Trification: A Comprehensive Tree-based Strategy Planner and Structural Verification for Fact-Checking

Anonymous ACL submission

Abstract

Technological advancements allow information to be shared with a single click, which has enabled the rapid spread of false information. This makes automated fact-checking systems necessary to ensure the safety and integrity of our online media ecosystem. Previous methods have demonstrated the effectiveness of decomposing the claim into simpler sub-tasks and utilizing LLM-based multi-agent system to execute them. However, those models face two limitations: they often fail to verify every component of a claim and lack a structured framework to logically connect the results of sub-tasks for a final prediction. In this work, we propose a novel automated fact-checking framework called `Trification`. Our framework begins by generating a comprehensive set of verification actions to ensure complete coverage of the claim. It then structures these actions into a dependency graph to model the logical relationships between actions. Furthermore, the graph can be dynamically modified, allowing the system to adapt its verification strategy. Experimental results on two challenging benchmarks demonstrate that our framework significantly enhances fact-checking accuracy, thereby advancing current state-of-the-art in automated fact-checking systems.

1 Introduction

Technological advancements have fundamentally changed how online media operates. The rise of the internet and social media platforms allows information to be shared with just a single click. However, this ease of sharing has also enabled the rapid spread of false information, underscoring the critical need for automated fact-checking systems. Automated fact-checking systems have evolved significantly over the past few decades. Early approaches (Hanselowski et al., 2018; Zhou et al., 2019; Liu et al., 2020; Zhong et al., 2020) typically followed a standard three-step pipeline: (1) document retrieval for retrieving relevant documents, (2) sentence selection for extracting top-k candidate evidence, and (3) veracity prediction for label prediction. With the recent advancements of Large Language Models (LLMs), the paradigm is now shifting towards more integrated, LLM-based agent methods. Instead of executing predefined steps, LLM-based agents leverage the LLM's reasoning capabilities to plan a verification strategy, interact with external tools, and synthesize information to arrive at a final verdict. These strategies have primarily focused on steps like claim decomposition, retrieval methods, question answering, and final reasoning (Pan et al., 2023; Zhao et al., 2024; Ma et al., 2025; Xu et al., 2024).

Despite the success of these LLM-based fact-checking agents, they often suffer from two critical shortcomings, as shown in Figure 1. First, they frequently fail to perform complete verification of every component within a complex claim, often because of their dependence on a linear reasoning path that may overlook short or implicit statements in the claim. Second, they lack a mechanism to make logical connections between the results of each sub-task because they often treat them as isolated steps rather than interdependent nodes in a reasoning graph. This prevents the agents from performing logical and more comprehensive reasoning, leading to logical errors and causal issues (Ma et al., 2025).

Therefore, to address the above limitations, we propose a fact-checking framework that leverages a tree-based strategy planner and a structural verification process called `Trification`. Unlike previous systems (Ma et al., 2025; Zhao et al., 2024; Xu et al., 2024) that generate sub-tasks sequentially - a process prone to premature termination, our framework begins by generating a complete set of verification actions. This approach ensures that every part of the claim is covered by the set. Our framework then structures these actions into

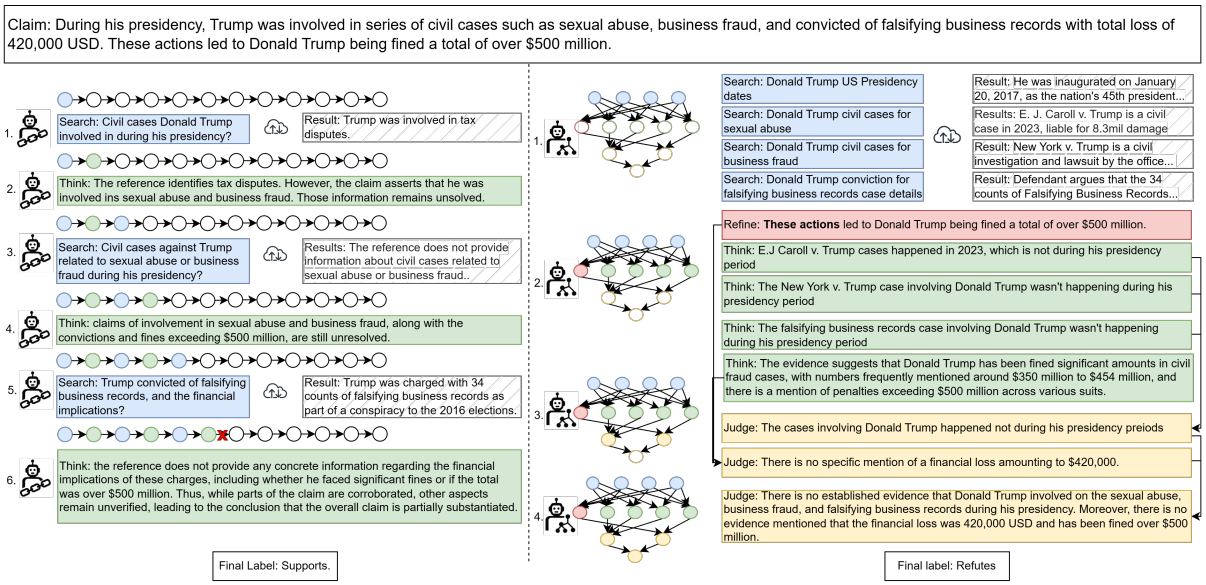

Figure 1: Case Comparison: Chain Search vs. Tree Verification

a dependency graph, which provides two key advantages. First, in contrast to earlier methods (Pan et al., 2023; Zhao et al., 2024) that combine all sub-tasks' results at the final step, our graph introduces intermediate logical verification between dependent actions. This enables a more fine-grained and rigorous evaluation of a claim. Second, by explicitly modeling dependencies, ready-to-execute actions (those with no pending dependencies) can be processed concurrently, alleviating the computational bottleneck inherent in the sequential process.

Moreover, to prevent our system from ending prematurely or producing non-sensical output, our framework allows for dynamic modification of the verification graph when a specific action fails. In such cases, the system is prompted to generate a new sub-graph with an alternative verification strategy, which is then integrated into the original verification graph. This iterative recovery mechanism enhances the system's adaptability and mirrors the creative approach of human fact-checkers.

In summary, our contributions are threefold:

1. We propose `Trification`, a novel fact-checking framework that uses a tree-based strategy to generate a comprehensive set of verification actions, ensuring no component of a claim is overlooked.

2. We introduce an executable dependency graph to structure the verification actions. This graph explicitly models logical relationships between actions, enabling more fine-grained reasoning and concurrent execution to reduce computational bottlenecks.

3. We design a graph modification mechanism that allows the system to adapt its strategy upon failure, enhancing its ability to explore alternative verification strategies.

2 Related Work

Automated fact-checking has undergone substantial evolution, shifting from modular pipelines to LLM-based reasoning agents. Early pipeline systems, such as UKP-Athene (Hanselowski et al., 2018), laid the foundation by combining document retrieval with neural entailment models for claim classification. However, these systems typically processed evidence in isolation, limiting their ability to capture inter-evidence dependencies. To address this, subsequent work introduced graph-based reasoning architectures that modeled relationships among multiple evidence pieces. GEAR (Zhou et al., 2019) employed a fully connected evidence graph with BERT-based aggregators to enable information transfer between nodes, while KGAT (Liu et al., 2020) refined this design using kernel-based attention to control evidence propagation and assess node importance more precisely. Later systems extended this idea to semantic-level graphs, leveraging semantic role labeling and Graph Neural Networks over contextual encoders to reason over structured evidence representations (Zhong et al., 2020). This entire evolutionary trajectory fundamentally underscores the critical importance

of structured evidence processing for robust fact-checking, but these systems remained limited by their reliance on supervised learning strategies.

Recent advances in Large Language Models have enabled a new generation of agent-based fact-checking systems, in which the model plans and executes verification steps. Instead of adhering to a fixed pipeline, LLM-based agents leverage their reasoning abilities to decompose claims, retrieve evidence, and synthesize information across multiple sources (Pan et al., 2023; Zhao et al., 2024; Ma et al., 2025; Xu et al., 2024). This paradigm shift has improved flexibility and interpretability, allowing models to mimic human-like investigative workflows, however, they still face two major limitations. First, their reasoning often follows a linear verification chain, which can overlook implicit or short sub-claims embedded within complex statements. Second, existing agents generally treat subtasks as independent modules, lacking a coherent framework for connecting and reasoning over the interdependencies among sub-claims or evidence. Without such structured reasoning, models are prone to logical inconsistencies and causal misinterpretations, as observed in recent analyses (Ma et al., 2025). In contrast to sequential approaches, our tree-based fact-checking framework introduces a structured and adaptive reasoning process. It enables the concurrent execution of independent tasks and incorporates dynamic graph modification to ensure robust and adaptive reasoning.

3 Methods

Given a claim C, the goal of fact-checking is to assess its accuracy by assigning a label y, either SUPPORTS or REFUTES, indicating the veracity label of the given claim. The process involves gathering a set of evidence E from the provided knowledge base or the internet.

3.1 Tree Planner Generation

To effectively address these challenges and improve operational efficiency—particularly in parallel processes—we introduce `Trification`, a comprehensive tree-based strategy planner and structural verification framework. Our approach models the fact-checking process as a dependency graph, utilizing a tree planner to establish a clear and organized verification workflow. This design underpins the core of our **Tree-based Strategy Planner** and **Structural Verification system**, known as `Trification`.

Trification generates an initial verification plan based on the input claim. This initial plan is represented as a directed acyclic graph (DAG), where nodes correspond to specific verification actions and edges define their dependencies. The DAG structure facilitates concurrent action execution, significantly improving computation efficiency by parallel processing. The prompt for plan generation is provided in Table 4.

The framework starts by constructing an initial Directed Acyclic Graph ($\mathcal{G}$), where nodes represent verification actions and edges define dependencies between them. The framework employs four node types, each representing a specific action in the verification process: (1) `SEARCH` Node to perform evidence retrieval, (2) `REFINE` Node to resolve ambiguity in the node's input, (3) `THINK` Node to perform intermediate logical reasoning by synthesizing information from multiple nodes, and (4) `JUDGE` Node to predict the final verdict. Once the graph $\mathcal{G}$ is generated, the framework executes the actions concurrently, as long as no dependencies are blocking their execution.

3.2 Node Specification

Each node in the graph is an object characterized by the following mandatory attributes:

- **id**: The node's unique identifier.
- **type**: The type of node, which determines its operations. The four possible types are `SEARCH`, `REFINE`, `THINK`, and `JUDGE`.
- **input**: The specific instruction or task the node is designed to execute.
- **hint**: The additional context that defines both the integration logic for parent node outputs and the node's role within the global verification strategy.
- **dependencies**: A list of ids corresponding to the node's parent nodes. A node is eligible for execution only when all its dependencies have been successfully processed.

The specifications for each node type are detailed in the following subsections.

`SEARCH` Node The `SEARCH` node is responsible for executing a search operation to retrieve relevant information for subsequent steps in the process. The input is a query string, which represents the

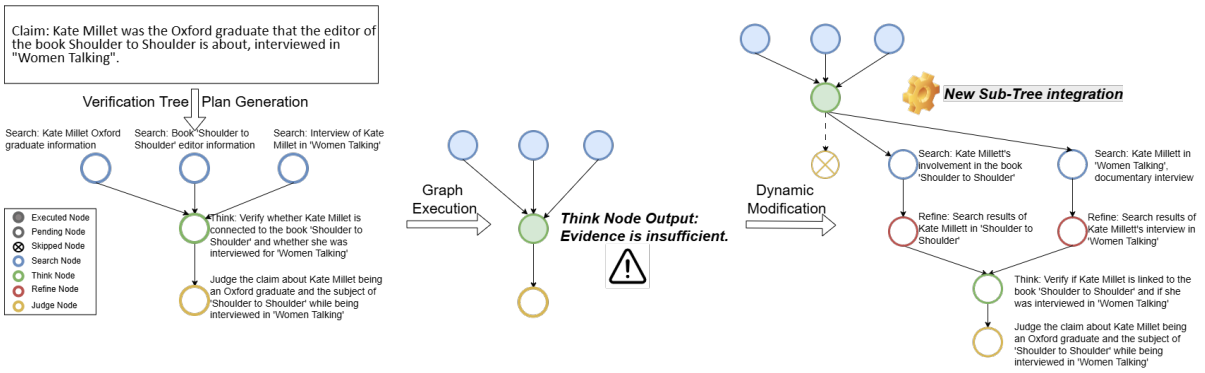

Figure 2: Dynamic Tree Planner: insufficient evidence triggers dynamic graph modification and integration of a newly generated subtree

specific information or evidence needed to validate the claim or address the question at hand.

The node begins by utilizing an LLM to generate a search query based on the node's input and hints. The query is conditioned on the node's input and strategic hints, ensuring it retrieves information relevant to downstream nodes in the verification graph. The prompt used for search query generation can be found in Appendix A.2.

For each generated query search, we perform evidence retrieval using one of the two strategies: Wiki or Search_engine. The Wiki strategy follows ProgramFC (Pan et al., 2023), using BM25 to retrieve top-k paragraphs from the Wikipedia dump. The Search_engine strategy employs the open-source search tool Dux Distributed Global Search (DDGS)[1] to process the search query and collect the top-k returned snippets.

The search results are then stored in the evidence field as a list of pertinent data points. The SEARCH node synthesizes the gathered information, ensuring that it aligns with the original query to provide meaningful and useful results for the subsequent nodes in the process.

THINK Node The THINK node functions as a central reasoning component that processes a specific declaration requiring verification by first synthesizing all available evidence from its parent dependencies. It then engages an LLM to perform intermediate reasoning over this synthesized information, aiming to produce a coherent conclusion.

REFINE Node The REFINE node acts as a critical pre-processing agent designed to resolve ambiguity in the original user query. For instance, given the claim in Figure 1 , a node might be generated with the following input: "These actions led to Trump being fined over $500 million". This node would replace the ambiguous phrase "These actions" with the specific case names, which is essential for proper verification.

REFINE node resolves ambiguity by leveraging contextual evidence gathered from its dependency nodes. The process begins by collecting the outputs from all its parent nodes, which typically contain relevant search results or established facts. It then employs an LLM to analyze this contextual information to identify and resolve vague references—such as ambiguous pronouns like "they" or "those cases". The core function of the node is not to evaluate the truth of a claim but to generate a new, precise, and disambiguated query that can be effectively processed by subsequent operational nodes. The output is an optimized query text, which subsequently replaces the original input for downstream nodes, ensuring that those nodes will operate on a clear and unambiguous foundation.

JUDGE Node The JUDGE node serves as the terminal verdict mechanism within the reasoning graph, tasked with delivering a final judgment on the truthfulness of a target declaration. It operates by first aggregating the complete body of evidence compiled throughout the verification process, which is sourced from all of its dependency nodes. This synthesized information is then presented to an LLM to perform a comprehensive analysis and render one of two possible labels: SUPPORTS if the evidence confirms the declaration otherwise REFUTES.

3.3 Graph Execution Flow

Each node in the DAG is pre-processed before execution to ensure the accuracy and consistency of the verification process. The process involves:

[1]DDGS: https://github.com/deedy5/ddgs

1. Evidence Propagation: Evidence and output from parent nodes are propagated to the current node. This step ensures that all supporting information is available for task execution, enabling the node to operate on the latest and most relevant information.

2. Input Refinement: If any parent node is a `REFINE` node, the current node's input is updated with the output of that `REFINE` node. To maintain clarity in the refinement chain, we constrain `REFINE` nodes to have at most one parent node.

The graph executes via topological sort. Execution begins with the root nodes running in parallel. The system dynamically schedules a node for execution as soon as all its parent nodes have finished. This process continues until all the nodes have been processed.

3.4 Graph Modification for Dynamic Planner

During the verification process, an agent might deem a `THINK` Node or a `JUDGE` Node as unsolvable because of lacking information. A key feature of our framework is the dynamic generation of the verification actions. The core algorithm governing our dynamic tree generation and action execution is outlined in the Algorithm 1.

For the `THINK` Node, since this is a dependency tree, if a node fails, we cannot perform accurate execution for its descendants. In such cases, the framework dynamically adapts by skipping over its descendant tasks and generating a new sub-tree ($\mathcal{G}_{sub}$) specifically designed to seek out the missing information. Once the new subtree is generated, it is integrated into the original graph $\mathcal{G}$. Ultimately, the `THINK` node outputs a finalized reasoning summary that is stored as its result, thereby consolidating its analytical step within the larger reasoning framework. For example, in Figure 2, after execution the thinking node, the output is evidence is insufficient, so our framework regenerate a new sub-tree to search more information.

For the `JUDGE` node, if the output is uncertain, the node does not simply exit but proactively triggers a graph correction to initiate a new subtree aimed at acquiring the missing evidence. Upon reaching a determinate verdict, the node's output consists of both the definitive label and a detailed textual explanation outlining the reasoning behind its final judgment. This ensures that the verification process adapts dynamically to prevent the system from hallucinating or ending prematurely.

The complete pipeline is illustrated in Figure 2.

Algorithm 1 Dynamic Tree Planner Algorithm

Input: Claim ($\mathbf{C}$)
Output: $\mathbf{Label}$
Generate a Directed Acyclic Graph verification plan $\mathcal{G}$
Concurrently execute nodes in $\mathcal{G}$ following topological order
for each node $\mathbf{v} \in \mathcal{G}$ **do**
  Let $\mathbf{v_{orig}}$ be the intrinsic input associated with node v
  Let $\text{Parents}(v) = \{u \mid (u \to v) \in \mathcal{G}\}$
  Collect parent outputs:

$$\mathbf{v_{parents}} = \{\text{output}(u) : u \in \text{Parents}(v)\}$$

  Combine intrinsic input and parent outputs:

$$\mathbf{v_{input}} = \text{Combine}(\mathbf{v_{orig}}, \mathbf{v_{parents}})$$

  if type(v) = SEARCH **then**
    $\text{output}(v) \leftarrow \text{Search}(\mathbf{v_{input}})$
  else if type(v) = REFINE **then**
    $\text{output}(v) \leftarrow \text{Refine}(\mathbf{v_{input}})$
  else if type(v) = THINK **then**
    $\text{output}(v) \leftarrow \text{Reason}(\mathbf{v_{input}})$
    if evidence insufficient **then**
      Mark v as failed
      Generate new subtree $\mathcal{G}_{sub}$ from v
      $G \leftarrow G \cup \mathcal{G}_{sub}$
      Continue concurrent execution
    end if
  else if type(v) = JUDGE **then**
    $\text{output}(v) \leftarrow \text{Judge}(\mathbf{v_{input}})$
    if evidence insufficient **then**
      Mark v as failed
      Generate new subtree $\mathcal{G}_{sub}$ from v
      $G \leftarrow G \cup \mathcal{G}_{sub}$
      Continue concurrent execution
    end if
  end if
end for
Output: Label = SUPPORTS / REFUTES

4 Experiment Setups

4.1 Datasets

We evaluated our `Trification` framework using two widely adopted claim verification benchmarks: FEVEROUS (Aly et al., 2021) and HOVER (Jiang et al., 2020). Both datasets consist of claims originating from Wikipedia and cover a diverse range of complexity that requires multi-hop reasoning. Specifically, the HOVER dataset consists of 4,000 claims categorized into three complexity levels: 1,126 two-hop, 1,835 three-hop, and 1,039 four-hop claims. The FEVEROUS dataset consists of 2,962 complex claims. In line with the previous method (Pan et al., 2023), we only select claims that require sentence-level evidence only.

4.2 Baseline

We compare Trification with various baselines that are categorized into four different categories:

- **Fine-tuned**: These methods leverage language models fine-tuned on claim verification dataset. Specifically, BERT-FC (Soleimani et al., 2019) and LisT5 (Jiang et al., 2021) fine-tuned BERT and T5 models on the fact-checking benchmarks, respectively. RoBERTa-NLI (Nie et al., 2020) and DeBERTa-NLI (He et al., 2021) are based on the RoBERTa-large and DeBERTaV3 models, respectively, and are fine-tuned on the FEVER (Thorne et al., 2018) dataset along with four NLI datasets. MULTIVERS (Wadden et al., 2022) uses a LongFormer (Beltagy et al., 2020) model and tuned under FEVER dataset.

- **LLM-based**: These methods perform claim verification by directly prompting LLMs. Examples include using zero-shot prompts with ChatGPT (Zhao et al., 2024) or few-shot prompts with models like Codex (Li et al., 2022) and Flan-T5 (Chung et al., 2024).

- **LLM Agent-based**: These methods leverage LLM as agents to execute specific sub-tasks within a verification pipeline. ProgramFC (Pan et al., 2023) generates executable programs for operations like retrieval, question answering, or fact verification. Similarly, PACAR (Zhao et al., 2024) introduces a multi-agent system that decomposes claim and assigns the resulting tasks to specialized tool executors, followed by a final verification step.

- **LLM Agent-based with Dynamic Planning**: These methods go beyond standard LLM-agent-based approaches by adjusting their verification plan during execution. LoCaL (Ma et al., 2025) uses a multi-round decomposer for dynamic planning and two evaluating agents for two-way confidence updating to enhance logical and causal consistency. SearChain (Xu et al., 2024) uses LLM-based agents to dynamically verify and complete Chain-of-Query (CoQ) reasoning path, allowing it to iteratively correct and restructure its reasoning process. To ensure a fair comparison, our evaluation of these baselines uses a configuration consistent with our own settings, i.e., utilizing GPT-4o-mini as the agent and a Wikipedia dump as the knowledge base.

4.3 Implementation Details

We employ GPT-4o-mini with zero-shot prompting technique as our primary LLM backbone. For evaluation, we adopt the standard Macro-f1 score (Pan et al., 2023; Ma et al., 2025; Zhao et al., 2024) to assess claim verification performance. We conduct experiments under two distinct settings:

- **Static Mode**: This is a baseline configuration designed for a fair comparison with other baselines such as ProgramFC (Pan et al., 2023). In this mode, the verification graph is executed once without modification. The SEARCH node employs the Wiki strategy. Specifically, the BM25 retriever from the Pyserini toolkit (Lin et al., 2021) over a static Wikipedia dump, fetching the top-10 paragraphs.

- **Dynamic Mode**: This is the full expression of our framework's adaptive capability. Specifically, the SEARCH node employs the Search_engine strategy, fetching the top-10 relevant snippets. The graph can undergo up to 3 modifications to recover from failures or refine its strategy. Should verification remain unsuccessful after reaching the maximum number of modifications, the process defaults to static mode.

5 Experimental Results

5.1 Results of Trification in Static Mode

Table 1 presents the macro F1-score of our Trification compared to state-of-the-art models across different categories. On the HOVER dataset, our method achieves performance gains of 2.00%, 2.35%, and 2.41% for the 2-hop, 3-hop, and 4-hop settings. The increasing trend of the improvements underscores that our tree-based planner and graph structure are crucial for managing the increased complexity of multi-hop claims. Consequently, our framework successfully narrows the performance gap between complexity levels, reducing the difference between 2-hop and 3-hop to 8.71%, and between 3-hop and 4-hop to just 0.19%. Similarly, our framework outperforms all state-of-the-art methods by 2.11% on the FEVEROUS dataset. In average, Trification improves fact-checking performance by 2.23% across all evaluated benchmarks.

Models		HOVER 2-hop	HOVER 3-hop	HOVER 4-hop	FEVEROUS	AVERAGE
I	BERT-FC	50.68	49.86	48.57	51.67	50.20
	LisT5	52.56	51.86	50.46	54.15	52.27
	RoBERTa-NLI	63.62	53.99	52.40	57.80	56.95
	DeBERTaV3-NLI	68.72	60.76	56.00	58.81	61.07
	MULTIVERS	60.17	52.55	51.86	56.61	55.30
II	ChatGPT	66.94	60.56	58.73	55.72	60.49
	Codex	65.07	56.63	57.27	62.58	60.39
	Flan-T5	69.02	60.23	55.42	63.73	62.10
III	ProgramFC (N=1)	69.36	60.63	59.16	67.80	64.24
	ProgramFC (N=5)	70.30	63.43	57.74	68.06	64.88
	PACAR	73.13	64.07	63.82	72.61	68.40
IV	LoCaL	72.71	64.11	61.59	68.22	66.66
	SearChain	64.46	60.30	56.54	66.69	62.00
Our	Trification Dynamic	**75.61**	**67.89**	65.32	**77.95**	**71.69**
	Trification Static	75.13	66.42	**66.23**	74.72	70.63
	w/o REFINE	71.22	61.56	60.99	71.30	66.26
	w/o REFINE and THINK	65.64	55.10	55.20	70.58	61.63

Table 1: Macro-F1 (%) score on HOVER and FEVEROUS dataset. The best and second-best results in each column are indicated with bold and underlined text, respectively.

5.2 Results of Trification in Dynamic Mode

Table 1 shows the results of Trification in dynamic settings. Our dynamic settings improve the macro F1-score upon our static setting on most subsets:, by 0.48%, 1.47% on 2-hop and 3-hop HOVER dataset, and 3.23% on FEVEROUS dataset.

This result underscores the value of a structured approach to dynamic planning. Unlike some prior agents (e.g., SearChain) which may underperform fixed-planning baselines due to unfocused iteration, our framework embeds dynamic adaptation within a DAG. This structure guides the process by predefining a comprehensive set of verification actions and their dependencies, ensuring that adaptation occurs in a controlled and goal-oriented manner. Consequently, it harnesses the benefits of dynamic planning while mitigating the risk of inefficiency, leading to the observed performance gains.

This outcome is achieved efficiently. We observe a decrease in the number of uncertain predictions under the dynamic tree setting, which indicates the effectiveness of the dynamic strategy. On average, our framework required 0.85 graph modifications on HOVER and 0.96 on FEVEROUS. The number of SEARCH nodes increases slightly (2.74 $\rightarrow$ 3.44 on HOVER and 2.64 $\rightarrow$ 4.26 on FEVEROUS). This indicates that our approach recovers from failures with minimal overhead, largely by using information from the original tree rather than initiating new searches for the same information.

Beyond search, the dynamic planner also introduces modest increases across other node types. The number of THINK nodes rises from 1.45 $\rightarrow$ 2.04 on HOVER and 1.67 $\rightarrow$ 2.37 on FEVEROUS, while REFINE nodes increase from 0.40 $\rightarrow$ 0.88 and 0.16 $\rightarrow$ 0.56, respectively. Likewise, JUDGE nodes grow from 1.04 $\rightarrow$ 1.88 on HOVER and 1.02 $\rightarrow$ 1.98 on FEVEROUS. These increases reflect the dynamic planner's ability to explore alternative verification paths, but the magnitude remains modest—further supporting the conclusion that the method recovers from failures using minimal overhead, largely by reusing evidence from the original tree rather than repeatedly initiating new searches.

In terms of latency, the dynamic tree is approximately 8 seconds slower on average than the static tree. This delay is primarily due to the increased number of SEARCH nodes, which necessitates more calls to the external search engine API. Additionally, at the node level, we observe that REFINE nodes are executed significantly more frequently in the dynamic setting. The THINK nodes in the dynamic tree also exhibit shorter average processing time than those in the static tree. A possible explanation is that dynamic-tree think nodes have a simpler objective—they only need to determine whether the existing evidence is sufficient—whereas static-tree think nodes must also generate intermediate stances and reasoning outputs. These findings collectively underscore the importance of combining graph modification with a search engine to explore alternative verification paths for complex reasoning tasks.

Collectively, these findings highlight the effec-

	HOVER	FEVEROUS
Graph Modification	0.85%	0.96%
SEARCH Node Increased	2.74 → 3.44	2.64 → 4.26
THINK Node Increased	1.45 → 2.04	1.67 → 2.37
REFINE Node Increased	0.40 → 0.88	0.16 → 0.56
JUDGE Node Increased	1.04 → 1.88	1.02 → 1.98

Table 2: Dynamic Tree Planner Statistics.

tiveness of combining graph modification with targeted search to explore alternative verification paths in complex, multi-hop reasoning tasks, achieving large performance gains with minimal computational overhead.

5.3 Contribution of REFINE and THINK Nodes

We examine the effectiveness of each modules in Trification by implementing several variants in the open book setting, as shown in the lower part of Table 1. The full model achieves the highest performance across all settings, with an average Macro-F1 score of 70.63%, demonstrating the effectiveness of the complete reasoning pipeline that integrates SEARCH, REFINE, THINK, and JUDGE nodes. Removing the REFINE nodes leads to a consistent drop in performance (average = 66.26%), indicating that iterative evidence refinement plays a crucial role in enhancing multi-hop reasoning accuracy. When both the THINK and REFINE nodes are excluded, performance declines further (average = 61.63%), particularly on HOVER test data, confirming that the reasoning and refinement stages are vital for managing complex evidence trees. Overall, these results validate the complementary roles of the REFINE and THINK processes in enabling the model to generate coherent, evidence-grounded fact-checking decisions.

5.4 Error Analysis

To better understand the limitations of our system, we conducted a manual analysis on 10 failed claims. The analysis reveals two main sources of failure: (1) Search Engine Errors and (2) Ambiguous Claims. Table 3 presents representative examples of each failure type.

Search Engine 70% Among the analyzed cases, 70% (7 out of 10) of the errors originated from the search engine component. Of these seven, four were due to the search engine retrieving evidence that *contradicted* the ground truth, while the remaining three occurred because the search engine *failed to retrieve* relevant evidence.

In the first case type, the search results supported an incorrect interpretation of the claim. For example, the claim "Gregg Rolie is not a keyboardist" is labelled as Supports, but the search engine returned evidence asserting that "Gregg Rolie is a keyboardist" leading to a contradiction. In the second case type, the search engine could only retrieve partial evidence. For instance, the claim that "Arthur Noss was a gunner at the Battle of Britain and Battle of Malta" was partially supported—the evidence confirmed that Noss was a gunner, but not that he served in the two specified battles.

Ambiguous Claim 30% The remaining 30% (3/10) of failures resulted from ambiguities in the claim itself, which confused the reasoning process of either the planner or the judge module.

For example, the claim "The by-election of a constituency represented in the House of Commons of the UK Parliament since 2010 by David Rutley was held on 30 September 1971" combines events from different time periods (2010 and 1971), making temporal interpretation unclear. Another example is the claim "There were 331 episodes of the TV series where Julianna Margulies had the role of Carol Hathaway." The system misinterpreted this as implying that Julianna appeared in all 331 episodes, whereas the claim only asserts that she played that role in a TV series that had 331 episodes in total, not necessarily in every episode. Developing post-processing strategies to handle conflicting search results or to disambiguate unclear claims remains an important direction for future work.

6 Conclusion

In this work, we propose Trification, an automated fact-checking framework that improves verification accuracy through dynamic, graph-based planning. The framework first constructs a complete dependency graph of verification actions and then adaptively modifies it in response to insufficient or ambiguous evidence. Rigorously evaluated on the FEVEROUS and HOVER benchmarks, Trification achieves superior performance over existing methods, demonstrating the effectiveness of structured yet flexible verification planning. Moving forward, we aim to enhance its applicability by better handling claim ambiguity and conflicting evidence, paving the way for more reliable fact-checking in a wider range of real-world scenarios.

Limitations

One of the limitation is the data leakage on when using search engine. As the dataset was created several years ago, using a contemporary search engine introduces the risk of evaluating claims against information that was unavailable at the dataset's creation time. This temporal leakage could confound the assessment of pure reasoning ability with access to newer data.

References

Rami Aly, Zhijiang Guo, Michael Sejr Schlichtkrull, James Thorne, Andreas Vlachos, Christos Christodoulopoulos, Oana Cocarascu, and Arpit Mittal. 2021. FEVEROUS: Fact extraction and VERification over unstructured and structured information. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, and 1 others. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Andreas Hanselowski, Hao Zhang, Zile Li, Daniil Sorokin, Benjamin Schiller, Claudia Schulz, and Iryna Gurevych. 2018. UKP-athene: Multi-sentence textual entailment for claim verification. In *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)*, pages 103–108, Brussels, Belgium. Association for Computational Linguistics.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*.

Kelvin Jiang, Ronak Pradeep, and Jimmy Lin. 2021. Exploring listwise evidence reasoning with t5 for fact verification. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 402–410.

Yichen Jiang, Shikha Bordia, Zheng Zhong, Charles Dognin, Maneesh Singh, and Mohit Bansal. 2020. HoVer: A dataset for many-hop fact extraction and claim verification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3441–3460, Online. Association for Computational Linguistics.

X Li, DG Wang, S Wang, S Wang, Y Wang, Y Wang, Y Wang, Y Wang, Z Wang, Z Wang, and 1 others. 2022. Evaluating large language models trained on code. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 12345–12356.

Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: An easy-to-use python toolkit to support replicable ir research with sparse and dense representations. *arXiv preprint arXiv:2102.10073*.

Zhenghao Liu, Chenyan Xiong, Maosong Sun, and Zhiyuan Liu. 2020. Fine-grained fact verification with kernel graph attention network. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7342–7351, Online. Association for Computational Linguistics.

Jiatong Ma, Linmei Hu, Rang Li, and Wenbo Fu. 2025. Local: Logical and causal fact-checking with llm-based multi-agents. In *Proceedings of the ACM on Web Conference 2025*, pages 1614–1625.

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2020. Adversarial NLI: A new benchmark for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4885–4901, Online. Association for Computational Linguistics.

Liangming Pan, Xiaobao Wu, Xinyuan Lu, Anh Tuan Luu, William Yang Wang, Min-Yen Kan, and Preslav Nakov. 2023. Fact-checking complex claims with program-guided reasoning.

Amir Soleimani, Christof Monz, and Marcel Worring. 2019. Bert for evidence retrieval and claim verification. *Preprint*, arXiv:1910.02655.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. Fever: a large-scale dataset for fact extraction and verification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 809–819.

David Wadden, Kyle Lo, Lucy Lu Wang, Arman Cohan, Iz Beltagy, and Hannaneh Hajishirzi. 2022. MultiVerS: Improving scientific claim verification with weak supervision and full-document context. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 61–76, Seattle, United States. Association for Computational Linguistics.

Shicheng Xu, Liang Pang, Huawei Shen, Xueqi Cheng, and Tat-Seng Chua. 2024. Search-in-the-chain: Interactively enhancing large language models with search for knowledge-intensive tasks. In *Proceedings of the ACM Web Conference 2024*, WWW '24, page 1362–1373, New York, NY, USA. Association for Computing Machinery.

Xiaoyan Zhao, Lingzhi Wang, Zhanghao Wang, Hong Cheng, Rui Zhang, and Kam-Fai Wong. 2024. PACAR: Automated fact-checking with planning and customized action reasoning using large language models. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 12564–12573, Torino, Italia. ELRA and ICCL.

Wanjun Zhong, Jingjing Xu, Duyu Tang, Zenan Xu, Nan Duan, Ming Zhou, Jiahai Wang, and Jian Yin. 2020. Reasoning over semantic-level graph for fact checking. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6170–6180, Online. Association for Computational Linguistics.

Jie Zhou, Xu Han, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2019. GEAR: Graph-based evidence aggregating and reasoning for fact verification. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 892–901, Florence, Italy. Association for Computational Linguistics.

A Appendix

A.1 Error Case Analysis Details

A.2 Prompts

Prompts for nodes:

Table 3: Representative failure cases from error analysis.

Failure Type	Case Description
Search Engine	**Contradictory Evidence:** A Claim stated that Gregg Rolie is not a keyboardist is labelled as Supports. However, retrieved evidence asserted that he *is* a keyboardist.
Search Engine	**Incomplete Evidence:** Claim stated that Arthur Noss was a gunner at the Battle of Britain and Battle of Malta. The search engine only found evidence of him being a gunner, but not his participation in the two battles.
Ambiguous Claim	**Temporal Ambiguity:** "The by-election of a constituency represented in the House of Commons of the UK Parliament since 2010 by David Rutley was held on 30 September 1971." Ambiguous temporal relation between events in 1971 and 2010.
Ambiguous Claim	**Quantitative Ambiguity:** "There were 331 episodes of the TV series where Julianna Margulies had the role of Carol Hathaway." Misinterpreted as implying her appearance in all 331 episodes.

Prompts for tree planner generation

You are an expert at analyzing paragraphs and creating structured verification graph for fact-checking. Given a paragraph that contains multiple sentences, you will generate a JSON verification graph that exactly represents the logical structure on how to perform verification on the paragraphs. The verification focuses on the entities, statistics, and temporal information.
Your Task
Given:
1. A natural language paragraph.
Generate a structured verification graph in JSON format that:
* Captures **every logical reasoning** steps to verify the input. * Tracks **dependencies** between steps. * The resulting graph must be a **Directed Acyclic Graph** * Before writing the JSON, you can think as long as needed. * Please encapsulate the JSON output inside “‘ “‘
—
Node Structure Each node will have these keys: 1. **id**: The node identifier 2. **input**: The input of the node. Different type of node has different definition of input. 3. **hints**: A string contains hint on what this node is about, become explanation why we need this node and how this node connect to the other nodes on the verification graph. 4. **dependencies**: List of node ids indicates that what other nodes this node depends on. The verification graph should respect this dependencies. It means, to perform accurate verification of this node, we need to finish the execution of all nodes in this list.
Node Types
There are **four node types**, where each type has different action to execute.
1. **Search Node (‘s_i‘)** : Node that tasked to perform search operation. * Root nodes in the verification graph should be search node. * But search node can be positioned in the middle of the node. * The **input** is a query string, indicates information required to gather for the following steps. 2. **Refine Node (‘r_i‘)**: Node which input is not verbosely clear that needs to be refined before verification can be performed. * Usually the **input** contains ambiguous pronouns or entity. * The **input** refers to the user original paragraph, but it is not clear and search operation must be done prior to this node. * **MUST** have dependencies, so we can leverage the information from its dependency nodes to resolve the input. 3. **Think Node (‘t_i‘)**: Node to generate intermediate reasoning. * The **input** is a string contains which information from the input paragraph needs to be verified. Make sure the input is very clear and do not contains ambiguous references. * The **input** is empty string, if it comes from **Refine Node**, because we will set the **input** with the output from the **Refine Node**. * **MUST** have dependencies, so we can leverage the information from its dependency nodes to resolve the input. * This node will combine all information from its dependencies, and perform reasoning about the **input**. 4. **Judge Node (‘j_i‘)**: Node to judge the input and generate the label and explanation * The **input** is a string contains which information from the input paragraph needs to be verified. Make sure the input is very clear and do not contains ambiguous references. * The **input** is empty string, if it comes from **Refine Node**, because we will set the **input** with the output from the **Refine Node**. * **MUST** have dependencies, so we can leverage the information from its dependency nodes to resolve the input. * This node will combine all information from its dependencies, and judge the input to 3 labels: **SUPPORT**, **REFUTE**, or **NOT CERTAIN**

Table 4: Tree plan generation prompt

Prompts for tree planner generation-continue

Graph Management * Graph must be a Directed-Acyclic Graph. * The graph can contains multiple root nodes. * The root nodes must be a search node, to provide information for the thinking nodes in the later steps. * **Search Node**, **Think Node** and **Judge Node* inputs **MUST** be understandable on its own. If it contains ambiguous references or unclear time, you need to define **Refine Node** prior to this node. If necessary, you might also need **Search Node** before the **Refine Node**. * But **Search Node** not always in the root nodes. Minimize the **Search Node** in the root to minimize the number of search operation. * A **Think node** and **Refine Node** **MUST** has dependencies, because a think node needs information obtained from the previous nodes to perform reasoning. * Multiple nodes can direct to the same node, indicates that node combination occurs. Nodes are combined if and only if there are logical relationships between nodes to combine. The logical relationships involves but not limited to: * Combination: Combining multiple information into 1 node. * Comparison: Comparing some information from parent nodes. * Causality: There is a causal-effect relationships between parent nodes and this node. * Inductivity: This node become a general conclusion from specific information contained by the parent nodes. * Deductivity: This node becomes a specific information from general information contained by the parent nodes. ——
Critical Requirements
1. Contextual and Temporal Understanding * You should understand the context and the time of the input to perform verification. * You should focus on the factuality of the data statistics, numbers, entities, or time of the input. * Sometimes the user do not state the time or statistics explicitly. In that case, you need to perform search operation to find those information. * Once you understand the input and the contexts, you can list all information that need to be verified. * This list can become the starting node of the verification graph.
2. Complete Coverage * The verification graph should verify every piece of information on the input * Perform search if necessary to be able to perform accurate verification. * Be creative and explorative: * Do not make assumption. If it is not clear, you need to first perform the search operation before continue the verification process. * Nodes are connecting with each other with an edge, for either these purposes: * Sending evidence from search node. * Combine information from multiple nodes and perform intermediate reasoning over the information contains in the node.
3. Coherent, and Structural * The verification graph must outline the necessary steps to accurately determine the veracity (truthfulness) of the claim. * Each node treated as a step. Each step must be a self-contained, that, when answered or verified, logically contributes to proving or disproving the main Claim.
4. Uniqueness & Non-Redundancy: * Each step should aim to provide unique information to the claim and avoid redundancy.
5. Clarity * Avoid using vague language such as ambiguous pronouns, vague references, or incomplete names. Make sure each steps clearly stating the complete name entity. * If, you want to create a node, and the input refers to some ambiguous pronouns, vague references to the entities, or events, you **MUST** perform "Refine Node" first to resolve those ambiguous texts. There are 2 options: * If you think search is required: (Search Node) -> (Refine Node) -> (This Node) * If you think search is not requred: (Refine Node) -> (THIS NODE)
* For instance, the node's input is "... Donald Trump's falsifying business records case" and the type is **Search Node**. This is still vague references. Therefore, we first need to have **Search Node** on "Donald Trump's cases on falsifying business records", then **Refine Node** to refine the input, and then you can perform the **Search Node** on this input.

Prompts for tree planner generation-continue

Output Format “‘json ["id": "s_1", "input": "[query_search]", "hints": "To verify the paragraph, we need to know the exact number of ...", "dependencies": [] , "id": "s_2", "input": "[query_search]", "hints": "To verify the paragraph, we also need to know the time period of ...", "dependencies": ["tc_1"] , "id": "r_2", "input": "input text", "hints": "Since the input says 'that period' we need to search the correct time period and replace the text with the correct period", "dependencies": ["s_2"], ,
"id": "t_1", "natural_language": "input text", "statement": "To understand the input, we need the information of the number and the time period", "dependencies": ["s1", "r_2"] , "id": "j_1", "natural_language": "input text", "statement": "Based on the information obtained from the dependencies node, we now can verify the factuality of the input text", "dependencies": ["t1"]] “‘ **Very important** * **ID reference format**: Use [s_1], [s_2], for search node, [r_1] for refine node, [t_1] for think node, and [j_1] for judge node. * **Consistency rule:** All content from nodes in ‘"dependencies"‘ must be restated with their corresponding IDs in the assumptions list of the ‘statement‘.

Prompts for SEARCH Node

You are an expert at analyzing text for finding relevant information using Search Engine.
Given a text and a hint that what information we need from the text, you will generate a JSON output with key "QUERIES" that contains query search string to be executed against a search engine to ensure the required information is obtained.
Please combine the answer and the query search into a natural sentences. Please DO NOT GENERATE anything else.
Output Format:
{"QUERIES": [query_search_1, query_search_2, ...],}

Prompts for THINK node

You are an fact-checking expert that excel at analyzing a claim and understand the current stance based on provided evidence.
You are given a claim, and right now, you are in the middle of the verification process. Now, you are also given an input that is derived from the text, and a hint to explain how this input can help to verify the claim. Your task is to analyze the claim, the input, and the hint. Then, examine the input against the evidence, and provides your stance. You will generate a string about your explanation of the input, and what is the logical relationship between the input, and the evidence. If you think the input is answerable given the information, please write "i dont know".

Prompts for REFINE node

You are an fact-checking expert that excel at analyzing a text. You are given an input. The input contains ambiguous references, pronouns, or unclear entities. We have tried to perform search operation in prior to resolve those ambiguity. Your task is to analyze the input and the provided search results, then generate a new input that resolve those ambiguity. The new input MUST have similar meaning as the original input, without any ambiguous references, pronouns, or unclear entities.
You will generate a JSON contains "NEW_INPUT": a string contains the input without ambiguous references, pronouns, or unclear entities. Please DO NOT GENERATE anything else.
Output format: {"NEW_INPUT": "new_input"}

Prompts for JUDGE node

You are an fact-checking expert that excel at analyzing a claim and understand the current stance based on provided evidence. You are given a claim, and right now, you are at the end of verification process to examine the truthfullness of the input. Now, you are also given an input that is derived from the text, and a hint to explain how this input can help to verify the claim. Your task is to the input, with the help the hint and the context. Then, examine those against the evidence, and provides your label. There are 3 possible labels, Supports, Refutes, and Uncertain.
1. If you are confident that all information in the claim is True, return Supports.
2. If you are confident that the information in the claim contradict with the evidnece, return Refutes.
3. If you are not certain how to label the claim, return Uncertain
You will generate a JSON contains "EXPLANATION": a string about your explanation of the input, and what is the logical relationship between the claim, the input, and the evidence. If you think the input is not answerable given the information, please write "i dont know". "LABEL": either Supports, Refutes, or Uncertain. Please DO NOT GENERATE anything else.
Output format: { "EXPLANATION": "explanation", "LABEL": "The label, either Supports, Refutes, or Uncertain" }