

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

Anab Maulana Barik*, Shou Ziyi, Yang Kaiwen, Yang Qi, Shen Xin[†]

Huawei Celia Team

maulanaanab@gmail.com, shenxin19@huawei.com

Abstract

Technological advancement allows information to be shared in just a single click, which has enabled the rapid spread of false information. This makes automated fact-checking system 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 faces two limitations: they often fail to verify every component in the claim and lack of 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 structured these actions into a dependency graph to model the logical interaction 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 system.

1 Introduction

Technological advancements have fundamentally changed how online media operates. The rise of internet and social media platforms allow 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 system.

Automated fact-checking systems have evolved significantly over the past decades. Early approached (Hanselowski et al., 2018; Zhou et al.,

2019; Liu et al., 2020; Zhong et al., 2020) typically followed a standard three-step pipelines: (1) document retrieval for retrieving relevant document, (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 leverages 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 decompositions, retrieval method, 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 two critical shortcomings, as shown in Figure 1. First, they frequently fail to perform complete verification on every component within a complex claim, often because of the 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-tasks because they often treat them as isolated steps rather than interdependent nodes in a reasoning graph. This prevents the agents to perform logical and more comprehensive reasoning, leads 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 structural verification process called Trification. Unlike earlier systems that generate sub-tasks sequentially (Ma et al., 2025; Zhao et al., 2024; Xu et al., 2024) – a process prone to premature termination, our framework begins by generating a complete set of verification actions. This approach ensures that

* This work was done during an internship at Huawei Celia Team.

[†] Corresponding author

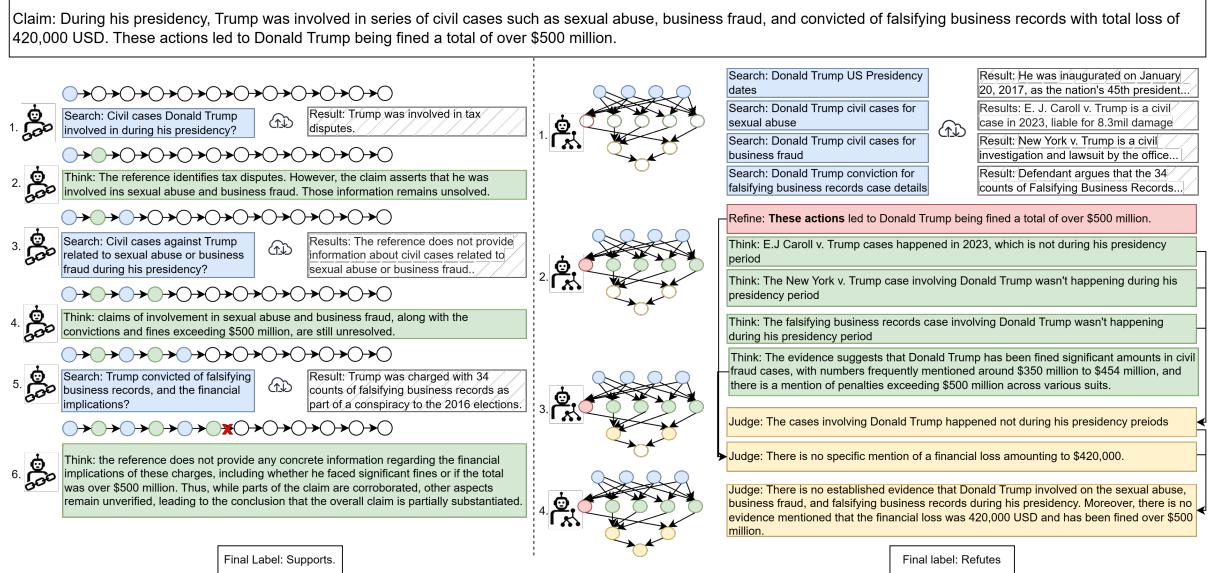


Figure 1: Case Comparison: Chain Search vs. Tree Verification

every part of the claim is covered from the set. Our framework then structures these actions into a dependency graph, which provides two key advantages. First, in contrast to previous methods that combine all sub-tasks’ results at the final step (Pan et al., 2023; Zhao et al., 2024), 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 a sequential process.

Moreover, to prevent our system to end prematurely or produce 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 systems adaptability, mirrors the creative approach of human fact-checkers.

In summary, our main contributions are

1. We propose Trification, a novel fact-checking framework that uses a tree-based strategy to generate a comprehensive set of verification actions, to ensure 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 demonstrate a graph modification mechanism that allows the system to adapt its strategy upon failing, enhancing the creativity to explore alternative verification strategy.

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 evolution fundamentally

underscores the critical importance of structured evidence processing for robust fact-checking but 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, where 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 incorporating 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 framework starts by constructing an initial Directed Acyclic Graph (\mathcal{G}), where nodes representing verification actions, and a set of edges defining 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. Its input is a query string, which represents the

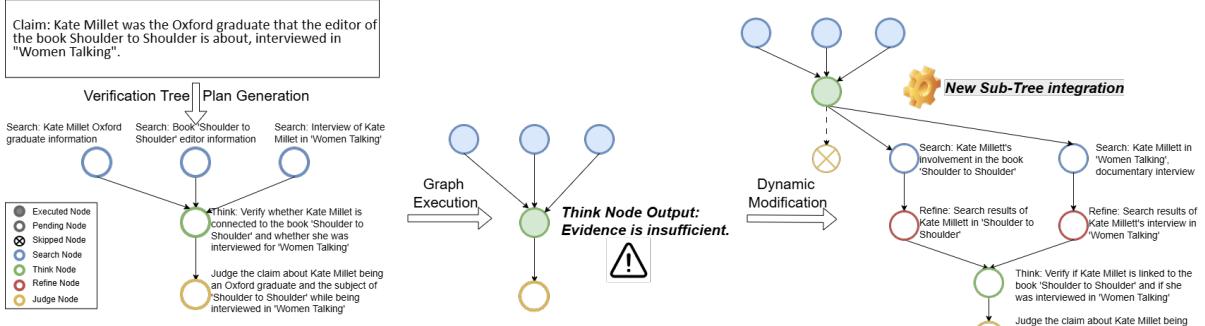


Figure 2: Dynamic Tree Planner: insufficient evidence triggers dynamic graph modification and integration of a newly generated subtree

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 query search based on the node’s input and hints. The query search is conditioned on the node’s input and strategic hints, ensuring it retrieves information relevant to downstream nodes in the verification graph.

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, passing the query search and collecting 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. Evidence propagation: Evidence and output

¹DDGS: <https://github.com/deedy5/ddgs>

from parent nodes is 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. Refine information Update: Check if one of the parent node is REFINE Node. If yes, we need to replace the input node with the output from that REFINE 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 is failing, we cannot perform accurate execution for their descendant. 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 (C)
Output: Label
Generate a Directed Acyclic Graph verification plan  $\mathcal{G}$ 
Concurrently execute nodes in  $\mathcal{G}$  following topological order
for each node  $v \in \mathcal{G}$  do
    Let  $v_{orig}$  be the intrinsic input associated with node  $v$ 
    Let  $Parents(v) = \{u \mid (u \rightarrow v) \in \mathcal{G}\}$ 
    Collect parent outputs:
         $v_{parents} = \{\text{output}(u) : u \in Parents(v)\}$ 
    Combine intrinsic input and parent outputs:
         $v_{input} = \text{Combine}(v_{orig}, v_{parents})$ 
    if type( $v$ ) = SEARCH then
        output( $v$ )  $\leftarrow$  Search( $v_{input}$ )
    else if type( $v$ ) = REFINE then
        output( $v$ )  $\leftarrow$  Refine( $v_{input}$ )
    else if type( $v$ ) = THINK then
        output( $v$ )  $\leftarrow$  Reason( $v_{input}$ )
        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
        output( $v$ )  $\leftarrow$  Judge( $v_{input}$ )
        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 3 complexity levels: 1,126 two-hop, 1,835 three-hop, and 1,039 four-hop claims. The FEVEROUS dataset consist 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) leverage RoBERTa-large and DeBERTaV3 model and fine-tuned on FEVER (Thorne et al., 2018) dataset and 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 uses LLM through prompting technique. This includes using zero-shot prompts using ChatGPT (Zhao et al., 2024) or few-shot prompts 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 method goes beyond standard LLM-agent based 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.

4.3 Implementation Details

We employ a high performance model 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 per-

formance. 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.

Unless otherwise specified, results for our Trification framework are reported using the **Static Mode** to ensure a direct comparison with prior work that uses static knowledge bases. Due to the computational and API costs associated with live search engine queries and iterative graph modifications, the results for the **Dynamic Mode** (presented in subsection 5.2) are reported on a 150-claim subset of each test set.

5 Experimental Results

5.1 Main Results

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 trends in the improvements underscores the importance of how 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 level, 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.

An interesting findings from the table is that LLM agents with dynamic planner underperform

Models	HOVER			FEVEROUS	AVERAGE
	2-hop	3-hop	4-hop		
I	BERT-FC	50.68	49.86	48.57	50.20
	LiST5	52.56	51.86	50.46	52.27
	RoBERTa-NLI	63.62	53.99	52.40	56.95
	DeBERTaV3-NLI	68.72	60.76	56.00	61.07
II	MUTIVERS	60.17	52.55	51.86	55.30
	ChatGPT	66.94	60.56	58.73	60.49
	Codex	65.07	56.63	57.27	60.39
III	F1lan-T5	69.02	60.23	55.42	62.10
	ProgramFC (N=1)	69.36	60.63	59.16	67.80
	ProgramFC (N=5)	70.30	63.43	57.74	68.06
IV	PACAR	73.13	64.07	<u>63.82</u>	<u>72.61</u>
	LoCaL	72.71	<u>64.11</u>	61.59	68.22
	SearChain	64.46	60.30	56.54	66.69
Our	Trification	75.13	66.42	66.23	74.72
	w/o REFINE	71.22	61.56	60.99	71.30
	w/o REFINE and THINK	65.64	55.10	55.20	70.58

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.

compared to agents with a fixed plan. This suggests that unlimited dynamic planning can be detrimental without a guiding structure, potentially leading to unfocused and inefficient iteration over sub-claims. Our framework addresses this weakness by generating a comprehensive set of actions and structuring them into a DAG, thereby providing the benefits of dynamic adaptation within a controlled, goal-oriented framework, which ultimately leads to superior performance.

5.2 Results of Trification in Dynamic Settings

Due to the significant computational overhead and API costs for performing live search engine queries and dynamic graph modifications, the evaluation of the **Dynamic Mode** was conducted on a curated subset of 150 challenging claims from each test set. These ‘Hard’ samples are defined as claims that the **Static Mode** could not verify correctly.

The results in Table 2 demonstrate that our dynamic method substantially outperforms the static approach on this challenging subset. This performance gain is more pronounced with the increasing claim complexity levels: 4-hop claims show a 50% improvement, nearly double the 28% gain for 2-hop claims (see Table 4). This outcome is achieved efficiently. For the final results, we observe that the number of Uncertain predictions decreases under the dynamic tree setting, indicating that the dynamic strategy is effective. On average, our framework required 1.92 graph modifications on HOVER and 2.13 on FEVEROUS. The number

of SEARCH nodes increases slightly ($2.74 \rightarrow 3.51$ on HOVER and $2.64 \rightarrow 3.60$ 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 $1.52 \rightarrow 1.88$ on HOVER and $1.67 \rightarrow 2.16$ on FEVEROUS, while REFINE nodes increase from $0.36 \rightarrow 0.65$ and $0.12 \rightarrow 0.42$, respectively. Likewise, JUDGE nodes grow from $1.09 \rightarrow 1.53$ on HOVER and $1.01 \rightarrow 1.53$ 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 one second slower on average compared to the static tree. At the node level, REFINE nodes appear significantly more frequently. 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

	HOVER	FEVEROUS
Graph Modification	1.92%	2.13%
Macro-F1	+40.0%	+60.3%
SEARCH Node Increased	$2.74 \rightarrow 3.51$	$2.64 \rightarrow 3.60$
THINK Node Increased	$1.52 \rightarrow 1.88$	$1.67 \rightarrow 2.16$
REFINE Node Increased	$0.36 \rightarrow 0.65$	$0.12 \rightarrow 0.42$
JUDGE Node Increased	$1.09 \rightarrow 1.53$	$1.01 \rightarrow 1.53$

Table 2: Dynamic Tree Planner Average Performance on 150 hard samples from HOVER and FEVEROUS datasets. Hard samples are those our static planner cannot predict correctly.

with a search engine to explore alternative verification paths for complex reasoning tasks.

Collectively, these findings highlight the effectiveness 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% (7/10) of the errors originated from the search engine component. Specifically, four cases (4/7) were due to the search engine retrieving evidence that *contradicted* the ground truth, while the remaining three cases (3/7) 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 retrieved evidence confirmed that Arthur 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 the role in that TV series.

Developing post-processing strategies to handle conflicting search results or disambiguate unclear claims remains a challenge for future work.

6 Conclusion

We presented Trification, a novel automated fact-checking framework that ensures complete claim coverage through a comprehensive set of verification actions organized in a dependency graph. This graph models logical interactions and can be dynamically modified to adapt the verification strategy. Experiments on two challenging benchmarks show that Trification significantly improves fact-checking accuracy over previous methods. Future work includes handling conflicting

search results and resolving ambiguities in claims to further enhance robustness and reliability.

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 pre-training 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

Table 3 shows the error analysis for Trification.

A.2 Detail Results of Trification Dynamic Settings

Table 4 presents the detailed performance results of dynamic Trification.

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 <i>is</i> 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.

Table 3: Representative failure cases from error analysis.

Models	HOVER			FEVEROUS	AVERAGE
	2-hop	3-hop	4-hop		
Trification	00.00	00.00	00.00	00.00	00.00
Trification-dynamic	28.00	42.00	50.00	60.26	45.07

Table 4: Performance on 150 hard samples of HOVER and FEVEROUS; Hard samples = Our static planner cannot predict this correctly