread-based method outperforms the doc-based method which provides the entire relevant document, indicating that Thread offers information with less redundancy. The chunk-based method is not included as it failed to retrieve enough relevant chunks using the question as the query.

In the multi-turn setting, the QA system performs iterative retrieval, which shows superior performance compared to single-turn. This indicates the advantage of a step-by-step approach in handling how-to questions. Notably, the SR improves significantly from 58.76% to 68.04% in doc-based paradigm and from 63.91% to 72.16% in Thread paradigm. Within multi-turn setting, chunk-based method divides documents into chunks which disrupts internal connections, leading to a low SR of 20.62%. In contrast, both doc-based and Thread-based paradigms perform significantly better. Our Thread paradigm excels across all metrics, achieving the highest SR of 72.16%, highlighting its effectiveness in maintaining and modeling the connections between steps in linear how-to questions.

Incident Mitigation. Figure 4 illustrates the advantages of Thread over other paradigms when addressing complex dynamic how-to questions. Both chunk-based and doc-based RAG methods exhibit comparable ineffectiveness in incident mitigation. In contrast, our Thread paradigm achieves the best score across all metrics, particularly achieving a significant increase in SR from 21.02% to 33.33%. Furthermore, *P.F. Step SR* indicates our method's capability to handle significantly more steps before encountering a failure, thus reducing the need for human intervention. Consequently, our method achieves the highest Step SR and requires the fewest interaction Turns to mitigate both simple and complex incidents.

Paradigm	WikiHow			
	SR	P	R	F1
Single-Turn				
Doc	58.76	77.71	57.93	66.37
Thread	**63.91**	**83.43**	**71.48**	**76.99**
Multi-Turn				
Chunk	20.62	52.95	25.54	34.45
Doc	68.04	87.10	70.65	78.02
Thread	**72.16**	**89.77**	**73.36**	**80.74**

Table 3: Experiment results on WikiHow with different interaction manners and data organization paradigms.

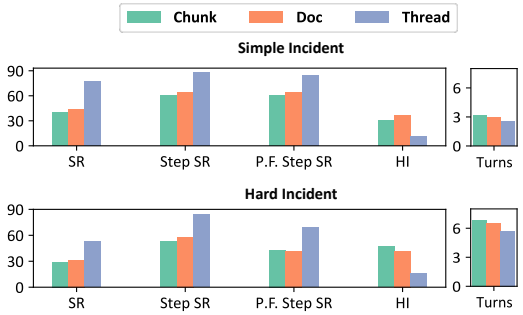

Figure 4: Experiment results on simple and hard incidents of IcM dataset with OCEs.

5.1 Ablation on QA System Settings

This section presents an ablation study on the key settings of a RAG-based QA system. While retriever and generator variants have been explored

in other research (Gao et al., 2023), we use text-embedding-ada-002 (OpenAI, 2022) for the retriever and GPT-4 as the LLM-based generator. The ablation study is performed in Mind2web dataset.

Multi-turn Interaction. As shown in Table 3, the multi-turn setting is better than single-turn setting in answering how-to questions. We use multi-turn setting for the other scenarios in this paper.

Chat History. The chat history helps the system to check previous actions and results, in order to help the system make better decisions. Without chat history, the performance drops as shown in Appendix E.3. We include chat history for all RAG-based methods in the experiments.

Retrieval Units Selector. As aforementioned in §3.4 and Figure 3, the Selector picks the most relevant retrieval units from the top-K retrieved units. Table 4 shows the ablation of the retrieval unit selector. We compare chunk and LU as retrieval units, finding that chunk selection degrades the performance of chunk-based RAG, reducing Ele. Acc by 4.29% and Op.F1 by 3.38%. Conversely, the selector improves all metrics in the Thread-based RAG method. Unlike LU selection, which filters out irrelevant LUs with prerequisites, chunk selection does not consider connections between chunks and may filter out relevant chunks. Therefore, we enable the selector only for Thread-based method.

Paradigm	Ele. Acc	Op. F1	Step SR
Chunk	59.94	62.58	54.39
$w/o.$ chunk selection	**64.23**	**65.96**	**58.45**
Thread	**68.29**	**69.53**	**61.94**
$w/o.$ LU selection	67.05	68.43	60.79

Table 4: Ablation study of retrieval unit selection.

Multi-Agent System. We investigate the multi-agent design in QA system. As detailed in Appendix E.4, the multi-agent RAG system (Wu et al., 2023; An et al., 2024) outperforms single-agent systems, where only the LLM-based generator acts as the agent, in resolving how-to questions.

5.2 Analysis of Data Organization Paradigms

We compare various data organization paradigms within RAG, implementing Semantic Chunking (Kamradt, 2024) and Proposition (Chen et al., 2023) in addition to recursive chunking (Splitter, 2023) (chunk-based) and entire document (doc-based) approaches. As shown in Table 5, our Thread paradigm outperforms the other paradigms across all metrics. Although Semantic and Proposition methods use LLMs to merge semantically similar sentences, they fail to adequately address logical connections, resulting in poor performance. Additionally, our system processes the smallest token length of retrieval units yet achieves the highest performance, underscoring our approach's efficiency and effectiveness.

Paradigm	Ele. Acc	Op. F1	Step SR	#Tokens in RU
Doc	63.80	65.89	58.36	663.84
Recursive	64.23	65.96	58.45	695.77
Semantic	65.14	67.30	59.93	1337.16
Proposition	62.37	64.78	56.78	790.14
$\text{Thread}_{\text{w/o.}}$	67.05	68.43	60.79	772.67
Thread	**68.29**	**69.53**	**61.94**	**157.10**

Table 5: Analysis of data organization paradigms. $\text{Thread}_{\text{w/o.}}$ represents Thread without LU selector.

5.3 Document Formats in LU Extraction

As mentioned in §4.2, we tested our LU extraction method in accommodating varying document structures. We generated different document formats (detailed in Appendix A, E.2), including structured markdown, hierarchical guidelines, tabular checklists, and narrative documents. Table 6 shows that our Thread paradigm effectively organizes these formats, consistently outperforming the chunk-based paradigm across all metrics. Our Thread improves Ele. Acc by up to 9.61%, Op. F1 by up to 8.43%, and Step SR by up to 8.51%. Notably, our Thread achieves the highest performance with structured documents, as this format helps construct a higher-quality knowledge base.

Format	Paradigm	Ele. Acc	Op. F1	Step SR
Structured	Chunk	64.23	65.96	58.45
	Thread	**68.29**	**69.53**	**61.94**
Hierarchical	Chunk	60.60	63.46	55.06
	Thread	66.57	67.89	60.08
Tabular	Chunk	56.30	59.26	51.43
	Thread	65.71	67.69	59.55
Narrative	Chunk	56.63	60.39	51.66
	Thread	66.24	68.22	60.17

Table 6: Analysis of different document formats.

6 Conclusion

In this paper, we address the overlooked category of how-to questions within existing QA systems. We propose Thread, a novel data organization paradigm to capture logic connections with documents. Thread is compatible and integrated to current RAG-based QA system. Experimental results demonstrate Thread's effectiveness in handling how-to questions, surpassing other data organization paradigms in performance.

Limitations

This work choose three scenarios to design how-to questions, and evaluate its effectiveness of Thread on handling both linear how-to questions and dynamic how-to questions. However, the proposed paradigm still has some limitations for future directions. On the one hand, since our logic knowledge base can coexist with the original chunk knowledge base, we do not extend our method to test factoid questions, like multi-hop questions, long-form questions and so on. On the other hand, the experiments primarily rely on OpenAI's backbone models. Future studies should include evaluations using other LLMs like LLaMA and retrievers such as Contriver to better validate the effectiveness of our paradigm. Additionally, while extracting logic units involves initial costs in calling LLMs, our approach is a one-time procedure. Once the knowledge base is constructed, it becomes advantageous for industrial applications, particularly in terms of subsequent updates and maintenance.

Ethics Statement

All the experiments are conducted on existing dataset or resources that collected by ourselves or from internal sources. We keep fair and honest in our analysis of experimental results. As the scenario of IcM need human evaluation, we ensure that there is no bias in selecting OCEs and ensure the randomness of the sampling.

References

Kaikai An, Fangkai Yang, Liqun Li, Zhixing Ren, Hao Huang, Lu Wang, Pu Zhao, Yu Kang, Hua Ding, Qingwei Lin, et al. 2024. Nissist: An incident mitigation copilot based on troubleshooting guides. *arXiv preprint arXiv:2402.17531*.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *CoRR*, abs/2310.11511.

Valeria Bolotova, Vladislav Blinov, Falk Scholer, W. Bruce Croft, and Mark Sanderson. 2022. A non-factoid question-answering taxonomy. In *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, pages 1196–1207. ACM.

Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. 2023. Dense X retrieval: What retrieval granularity should we use? *CoRR*, abs/2312.06648.

Roger Crisp. 2014. *Aristotle: nicomachean ethics*. Cambridge University Press.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samual Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

Chunjing Gan, Dan Yang, Binbin Hu, Hanxiao Zhang, Siyuan Li, Ziqi Liu, Yue Shen, Lin Ju, Zhiqiang Zhang, Jinjie Gu, et al. 2024. Similarity is not all you need: Endowing retrieval augmented generation with multi layered thoughts. *arXiv preprint arXiv:2405.19893*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey.

Kami Garcia and Margaret Stohl. 2011. *Beautiful Chaos*. Hachette UK.

Manas Gaur, Kalpa Gunaratna, Vijay Srinivasan, and Hongxia Jin. 2022. Iseeq: Information seeking question generation using dynamic meta-information retrieval and knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10672–10680.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Retrieval augmented language model pre-training. In *Proceedings of the*

37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, volume 119 of *Proceedings of Machine Learning Research*, pages 3929–3938. PMLR.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *arXiv preprint arXiv:2402.07630*.

Kelvin Jiang, Dekun Wu, and Hui Jiang. 2019. Freebaseqa: A new factoid QA data set matching trivia-style question-answer pairs with freebase. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 318–323. Association for Computational Linguistics.

Yuxuan Jiang, Chaoyun Zhang, Shilin He, Zhihao Yang, Minghua Ma, Si Qin, Yu Kang, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2024. Xpert: Empowering incident management with query recommendations via large language models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*, pages 92:1–92:13. ACM.

Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 7969–7992. Association for Computational Linguistics.

Greg Kamradt. 2024. The 5 levels of text splitting for retrieval.

Gangwoo Kim, Sungdong Kim, Byeongguk Jeon, Joonsuk Park, and Jaewoo Kang. 2023. Tree of clarifications: Answering ambiguous questions with retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 996–1009. Association for Computational Linguistics.

Rudyard Kipling. 2018. *I keep six honest serving-men*. LibriVox.

Mahnaz Koupaee and William Yang Wang. 2018. Wikihow: A large scale text summarization dataset. *CoRR*, abs/1810.09305.

Kalpesh Krishna, Aurko Roy, and Mohit Iyyer. 2021. Hurdles to progress in long-form question answering. pages 4940–4957.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur P. Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: a benchmark for question answering research. *Trans. Assoc. Comput. Linguistics*, 7:452–466.

How People Learn. 2000. Brain, mind, experience, and school. *Committee on Developments in the Science of Learning*.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Junyi Li, Xiaoxue Cheng, Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2023. Halueval: A large-scale hallucination evaluation benchmark for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 6449–6464. Association for Computational Linguistics.

M David Merrill. 2012. *First principles of instruction*. John Wiley & Sons.

Allen Newell, Herbert Alexander Simon, et al. 1972. *Human problem solving*, volume 104. Prentice-hall Englewood Cliffs, NJ.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A human generated machine reading comprehension dataset. 1773.

OpenAI. 2022. https://openai.com/index/new-and-improved-embedding-model/.

Wenjun Peng, Guiyang Li, Yue Jiang, Zilong Wang, Dan Ou, Xiaoyi Zeng, Derong Xu, Tong Xu, and Enhong Chen. 2024. Large language model based long-tail query rewriting in taobao search. pages 20–28.

George Polya. 2004. *How to solve it: A new aspect of mathematical method*. 246. Princeton university press.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. pages 784–789.

Anna Rogers, Matt Gardner, and Isabelle Augenstein. 2023. QA dataset explosion: A taxonomy of NLP resources for question answering and reading comprehension. *ACM Comput. Surv.*, 55(10):197:1–197:45.

Priyanka Sen, Sandeep Mavadia, and Amir Saffari. 2023. Knowledge graph-augmented language models for complex question answering. In *Proceedings of the 1st Workshop on Natural Language Reasoning and Structured Explanations (NLRSE)*, pages 1–8, Toronto, Canada. Association for Computational Linguistics.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 9248–9274. Association for Computational Linguistics.

Manish Shetty, Chetan Bansal, Sai Pramod Upadhyayula, Arjun Radhakrishna, and Anurag Gupta. 2022. Autotsg: learning and synthesis for incident troubleshooting. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, pages 1477–1488. ACM.

Amir Soleimani, Christof Monz, and Marcel Worring. 2021. Nlquad: A non-factoid long question answering data set. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pages 1245–1255. Association for Computational Linguistics.

Text Splitter. 2023. Recursively split by character.

Ivan Stelmakh, Yi Luan, Bhuwan Dhingra, and Ming-Wei Chang. 2022. ASQA: factoid questions meet long-form answers. pages 8273–8288.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 10014–10037. Association for Computational Linguistics.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023. Improving text embeddings with large language models. *arXiv preprint arXiv:2401.00368*.

Yingxu Wang and Guenther Ruhe. 2007. The cognitive process of decision making. *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)*, 1(2):73–85.

Yu Wang, Nedim Lipka, Ryan A Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. 2024. Knowledge graph prompting for multi-document question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19206–19214.

Wikipedia. 2024. Five ws.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen LLM applications via multi-agent conversation framework. *CoRR*, abs/2308.08155.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. 2023. Siren's song in the AI ocean: A survey on hallucination in large language models. *CoRR*, abs/2309.01219.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2023. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations*.

A Details about Current Data Organization Paradigms

From Gao et al. (2023), current data organization paradigms can be categorized into phrases, sentences, propositions, chunks and so on. In our paper, we choose chunks [10] and propositions to compare with our proposed Thread[11].

Recursive Chunk. This chunking method splits the original documents using a list of separators, then reassembles them according to specified chunk sizes and overlap sizes. In our experiment, we use different chunk sizes for each dataset: 1000 for Mind2Web, 2000 for IcM, and 300 for WikiHow. The chunk overlap sizes also vary: 50 for Mind2Web, 100 for IcM, and 30 for WikiHow.

Entire Document. This method sends the entire document directly into the model, constrained by the document's length and structure.

Semantic Chunk. Kamradt (2024) proposes splitting chunks based on semantic similarity. The hypothesis is that semantically similar chunks should be grouped together. By comparing the semantic similarity between adjacent sentences, the method identifies "break points". If the similarity in the embedding space exceeds a certain threshold, it marks the start of a new semantic chunk.

Agentic Chunk (Proposition). Chen et al. (2023)[12] introduces the concept of the Proposition Paradigm, which involves extracting independent propositions from original documents. The Agentic Chunk method is based on this paradigm. It first splits the documents into paragraphs, then extracts propositions from each paragraph, at last merges similar propositions into chunks.

In our experiment, we set the temperature of LLMs to 0 and top_p to 1 for results reproduction.

B Examples of Each Dataset

Table 7 shows the examples of each dataset.

C Example of Logic Unit

Table 8 shows an example to demonstrate the reformulated process of document and its corresponding logic unit.

[10] We use the the implementation by LangChain https://python.langchain.com/v0.2/

[11] Note: For chunks, we retrieve the top-5 at each time, and for documents, we only retrieve the top-1.

[12] For proposition paradigm, we use agentic chunker since the input token of Flan-T5 is limited to 512, https://github.com/FullStackRetrieval-com/RetrievalTutorials.

D Details about Incident Mitigation

We take the scenario of incident mitigation to show the instructions we use to construct our knowledge base, including reformulation, code template extraction and selection.

D.1 Document Reformulation Instruction

Table 9 shows the instruction we use to reformulate document.

D.2 Code Template Instruction

Table 10 shows the instruction we use to extract code template and default parameters from original code.

D.3 Logic Unit Selection Instruction

Table 11 shows the instruction we use to select the most suitable logic unit from a list of candidates.

E Details about Mind2web

We show the details about the document generation instruction we use, the different formats of document and examples we generate.

E.1 Document Generation Instruction

Table 12 shows the instruction we use to generate specific format of document for Mind2web dataset.

E.2 Different formats of Document

Table 13 lists the formats we pre-define for Mind2web dataset.

E.3 Ablations Study of Chat History

Table 14 reveals that incorporating historical steps enhances the performance of LLMs, with an improvement of 5.83 % in Ele. Acc, suggesting that previous steps can help the system make more accurate decisions. So we incorporate historical steps in all our experiments.

E.4 Ablations Study of Multi-Agent System

We explore the impact of more complex RAG systems on their performance. We adapt the multi-agent system proposed by Wu et al. (2023); An et al. (2024), which is more intricate, to execute Mind2web tasks. In the multi-agent system, multiple LLM-based agents such as the Selector, Action Planner, and Query Generator collaborate, enhancing performance on complex how-to questions. As

Dataset	Example
Mind2web (Dynamic)	<html> ... </html> Based on the HTML webpage above, try to complete the following task Task: Book the lowest-priced and quickest flight for 5 adults and 1 child on May 20 from Mumbai to any airport near Washington. Previous actions: None What should be the next action? Please select from the following choices (If the correct action is not in the page above, please select A. 'None of the above'): A. None of the above B. <input radio triptype roundtrip true /> <label> C. <label id=1> Search flights one way One D. <a id=2> Celebrate World Wish Day Support E. <h2 id=3> Help F. C. Action: CLICK
WikiHow (Linear)	"Problem": "How to Add Captions to Tables in Microsoft Word", "Solution Steps": ["Select the table to which you want to add a caption.", "Using your mouse, click and drag over the entire table to select it.", "Right-click (or ctrl-click) the table and select Insert Caption.", "Enter your caption.", "Type the caption for this table into the \"Caption\" field.", "Select a caption label.", "Customize your caption numbers (optional).", "Choose where to place your caption.", "Click the \"Position\" drop-down menu, and choose whether to place the caption above or below the table.", "Click OK to add your caption to the table.", "Format your captions."]
IcM (Dynamic)	### Step 1:Step 1: Check Pull Task Execution From the Cluster The direct impact of connection failure is pull task execution will not work. If Service A can continue to pull from Service B, then the incident can be dismissed as false alarm, the feature owner can investigate further to see why Echo fails. This can be visualized by pull task count over time in the last 8 hours in the following query: *** Disregard the last data point, if the data point is always above zero, then consider the alert as false alarm. If the chart sometimes drops to zero one hour ago and the number is low in general (for instance less than 20), it means the customer traffic in the cluster is low. In this case, observe for a longer period of time. If the data point is zero consistently in the past 30 minutes, then it is a real problem, and please Check if Other Clusters In the Region are Impacted. Otherwise, continue to observe since Service A is pulling Service B just fine.

Table 7: Examples of each dataset. For mind2web, although the test set has fixed options for each step, there are different execution methods for the same task on each website, so it is essentially dynamic.

Trouble Shooting Guide: How to Investigate Service A-To-Service B Connection?	
Original	### Step 0: Determine the Region and Cluster Name The region and cluster name can be found in the incident title. ### Step 1: Check Pull Task Execution From the Cluster The direct impact of connection failure is pull task execution will not work. If Service A can continue to pull from Service B, then the incident can be dismissed as false alarm, the feature owner can investigate further to see why Echo fails. This can be visualized by pull task count over time in the last 8 hours in the following query: *** Disregard the last data point, if the data point is always above zero, then consider the alert as false alarm. If the chart sometimes drops to zero one hour ago and the number is low in general (for instance less than 20), it means the customer traffic in the cluster is low. In this case, observe for a longer period of time. If the data point is zero consistently in the past 30 minutes, then it is a real problem, and please Check if Other Clusters In the Region are Impacted. Otherwise, continue to observe since Service A is pulling Service B just fine. ...
Reformulated	## 1.Check Pull Task Execution From the Cluster. ### Prerequisite The region and cluster name can be found in the incident title. ### Header Check Pull Task Execution From the Cluster ### Body Run the following query to check pull task execution from the cluster (please use the cluster name from the previous step) *** ### Linker - If the data point is always above zero, then consider the alert as false alarm.[MITIGATE] - If the chart sometimes drops to zero one hour ago and the number is low in general, it means the customer traffic in the cluster is low. In this case, observe for a longer period of time.[MITIGATE] - If the data point is zero consistently in the past 30 minutes, then it is a real problem, and please Check if Other Clusters In the Region are Impacted.[CONTINUE] - Otherwise, continue to observe since Service A is pulling Service B just fine.[MITIGATE] ...

Logic Unit	{
	"#type#": "step",
	"#meta data#": {
	"#title#": "How to Investigate Service A-To-Service B Connection",
	"#id#: "",
	"#date#": ""
	},
	"#prerequisite#": "The region and cluster name are given.",
	"#header#": "Check Pull Task Execution From the Cluster.",
	"#body#": "Run the following query to check pull task execution from the cluster (please use the cluster name from the previous step):***",
	"#linker#": "If the data point is always above zero, then consider the alert as false alarm.[MITIGATE] If the chart sometimes drops to zero one hour ago and the number is low in general, it means the customer traffic in the cluster is low. In this case, observe for a longer period of time.[MITIGATE] If the data point is zero consistently in the past 30 minutes, then it is a real problem, and please Check if Other Clusters In the Region are Impacted.[CONTINUE] Otherwise, continue to observe since Service A is pulling Service B just fine.[MITIGATE]",
	"#default_parameters#": {
	"<TIME>": "",
	"<CLUSTER NAME>": ""
	}
	}
	
	...

Table 8: An example of reformulated TSG and its corresponding Logic Unit using our paradigm.

evidenced in Table 15, there is a noticeable improvement in the overall performance, with the Success Rate on Mind2web tasks rising by 3.36%. This increment suggests that the deployment of multi-agent systems can substantially improve the capability to manage tasks of greater complexity.

F Details about Wikihow

Table 16 shows the instruction about how to evaluate the result of Wikihow, including extraction of action items and assessment of success.

G Details about RAG System

We take the scenario of Mind2web to show the instructions we used in our RAG system.

G.1 Instruction of Baselines

Table 17 shows the instruction we use in our RAG system of baselines.

G.2 Instruction of Thread Paradigm

Table 18 shows the instruction we use in our RAG system utilizing Thread paradigm, which is different from baselines as it requires to point out the query for next time retrieval.

[System]
You are a helpful troubleshooting guide assistant that helps the user to formulate the manual unstructured troubleshooting guide <TSG> into structured one. The <TSG> is in markdown format, with the first level header describing the incident or problem, and the following each second level header providing information related to the incident or problem.
Each second level subsection can be categorized into the following types: Terminology, FAQ, Step and Appendix. Your reformulation should be strictly comply the following definition:
- Terminology: firstly, it should be the relationship or connection between terminology about the incident, if not, is can be the explanation or concept of the incident. Sometimes it should extract and summarize by yourself.
- FAQ: frequently asked questions which help to understand the incident.
- STEP: the processes to resolve the incident, and you should make sure its completeness. Usually, steps have causally inner connection, the former step will trigger the next step.
- Appendix: the supplement of the incident which is not important or labeled by TSG, usually providing additional resources, data, links and so on.

1. You need to identify each second level subsection, including third level subsection if it needed, analyze its content or purpose, and categorize it accordingly. For those belonging to Step, you should capture the inner connections, such as Causality or Temporal relations, and present them in the correct order.
2. Your returned formulated TSG should be in JSON format. Make sure that the keys originate from these categories: Terminology, FAQ, STEPS ad Appendix. Each value should be a list of dictionaries. The keys for them are "prerequisite", "header", "body", and "linker". All values within the lists need to align with the original context, with truthful meaning and necesary **code block**.
3. Importantly, the "linker" is used to imply dual role of providing the action's result and connecting to the next step using the "if-then" sentence format. You should formulate each steps's linker to be "If any results are obtained by executing the corresponding action in the previous step, then **the true intent of the following step** provided here". Implicit linkers like "proceed to the next step." or "then the intent of the following step should be taken into consideration." should be avoided.
4. For each "if" condition at every step in the STEP, it is necessary to add a special token behind the "then" condition within the "linker". The options for these tokens are "[CONTINUE]", "[CROSS]", and "[MITIGATE]". - The token "[CONTINUE]" indicates that the actions corresponding to this "if" condition are part of the continuum within the same TSG's STEPS. - The token "[CROSS]" signifies that the subsequent actions require a transition to a different set of steps that are external to the current TSG's STEPS. - The token "[MITIGATE]" implies that the actions following the "if" condition convey that the incident is mitigated, or necessitate communication with on-call engineers or teams.
The use of this special token is instrumental in verifying the completeness and structural integrity of the STEPS section.

<TWO EXAMPLES HERE>

[User]
Here is the <TSG> you need to formulate:
{TSG}

Table 9: Instruction that formulates the original unstructured trouble shooting guide into structured one.

[System]
You are a helpful assistant that extract the code template and the default parameters from the provided code instance in <CODE>. <CODE> is a code block contains several parameters. You should replace those parameters with placeholders and output the code template with placeholders and default parameters.

<ONE EXAMPLE HERE>

You response should be in the json format as below:

```
{
    "#CODE_TEMPLATE#": where you replace the parameters in <CODE> with
        placeholders,
    "#DEFAULT_PARAMETERS#": where you keep the parameters in <CODE> as default
         values.
}
```

[User]
Here is the <CODE> you need to extract:
{CODE}

Table 10: Instruction that extracts code template and default parameters from the source code.

[System]
You are a helpful assistant that selects the most relevant element from <LU_LIST> based on the user's query in <QUERY> and chat history in <ChAT_HISTORY>. Please respond with the JSON format.
The each element in <LU_LIST> are in json format and contains the following fields:

```
{
    "#type#": "the type of the element, select from the following types:
        Terminology, FAQ, Step and Appendix.",
    "#meta data#": "the description of the troubleshooting guide.",
    "#prerequisite#": "the prerequisite of this step, before taking current
        step, the prerequisites should be finished.",
    "#header#": "the information describes the intent of the <INFO>.",
    "#body#": "the action is the content which troubleshoots the incident or
        give explanation of the #header#. the action may contain code blocks
        in markdown format, and parameters are replaced with placeholders",
    "#linker#": "the expected output after taking the #action#. It is defined
        in the following format in markdown: -If **condition**, then **
        should_do**. It can contain multiple if-then cases.",
    "#default_parameters#": "the default parameters that could fill in
        placeholders in code blocks in #body#."
}
```

- The elements in <LU_LIST> contains possible information that can answer the user's query in <QUERY>. However, they may not be all relevant to the query or useful to answer the user's query. You should select the most relevant element from the <LU_LIST> based on the user's query in <QUERY>.
- In particular, you should focus on the following fields in the element: #header#, #body#. Most importantly, the <QUERY> need to match with the #intent# and the #body# has to provide actions to reach the goal of the <QUERY>, please ignore the #linker# and do not map the <QUERY> with #linker#.
- As you choosing from <LU_LIST>, you need to check if all the #prerequisite# are met in previous history. If the #prerequisite# not finished, then it should not be chosen.
- Try to select only one element from <LU_LIST>. If it is not possible to select only one element, you can select multiple elements from <LU_LIST>:

```
[
    {
        "INDEX": the index of the element in <INFO_LIST>.
        "INTENT": the #header# of the element, the index starts from 0.
        "EXPLANATION": justify why you select this node.
    }
]
```

- If there is no element in <LU_LIST> that can answer the user's query in <QUERY>, you should try select the most relevant element to the user's query considering that the user might use wrong terminology:

```
[
    {
        "INDEX": the index of the element in <INFO_LIST>.
        "INTENT": the #header# of the element, the index starts from 0.
        "REPHRASED_QUERY": the rephrased query that you think the user is
            asking about.
        "EXPLANATION": justify why you select this node.
    }
]
```

- Unless you are confident that there is no element in <LU_LIST> that is even close to the user's query:

```
{
    "NO_INFO_EXPLANATION": where you give your explanation.
}
```

- Your answer should be in the JSON format in a list after <RESPONSE>.

[User]
<LU_LIST>: {LU_LIST}
<QUERY>: {QUERY}
<CHAT_HISTORY>: {CHAT_HISTORY}

Table 11: Instruction that selects the most relevant logic unit based on user query and chat history.

[System]
You are adept at performing website navigation tasks, and you will be provided with a simulation data from Mind2Web, designed for developing and evaluating generalist agents capable of following language instructions to complete complex tasks on any websites.

The data includes a step-by-step execution process, each step encompassing HTML code, Tasks, Previous Actions and the Element and Action of this step. Note that the Element comes from the HTML code, and if the correct action is not present on current page, the Element is None, and you should retrieve from next step's Previous Actions.

Now your task is to write a comprehensive and adaptable reference document that outlines the general process for completing tasks like the given task. This document should serve as a guide for others to perform similar tasks on the same website in the future. So it should not be limited but can use this data to be the example, and should be general enough.

Please return the complete reference document that adheres to these guidelines.

[User]
The format of the documents should be as follows: {FORMAT}
The given execution process is as follows: {EXAMPLE}

Table 12: Instruction that generates specific format of document for Mind2web dataset.

Format	Description
Structured Markdown	- The document must be structured into sections in markdown format. - It should include task overview, introduction, process steps and conclusion. - Each step in the process including detailed explanations for Intent, Prerequisite, HTML Code Reference, Action, Reason and Result. - The Prerequisite is to specify any conditions or prior actions that must be met or completed before proceeding with the current step in the process. - Ensure that each step is explicitly connected to the next one, and the Result is written in the "if-then" schema where the "Intent" of this step is completed, the outcome "then" is the next step's Intent. - The HTML Code Reference gives hints of the Action likes some '<button>', '' or other elements or attributes. You need to use the given task as an example. - The Action comes from "Click", "Type", "Hover", "Press Enter".
Hierarchical Guideline	- A structured text document with numbered steps for each task. - Each step includes a title, description, the HTML code involved, and the action to be taken. - Previous actions are referenced where necessary, with hyperlinks to the relevant steps. - Appendices for HTML code references, glossary of terms, and FAQs.
Tabular Checklist	- A printable checklist with each task and subtask, including checkboxes for completion. - Each checklist item includes a code snippet and the action required. - A troubleshooting section that lists common problems and their solutions. - Tips for what to do when the expected element or action is not available. - References to more detailed instructions or external resources for complex tasks.
Narrative Document	An entire description of the execution process without special structures.

Table 13: The description of different formats we use in our experiment.

Paradigm	Mind2Web Cross-Task		
	Ele. Acc	**Op. F1**	**Step SR**
GPT-3.5	**40.69**	**49.66**	**33.91**
$w/o.$ history steps	36.25	45.00	31.38
GPT-4	**62.80**	**60.37**	**51.81**
$w/o.$ history steps	56.97	59.09	50.38

Table 14: Ablation study of historical steps on Mind2web.

#Agent	Mind2Web Cross-Task			
	Ele. Acc	**Op. F1**	**Step SR**	**SR**
Single	68.29	69.53	61.94	12.30
Multiple	**68.40**	**71.50**	**62.18**	**15.66**

Table 15: Analysis of systems with different agents for Dynamic How-To Questions.

[System]
You are a helpful and precise assistant for checking the quality of the answer. We would like to invite you to evaluate the performance of the system in answering a user's question in <Question>.

I will give you the answer generated by the system in <Generation> and the ground truth answer in <Ground Truth> respectively. Your evaluation will contain five sub-evaluation tasks:

1. Both two answers contain a list of steps. Your task is to extract action items from the provided steps in both answers. The action item is defined like a combination of action and element. Compare the action items to identify similarities. Output the similar action items. Count the count of similar action items.

- Your answer should contain the extracted two action item sets (in the format as a list of string).
- Your answer should contain the set of similar action items (in the format as a list of string). Similar action items are those sharing similar intent or achieving similar goals. Each similar action pair in the list should be in the format of "similar action item from action item set1 / similar action item from action item set2" - Your answer should contain the count of similar action items.

2. Can <Generation> completely solve the user's question?
- Your answer should be "Yes" or "No".
- Your answer should contain the reason(s) for your choice. You should not focus on the length of the answer or the details of the answer, but you should focus on whether the steps could solve the user's question and the quality of the steps compared with the ground truth.

Your output should be in the following format in json:

```
{
    "Subtask1": {
        "Action items in Generation": ["action item 1", "action item 2", ...],
        "Action items in Ground Truth: ["action item 1", "action item 2",
            ...],
        "Similar action items": ["similar action item 1", "similar action item
             2", ...],
        "Count of similar action items": 2
    },
    "Subtask2": {
        "Choice": "Yes" or "No",
        "Reason": "reason for your choice"
    }
}
```

[User]
Here is the user's question <Question>: {Question}
The answer from system <Generation> is: {Generation}
The ground truth answer <Ground Truth> is: {Ground Truth}

Table 16: Instruction that evaluates the generated answer compared with ground truth for Wikihow.

[System]
You are a helpful assistant that is great at website design, navigation, and executing tasks for the user. Now please proceed with the <CURRENT_STEP> and make your choice, remember that only based on the helpful document information from <DOC_CONTEXT> and the previous step chat history between user and assistant in <CHAT_HISTORY>.

Your response should be in the format of "Answer: C. Action: SELECT Value: Pickup".
The Answer is A, B, C... , the Action comes from [CLICK, TYPE, SELECT] and the Value is not always needed.

[User]
<DOC_CONTEXT>: {DOC_CONTEXT}
<CHAT_HISTORY>: {CHAT_HISTORY}
<CURRENT_STEP>: {CURRENT_STEP}

Table 17: Instruction that used for the baselines of RAG system on Mind2web.

[System]
You are a helpful assistant that is great at website design, navigation, and executing tasks for the user. Now please proceed with the <CURRENT_STEP> and make your choice, remember that only based on the helpful structured document information from <LU>, and the previous step chat history between user and assistant in <CHAT_HISTORY>.

Your response should be in the format of JSON:

```
{
    "CHOICE": the choice you make from A B C ...,
    "ACTION": the corresponding action choosing from ['CLICK', 'TYPE', 'SELECT
        '],
    "VALUE": the corresponding value if needed,
    "INTENT": the intent of next step, which should be retrieved and judged
        from the "if" conditions in #output# from <LU> according to current
        step and actions and choose the corresponding "then" outcome, do not
        guess it based on current Task in <CURRENT_STEP> by yourself unless
        the <LU> is irrelevant to <CURRENT_STEP>,
}
```

[User]
<LU>: {LU}
<CHAT_HISTORY>: {CHAT_HISTORY}
<CURRENT_STEP>: {CURRENT_STEP}

Table 18: Instruction that used for the RAG system utilizing Thread Paradigm on Mind2web.