

A Graph-based Verification Framework for Fact-Checking

Yani Huang¹, Richong Zhang^{1,2*}, Zhijie Nie¹, Junfan Chen³, Xuefeng Zhang¹

¹CCSE, School of Computer Science and Engineering, Beihang University, Beijing, China

²Zhongguancun Laboratory, Beijing, China

³School of Software, Beihang University, Beijing, China

{huangyn, zhangxf}@buaa.edu.cn

{zhangrc, niezj, chenjf}@act.buaa.edu.cn

Abstract

Fact-checking plays a crucial role in combating misinformation. Existing methods using large language models (LLMs) for claim decomposition face two key limitations: (1) *insufficient decomposition*, introducing unnecessary complexity to the verification process, and (2) *ambiguity of mentions*, leading to incorrect verification results. To address these challenges, we suggest introducing a claim graph consisting of triplets to address the insufficient decomposition problem and reduce mention ambiguity through graph structure. Based on this core idea, we propose a graph-based framework, GraphFC, for fact-checking. The framework features three key components: *graph construction*, which builds both claim and evidence graphs; *graph-guided planning*, which prioritizes the triplet verification order; and *graph-guided checking*, which verifies the triples one by one between claim and evidence graphs. Extensive experiments show that GraphFC enables fine-grained decomposition while resolving referential ambiguities through relational constraints, achieving state-of-the-art performance across three datasets.

1 Introduction

Fact-checking plays a crucial role in detecting misinformation and preventing the spread of rumors. The recent emergence of LLMs, which exhibit powerful semantic understanding capabilities, has opened up new potential solutions for fact-checking. Leveraging these LLMs, recent methodologies (Pan et al., 2023; Wang and Shu, 2023; Zhao et al., 2024) have introduced approaches that decompose claims into textual sub-claims. Such LLM-driven claim decomposition simplifies the verification process and allows for more precise identification of errors within the claim.

Despite their impressive performance, these decomposition-based methods still face two main

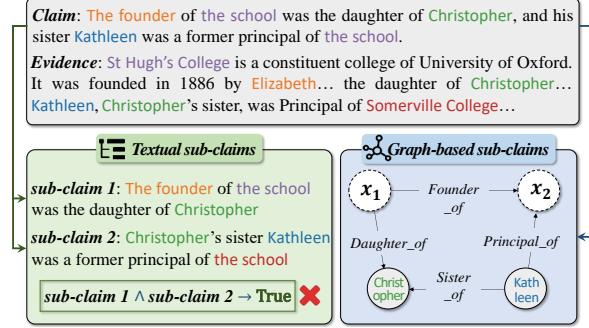


Figure 1: An illustration demonstrating the differences between existing textual sub-claims and our graph-based sub-claims. The example is from the HOVER dataset (Jiang et al., 2020), and the decomposition results are from GPT-3.5-Turbo.

challenges. First, without explicit rules, existing models decompose claims into isolated textual sub-claims, each maybe containing multiple subjects, relations, or objects. This decomposition method leads to *insufficient decomposition* problem, introducing unnecessary complexity to the claim verification process. As in Figure 1, sub-claim 1 comprises two distinct facts: one between “the founder” and “the school”, while the other between “the founder” and “Christopher”.

Furthermore, there may be multiple unknown mentions within the claim text, and the same unknown mentions could be shared across different sub-claims. By focusing solely on individual sub-claims, this decomposition method increases the *ambiguity of mentions*, as the shared unknowns are not contextualized within a unified framework and neglects the broader context and interconnections, thereby amplifying the uncertainty associated with these unknown mentions. As in Figure 1, “the school” in sub-claim 2 presents a referential ambiguity. Fact-checking models may incorrectly associate “the school” with “Somerville College” due to an inadequate consideration of the interdependencies between sub-claims.

*Corresponding author.

To overcome the aforementioned limitations, we suggest converting the claim text into the form of graph using <subject, relation, object> triplets. The graph contains two types of entities: the known entities (the ground entities in the claim) and the unknown entities (the entities existing as references or relationships pending resolution). Using this decomposition method, each triplet can be viewed as a minimal unit that cannot be further decomposed, so the problem of *insufficient decomposition* is eliminated. The graph structure also preserves all contextual information from the original claim, thus preventing *ambiguity of mentions*.

Building on such a foundation, we propose GraphFC, a graph-based fact-checking framework consisting of three components: (1) *graph construction*, besides the claim graph, an evidence graph for each claim is constructed correspondingly, inspired by the observation that irrelevant context impairs verification accuracy from recent research (Guan et al., 2024). (2) *graph-guided planning*, prioritizes the validation of triplets and determines the logical sequence for checking each triplet. and (3) *graph-guided checking*, which comprises two sub-tasks: *graph match*, which verifies the accuracy of known entity triplets, and *graph completion*, which infers incomplete entities for triplets with unknown entities.

Empirical studies employing closed-source and open-source LLMs, such as GPT-3.5-Turbo and Mistral-7B, along with widely-used fact verification datasets including HOVER (Jiang et al., 2020), FEVEROUS (Aly et al., 2021), and SciFact(Wadden et al., 2020), demonstrate the effectiveness of our proposed framework.

To summarize, the contribution of this study is three-fold.

- We introduce the graph structure to represent the complex claims, effectively addressing the insufficient decomposition problem and reducing mention ambiguity.
- We propose a graph-based fact-checking framework (GraphFC) with three components: *graph construction*, *graph-guided planning*, and *graph-guided checking*.
- We validate GraphFC’s effectiveness and rationality through experiments on three public fact-checking datasets, achieving state-of-the-art results.

2 Background

Fact Checking Given the natural-language claim C , fact-checking aims to find a model M_f that predicts the veracity label $Y \in \{\text{True}, \text{False}\}$ of the claim C based on the provided evidence E , which can be expressed as

$$M_f(C, E) \rightarrow Y. \quad (1)$$

Depending on how E is accessed, fact-checking is usually categorized into two types: (1) Gold Evidence setting, where E is directly given, and (2) Open Book setting, where E needs to be retrieved from a specified knowledge source \mathcal{K} .

PLM-based Fact Checking In the past few years, pre-trained language models have demonstrated strong performance in fact-checking. BERT-based approaches (Soleimani et al., 2020; Gi et al., 2021) effectively combine evidence retrieval and claim verification. Graph-based methods further enhance reasoning capabilities by modeling relationships between evidence pieces. GEAR (Zhou et al., 2019) employs evidence aggregation through graph networks, while Transformer-XH (Zhao et al., 2020) introduces extra hop attention for multi-evidence reasoning.

LLM-based Fact-Checking The emergence of LLMs has revolutionized fact-checking through their powerful reasoning capabilities. Chain-of-Thought (CoT) prompting (Wei et al., 2022) facilitates decomposing complex verification into intermediate steps, while self-consistency (Wang et al., 2022) boosts accuracy via multiple reasoning paths. Recent work has focused on developing specialized decomposition-based prompting strategies for fact verification. HiSS (Zhang and Gao, 2023) proposes a hierarchical approach that decomposes claims into verifiable sub-claims, while ProgramFC (Pan et al., 2023) introduces program-guided reasoning to enhance verification accuracy. PACAR (Zhao et al., 2024) combines planning with customized action reasoning. FOLK (Wang and Shu, 2023) introduces a first-order logic-guided method for claim decomposition. However, these decomposition-based methods suffer from insufficient decomposition and mention ambiguity. Our work addresses these limitations by introducing a claim graph built from fine-grained sub-claims that preserves shared unknown mentions via graph structure.

3 Method

3.1 Fact-Checking over Graphs

Task Definition We first give a graph-style definition for fact-checking here. Concretely, the natural language claim C is converted into a directed graph $G_c = \{(s, r, o), s, o \in \mathcal{E}_c \cup \mathcal{X}_c, r \in \mathcal{R}_c\}$, where \mathcal{E}_c is a known entity set, \mathcal{X}_c is the unknown entity set and \mathcal{R}_c is the relation set. Specifically, \mathcal{E}_c contains the named or disambiguated real-world entities, which are determined when decomposing the claim. In contrast, the entity in \mathcal{X}_c can not be determined according to claim only, as a claim may contain ambiguous references (e.g., pronouns, indefinite references) that cannot be immediately resolved through simple decomposition; instead, these entities require contextual understanding or additional inference steps to be determined. Similarly, the evidence set E is organized into an evidence graph $G_e = \{(s, r, o), s, o \in \mathcal{E}_c, r \in \mathcal{R}_c\}$, inspired by the finding of irrelevant evidence impair verification accuracy (Guan et al., 2024). Note that we define the nodes in G_e to contain only the known entities since evidence usually comes from knowledge bases, which are written with a clear presentation of the facts, with few ambiguous references. Finally, we give the sufficient and necessary conditions for a claim to be supported by evidence.

Assertion 1 A claim G_c is supported by evidence G_e if and only if there exists a one-to-one function $\phi : \mathcal{E}_c \cup \mathcal{X}_c \rightarrow \mathcal{E}_e$ satisfies the following two conditions at the same time:

$$\begin{aligned} p_1 : & \forall x \in \mathcal{E}_c, \phi(x) = x. \\ p_2 : & \forall x, y \in \mathcal{E}_c \cup \mathcal{X}_c, \forall r \in \mathcal{R}_c, \\ & (x, r, y) \in G_c \Rightarrow (\phi(x), r, \phi(y)) \in G_e. \end{aligned}$$

Based on this definition, we summarize the benefits of graph-based verification as follows: (1) is performed at the ternary level, preventing the problem of insufficient decomposition; (2) any node in the claim graph corresponds to a unique node in the evidence graph, which satisfies all the relation constraints on the node at the same time, avoiding the ambiguity of mentions.

GraphFC While Assertion 1 gives a clear idea of verification, converting it into a practical verification framework requires more effort. To validate our idea, we propose a fact-checking framework called **GraphFC** for systematically verifying claims through graphs. As shown in Figure

2, GraphFC consists of three primary components: *graph construction*, *graph-guided planning*, and *graph-guided checking*.

3.2 Graph Construction

In graph construction, we transform the claim C and the corresponding evidence E into the graph structures. A specialized graph construction agent \mathcal{A}_{gc} is introduced to complete this complex task.

Claim Graph Construction Claim graph construction aims to transform a natural language claim C to a graph G_c . The graph construction agent \mathcal{A}_{gc} is expected to extract all triplets from claim text to complete this graph construction, which can be denoted as

$$G_c = \{t_1^c, \dots, t_{n_c}^c\} = \mathcal{A}_{gc}(C) \quad (2)$$

where t_i^c represents the i -th triplet extracted from the claim text and n_c represents the extracted triplet number. In practice, the agent \mathcal{A}_{gc} is provided with a task description, an extraction guideline and K fixed in-context examples for generating triples that conform to a particular format. In particular, the unknown entities that appear in the claim are prompted to be replaced with placeholders like x_i to form the triplet. Please refer to Listing 3 to check the prompt in detail. For example, the triplets extracted from the claim in Figure 2 are shown as follows:

$t_1^c: <x_1, \text{Daughter_of}, \text{Christopher}>$
$t_2^c: <x_1, \text{Founder_of}, x_2>$
$t_3^c: <\text{Kathleen}, \text{Sister_of}, \text{Christopher}>$
$t_4^c: <\text{Kathleen}, \text{Principal_of}, x_2>$

Evidence Graph Construction Evidence graph construction aims to convert the natural language evidence E to a graph G_e . The agent \mathcal{A}_{gc} in evidence graph construction process identifies entities in evidence set E that are related to the known entities in \mathcal{E}_c and extract the related triplets, which is denoted as

$$G_e = \{t_1^e, \dots, t_{n_e}^e\} = \mathcal{A}_{gc}(E, \mathcal{E}_c) \quad (3)$$

where t_i^e represents the i -th triplet extracted from the evidence text and n_e represents the extracted triplet number. In practice, the agent \mathcal{A}_{gc} is provided with similar prompts and K in-context examples as that for the construction of claim graphs. Please refer to Listing 4 to check the prompt in detail. Even with constraints on the entity set \mathcal{E}_c ,

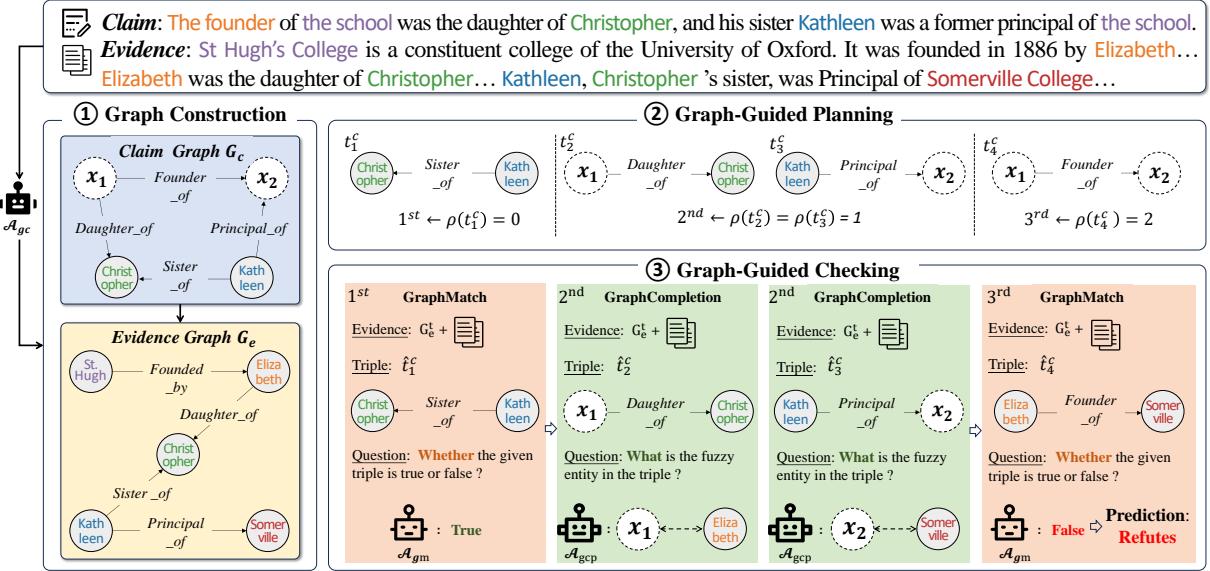


Figure 2: Overview of GraphFC. The framework consists of three components: (1) *graph construction* (&3.2), where claim graph and corresponding evidence graph are constructed for each claim; (2) *graph-guided planning* (&3.3), which determines the verification order based on unknown entity count and plans the sequence of verification tasks; and (3) *graph-guided checking* (&3.4), which executes either *graph match* for verifying the known entity triples or *graph completion* for inferring incomplete entity.

there are still far more triples extracted from evidence than claims, but usually, only a few of them are useful for fact verification. To improve readability, Figure 2 only shows the useful triples, whose output format is as follows:

$t_1^e: <\text{St Hugh's College}, \text{Founded_by}, \text{Elizabeth}>$
 $t_2^e: <\text{Elizabeth}, \text{Daughter_of}, \text{Christopher}>$
 $t_3^e: <\text{Kathleen}, \text{Sister_of}, \text{Christopher}>$
 $t_4^e: <\text{Kathleen}, \text{Principal_of}, \text{Somerville}>$

3.3 Graph-Guided Planning

The triplets $\{t_1^c, \dots, t_{n_c}^c\}$ obtained from the claim graph construction are unordered. If the claim graph does not contain unknown entities (i.e., $\mathcal{X}_c = \emptyset$), it is feasible to verify the triples one by one according to any order. However, due to the presence of unknown entities, the order of validation should be planned because the triple like $\langle x_1, \text{Founder_of}, x_2 \rangle$ will be difficult to verify before x_1 and x_2 are grounded based on known entities. In GraphFC, graph-guided planning is denoted as a sorting algorithm \mathcal{S}_{gp} , which is expressed as

$$\mathcal{T} = [\hat{t}_1^c, \dots, \hat{t}_{n_c}^c] = \mathcal{S}_{gp}(G_c) \quad (4)$$

where \hat{t}_i^c denotes the i -th triplet to be verified in the subsequent step of fact-checking. Specifically, \mathcal{S}_{gp} sorts the triplets of the claim graph by the unknown entity number in the triplet. For any triplet

$t = (s, p, o)$, we use $\rho(t)$ to present its priority for verification, which can be calculated by

$$\rho(t) = \begin{cases} 2 & \text{if } s \in \mathcal{X}_c \text{ and } o \in \mathcal{X}_c \\ 0 & \text{if } s \notin \mathcal{X}_c \text{ and } o \notin \mathcal{X}_c \\ 1 & \text{otherwise} \end{cases} \quad (5)$$

where the smaller value of $\rho(t)$ represents the higher priority for verification. The principle behind prioritizing in this way is summarized here: firstly, verify the triplet of two known entities only ($\rho = 0$); then, ground all unknown entities from the triplet of a known entity and an unknown entity ($\rho = 1$); finally, replace the unknown entities with the ground truth and verify the triplets of two unknown entities ($\rho = 2$). After obtaining the priority for each triplet, we sort all triplets in the claim graph based on their priorities in descending order. The order between two triplets with the same priority will be organized randomly. Therefore, a possible ordering for the triples of the claim graph in Figure 2 is as follows:

$\hat{t}_1^c: <\text{Kathleen}, \text{Sister_of}, \text{Christopher}>$
 $\hat{t}_2^c: <x_1, \text{Daughter_of}, \text{Christopher}>$
 $\hat{t}_3^c: <\text{Kathleen}, \text{Principal_of}, x_2>$
 $\hat{t}_4^c: <x_1, \text{Founder_of}, x_2>$

3.4 Graph-Guided Checking

Based on the sorted triple list \mathcal{T} in Equation 4, we conduct a fact-checking process for each triplet

in the claim graph G_c . In general, we expect to generate a binary label $Y_t = \{\text{True}, \text{False}\}$ for each triplet t , and the fact-verification final label Y with respect to the claim C satisfies:

$$Y = Y_{\hat{t}_1^c} \wedge \cdots \wedge Y_{\hat{t}_{n_c}^c} \quad (6)$$

To achieve this goal, we employ two specialized components: a graph match agent \mathcal{A}_{gm} for verifying the triple with two known entities and a graph completion agent \mathcal{A}_{gcp} for verifying the triple with only one known entity and resolving the other unknown entity. Note that \mathcal{A}_{gcp} will keep replacing unknown entities with known entities in the evidence graph. Therefore, as shown in Figure 2, all triplets originally with two unknown entities will be updated to have one or two known entities, and \mathcal{A}_{gm} or \mathcal{A}_{gcp} can therefore solve them. We also give an algorithm-style fact-checking process in Appendix A.

Graph Match When there is no unknown entity in the triplet t , we perform the process of graph match on it. Specifically, The graph match agent \mathcal{A}_{gm} aims to verify whether the triple t is supported by the evidence and output a binary label $Y_t = \{\text{True}, \text{False}\}$. Considering t only needs to match with the related triplets in G_e , we filter unrelated triplets from G_c to improve verification efficiency. Consequently, the triplets containing the same known entities in t remain for verification, which is denoted as $G_e^{(t)}$. In addition, we empirically find that adding original evidence text E as input will improve fault tolerance (See the analysis in Section 4.3.1 for details). Therefore, the verification process of \mathcal{A}_{gm} can be expressed as

$$Y_t = \mathcal{A}_{gm}(t, G_e^{(t)}, E) \quad (7)$$

In practice, \mathcal{A}_{gm} is provided with a task description and an output-format instruction only, without the in-context examples. Taking t_1^c in Figure 2 as an example, the simplified input and output content are shown as follows:

```
# Input (E is omitted)
t1c: <Kathleen, Sister_of, Christopher>

Ge(t1) : {
    t2e: <Elizabeth, Daughter_of, Christopher>
    t3e: <Kathleen, Sister_of, Christopher>
    t4e: <Kathleen, Principal_of, Somerville>
}

# Output
t1c ≡ t3e ⇒ Yt1 = True
```

Graph Completion When there is only one unknown entity in the triplet t , the graph completion agent \mathcal{A}_{gcp} aims to ground the unknown entity x in the triple and output (1) an entity $e \in \mathcal{E}_e$ to replace x in all triplets of claim graph G_c and (2) a binary $Y_t = \{\text{True}, \text{False}\}$. Indicating the success or failure of unknown entity completion, Y_t can be simply expressed as $Y_t = (e \neq \text{None})$. Similar to \mathcal{A}_{gm} , we use the filtered related triplets $G_e^{(t)}$ and the original evidence text E as the input. Finally, the completion process of \mathcal{A}_{gcp} can be written as

$$e, Y_t = \mathcal{A}_{gcp}(t, G_e^{(t)}, E) \quad (8)$$

If e exists, the unknown entity x of the subsequent triplets should be replaced by e . Without loss of generality, suppose that \mathcal{A}_{gcp} finds the corresponding known entity e for x in the i -th triplet \hat{t}_i^c , for any $i < j \leq n_c$, j -th triplet $\hat{t}_j^c = (s, r, o) \in \mathcal{T}$ also needs to replace x in it with e :

$$\hat{t}_j^c = \begin{cases} (e, r, o) & \text{if } p = x \\ (s, r, e) & \text{if } o = x \\ (s, r, o) & \text{otherwise} \end{cases} \quad (9)$$

Note that while an unknown entity could theoretically exist valid mappings to multiple known entities, in practice \mathcal{A}_{gcp} typically finds a unique entity in G_e due to the constraining relationships and context in G_c , making the determination of e well-defined. \mathcal{A}_{gcp} is provided with a task description and an output-format instruction only, without the in-context examples too. Taking t_2^c in Figure 2 as an example, the simplified input and output content are shown as follows:

```
# Input (E is omitted)
t2c: <x1, Daughter_of, Christopher>

Ge(t2) = {
    t2e: <Elizabeth, Daughter_of, Christopher>
    t3e: <Kathleen, Sister_of, Christopher>
}

# Output
e → Elizabeth ⇒ Yt2 = (x1! = None) = True

# Update other triplets
t4c: <x1, Founder_of, x2> ← <Elizabeth, Founder_of, x2>
```

Early Stop in Checking The claim requires all triples to be verified as True for support. In practice, if any triple validates as False, we terminate the process early and set the final label Y to False.

4 Experiment

4.1 Experiment Setup

Datasets We evaluate GraphFC on three widely used fact-checking datasets. **HOVER** (Jiang et al., 2020) comprises claims necessitating multi-hop reasoning across Wikipedia articles for validation. We employ its validation set, which is categorized into three subsets according to reasoning complexity: two-hop, three-hop, and four-hop. For **FEVEROUS** (Aly et al., 2021), we use its validation set and select claims that only require sentence evidence following (Pan et al., 2023). For **SciFact** (Wadden et al., 2020), we employ its validation set, where claims with complete evidence that either support or refute the claim are selected. These datasets encompass various domains and complexity levels, offering a thorough benchmark for automated fact-checking systems.

Baselines We evaluate GraphFC against ten strong baselines in two categories. The first category comprises **pretrained/fine-tuned models**, including BERT-FC (Soleimani et al., 2020), which uses BERT-large for binary classification; LisT5 (Jiang et al., 2021), leveraging T5 with listwise concatenation; RoBERTa-NLI (Nie et al., 2019a) and DeBERTaV3-NLI (He et al., 2021), both fine-tuned on multiple NLI datasets; and MULTIVERS (Wadden et al., 2021), which employs LongFormer (Beltagy et al., 2020) for processing long evidence sequences. The second category consists of **in-context learning models**, including Codex (Chen et al., 2021) and FLAN-T5 (Chung et al., 2022) with 20-shot learning, ProgramFC (Pan et al., 2023) combines Codex and FLAN-T5 for generating reasoning programs. We also implement two prompt-based approaches: Direct (zero-shot) and Decomposition (10-shot), which break claims into multiple textual sub-claims, using GPT-3.5-Turbo. Although other notable methods like PACAR (Zhao et al., 2024) exist, they are excluded due to differences in retrieval techniques and the lack of open-source code, ensuring a fair comparison. Details about the baselines are provided in Appendix B

Implementation Details We use gpt-3.5-turbo-0125 as our main language model, with Mistral-7B-Instruct-v0.3¹ for ablation study 4.3.2. Experiments run on a Tesla

¹<https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3>

V100 SXM2 32GB GPU with Intel(R) Xeon(R) Silver 4214 CPU. For Graph Construction, we implement 10-shot learning ($K=10$), more challenging than the 20 examples used by baselines (Codex, FLAN-T5, ProgramFC). Examples are randomly sampled from the training set. Graph completion and matching operate in a zero-shot mode without demonstrations. In the Gold Evidence setting (gold), we use provided gold evidence. In the Open Book setting (open), we follow ProgramFC (Pan et al., 2023), using BM25 (Robertson et al., 2009) retriever via Pyserini (Lin et al., 2021) for all methods. The top-5 retrieved paragraphs serve as evidence.

Evaluation Metric Following (Pan et al., 2023) and (Wang and Shu, 2023), we employ the macro-F1 score as the primary metric, which provides a balanced evaluation across all categories.

4.2 Main Results

GraphFC significantly outperforms state-of-the-art baselines, as shown in Table 1, highlighting three key findings that confirm its effectiveness.

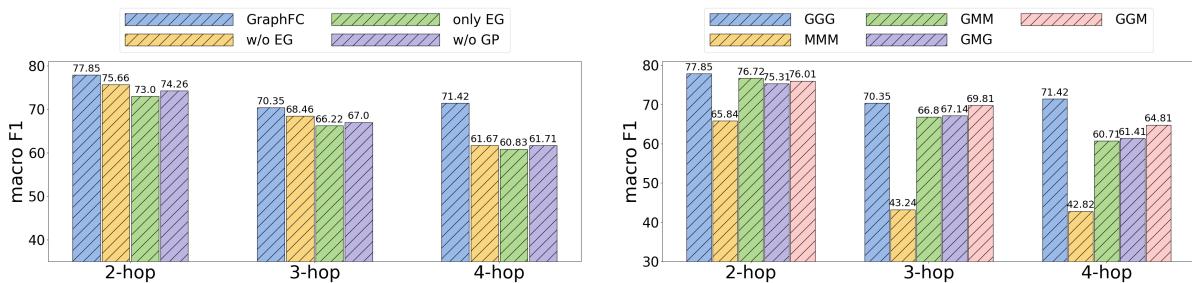
GraphFC consistently outperforms existing methods across all settings. On HOVER, we achieve accuracy improvements of 3.75-5.73% in gold settings and 4.08-8.31% in open-book settings across 2-4 hop claims compared to ProgramFC. For FEVEROUS, our method outperforming ProgramFC by 1.98% and 5.08% in gold and open settings, respectively. On SciFact, we obtain 2.47% and 7.71% gains in gold and open settings, respectively.

GraphFC shows increasing performance gains with claim complexity. Results on HOVER shows that performance improvements increase with reasoning complexity, from 4.08% for 2-hop to 8.31% for 4-hop claims in open setting. This validates the effectiveness of our graph-based approach in handling complex claims.

GraphFC maintains robust performance in open setting. The performance gap between open and gold setting is smaller with GraphFC, showing a 3.95% gap for 4-hop claims on HOVER compared to ProgramFC’s 6.53%. This improvement stems from our evidence graph construction that effectively filters and organizes relevant information to enhance reasoning.

Model	HOVER(2-hop)		HOVER(3-hop)		HOVER(4-hop)		FEVEROUS		SciFact	
	gold	open								
<i>Pretrained/Fine-tuned</i>										
BERT-FC	53.40	50.68	50.90	49.86	50.86	48.57	74.71	51.60	-	-
LisT5	56.15	52.56	53.76	51.89	51.67	50.46	77.88	54.15	-	-
RoBERTa-NLI	74.62	63.62	62.23	53.99	57.98	52.40	88.28	57.80	-	-
DeBERTaV3-NLI	<u>77.22</u>	68.72	65.98	60.76	60.49	56.00	<u>91.98</u>	58.81	-	-
MULTIVERS	68.86	60.17	59.87	52.55	55.67	51.86	86.03	56.61	77.24	54.17
<i>In-context learning</i>										
Codex	70.63	65.07	66.46	56.63	63.49	57.27	89.77	62.58	-	-
FLAN-T5	73.69	69.02	65.66	60.23	58.08	55.42	90.81	63.73	-	-
ProgramFC	74.10	<u>69.36</u>	66.13	<u>60.63</u>	<u>65.69</u>	<u>59.16</u>	91.77	<u>67.80</u>	84.90	70.63
Direct	73.53	68.85	65.09	53.77	59.03	46.93	88.44	66.41	73.30	60.54
Decompostion	72.67	65.40	<u>66.29</u>	54.65	60.88	54.41	85.79	59.38	81.64	72.92
GraphFC	77.85	73.44	70.35	66.54	71.42	67.47	93.75	72.88	87.37	80.63

Table 1: Comparison of main results (macro-F1 in %) across the HOVER, FEVEROUS, and SciFact datasets in both open-book and gold evidence settings. The best results are in **bold**, and the second-best results are underlined.



(a) Component analysis of GraphFC on HOVER in terms of macro-F1.
(b) Backbone analysis of GraphFC (GPT-3.5 vs Mistral-7B) on HOVER in terms of macro-F1.

Figure 3: Ablation studies of GraphFC.

4.3 Ablation Study

4.3.1 Component Analysis

To evaluate the effectiveness of key components in GraphFC, we conduct ablation experiments by removing or modifying specific components. As shown in Figure 3a, the variant *w/o EG* eliminates the Evidence Graph Construction component, directly performing fact-checking on raw evidence texts. *only EG* exclusively utilizes the constructed evidence graph without referencing the original text. *w/o GP* removes the Graph-Guided Planning component, leading to independent parallel verification of each triple in the claim graph.

Effectiveness of evidence Graph construction. Removing the evidence graph (*w/o EG*) causes performance drops at all hop levels on HOVER, especially a significant decrease at 4 hops claims, which contain richer knowledge and longer text, requiring extensive evidence for verification. This demonstrates the advantage of evidence graphs in constructing structured evidence that effectively

filters out irrelevant context. On the other hand, using only the evidence graph (*only EG*) leads to performance loss at all levels, indicating that relying solely on the graph can result in the loss of valuable information from the evidence text.

Effectiveness of graph-guided planning. Removing graph-guided planning (*w/o GP*) significantly decreases overall performance, especially in 4-hop claims on HOVER which require multi-step reasoning, highlighting its crucial role in structuring multi-step reasoning and ensuring that unknown entities are properly identified in complex verification tasks.

4.3.2 Backbone Analysis

To investigate the impact of different language models on model performance, we conducted ablation experiments by replacing GPT-3.5 Turbo with Mistral-7B across three components: graph construction, matching, and completion agents. The baseline configuration using GPT-3.5 for all components is denoted as *GGG*, while other configura-

Example		Decomposition	GraphFC		
Claim	Label		Claim Graph	Evidence Graph	Fact-Checking
Both Craig Serling and Zhang Yimou are film directors by profession.	supports	c_1 : Craig Serling is a film director → true c_2 : Zhang Yimou is a film director → true Prediction: supports ✓	t_1^1 : <Craig Serling, is, film director> t_2^1 : <Zhang Yimou, is, film director>	t_1^2 : <Zhang Yimou, is a, Film director> t_2^2 : <Craig Serling, is an, American film director>	$t_1^1 \rightarrow \text{true}$ $t_2^1 \rightarrow \text{true}$ Prediction: supports ✓
The professions of the host of the 2012 documentary Anti-Semitism in the 21st Century: The Resurgence were journalist and writer.	refutes	c_1 : The host of the documentary is a journalist → true c_2 : The host of the documentary is a writer → true Prediction: supports ✗	t_1^1 : <host, of, 2012 documentary> t_2^1 : < x_1 , is, journalist> t_3^1 : < x_1 , is, writer>	t_1^2 : <Anti-Semitism in the 21st Century: The Resurgence, first aired on, on January 8, 2007> t_2^2 : <Ant-Semitism in the 21st Century: The Resurgence, is hosted by, Judy Woodruff>	$t_1^1 \rightarrow \text{None}$ Prediction: refutes ✗
The man who produced Arabia Mountain (album) co-wrote this song. That song was sung by Lily Allen.	supports	c_1 : The man produced Arabia Mountain (album) → false c_2 : The man co-wrote this song → true c_3 : This song was sung by Lily Allen → true Prediction: refutes ✗	t_1^1 : < x_1 , was sung by, Lily Allen> t_2^1 : < x_2 , produced, Arabia Mountain> t_3^1 : < x_2 , co-wrote, x_3 >	t_1^2 : <Lily Allen, released, Littlest Things> t_2^2 : <Arabia Mountain, was produced by, Mark Ronson> t_3^2 : <Littlest Things, was written by, Mark Ronson>	$t_1^1 \rightarrow \text{Littlest Things}$ $t_2^1 \rightarrow \text{Mark Ronson}$ $t_3^1 \rightarrow \text{true}$ Prediction: supports ✓
The mountains known as the parent peak to Momhil Sar is found in northern Asia. The highest peak of Yengisogat is also found in northern Asia.	refutes	c_1 : The mountains known as the parent peak to Momhil Sar is found in northern Asia → true c_2 : The highest peak of Yengisogat is also found in northern Asia → true Prediction: supports ✗	t_1^1 : < x_1 , parent peak to, Momhil Sar> t_2^1 : < x_1 , is found in, northern Asia> t_3^1 : < x_2 , highest peak of, Yengisogat> t_4^1 : < x_2 , is found in, northern Asia>	t_1^2 : <Momhil Sar, situated in, the Hispar Muztagh subrange of the Karakoram range> t_2^2 : <Trivor, parent peak of, Momhil Sar> t_3^2 : <Yengisogat range, highest peak is, Huangguan Shan> t_4^2 : <Yengisogat range, is a subrange of, Karakoram mountain range>	$t_1^1 \rightarrow \text{Trivor}$ $t_2^1 \rightarrow \text{false}$ Prediction: refutes ✗

Table 2: Case study of representative examples from HOVER comparing Decomposition and GraphFC.

tions include *MMM* (all Mistral-7B), *GMM*, *GMG*, and *GGM* (combinations of both models).

Graph construction component critically depends on advanced LLM capabilities. The full Mistral-7B configuration (*MMM*) shows significant performance degradation, with 2-hop accuracy dropping from 77.85% to 65.84%. Graph construction requires strong reasoning abilities to decompose complex claims into fine-grained triples.

Graph match component shows higher model sensitivity than Completion. The graph matching configuration (*GMG*'s) leads to a notable drop in 2-hop accuracy to 75.31%, while the completion component (*GGM*'s) remains relatively robust with accuracy only decreasing to 76.01%. This aligns with the distribution of reasoning tasks on HOVER, as shown in Figure 4, where matching operations dominate verification tasks across all hops. These findings suggest that while graph construction requires powerful models, other components can utilize lighter models when prioritizing computational efficiency over maximum accuracy.

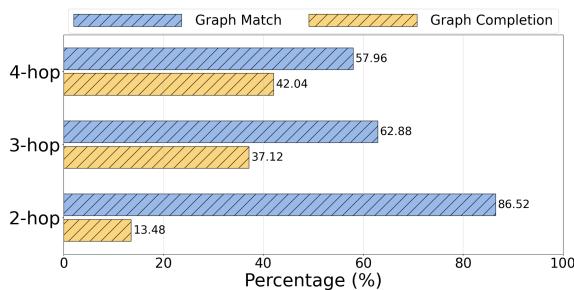


Figure 4: Distribution of task proportions for graph match and graph completion on HOVER

4.4 Case Study

To demonstrate GraphFC's ability to address the limitations of existing decomposition-based methods, we compare it with the baseline Decomposi-

tion method using four examples of varying complexity in Table 2. For clarity, only essential triples from the evidence graph that directly support or refute the claim are shown. The first example highlights the ability of both methods to handle simple claims with clear entity relationships. The third example exposes a major limitation of the Decomposition method: when verifying the sub-claim about "Arabia Mountain's producer," it fails to maintain contextual relationships between decomposed sub-claims, resulting in ambiguity around "the man." In contrast, GraphFC preserves these connections through its graph structure, explicitly modeling entities and their relationships as triples. Additionally, graph-guided planning optimizes the verification order, positioning triples with unknown entities at both ends for later processing. It demonstrates that GraphFC improves accuracy by preserving contextual relationships and ensuring an optimal verification sequence.

5 Conclusion

In this paper, we proposed GraphFC, a graph-based fact-checking framework that effectively addresses two critical challenges in LLM-driven decomposition-based method: insufficient decomposition and mention ambiguity. By converting claims into graph structures, our approach preserves contextual relationships while enabling fine-grained verification through minimal triplet units. Experimental results demonstrate GraphFC's effectiveness across three benchmarks, achieving state-of-the-art performance with significant improvements in complex reasoning scenarios. Ablation studies highlight the crucial role of graph construction and graph-guided planning in handling complex claims. These findings establish GraphFC as an effective approach for enhancing automated fact-checking systems.

Limitations

While GraphFC demonstrates substantial improvements over existing approaches, it exhibits limitations in two aspects.

- Our approach achieves relatively modest gains for simple claims verification. As shown in Table 1, the improvement margin on 2-hop claims in the HOVER dataset (0.63% in gold setting) is notably smaller compared to 3-hop and 4-hop claims. This limitation is inherent in our method’s design: while GraphFC excels at handling complex multi-hop reasoning, for simple claims where conventional methods already perform well, the sophisticated graph construction and matching procedures may introduce unnecessary complexity.
- The graph construction component of our method relies on the reasoning capabilities of LLMs. As demonstrated in Figure 3b, replacing GPT-3.5 with Mistral-7B in the graph construction phase leads to a significant decline in performance. This suggests that the accuracy of decomposing claims into fine-grained triple representations benefits from the advanced reasoning capabilities of more powerful language models. Future work will focus on developing lighter-weight graph construction techniques and exploring more efficient verification strategies for simple claims while maintaining the current effectiveness on complex reasoning tasks.

References

- Rami Aly, Zhijiang Guo, Michael Schlichtkrull, James Thorne, Andreas Vlachos, Christos Christodoulopoulos, Oana Cocarascu, and Arpit Mittal. 2021. Feverous: Fact extraction and verification over unstructured and structured information. *arXiv preprint arXiv:2106.05707*.
- Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.
- Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 632–642, Lisbon, Portugal.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.
- In-Zu Gi, Ting-Yu Fang, and Richard Tzong-Han Tsai. 2021. Verdict inference with claim and retrieved elements using roberta. In *Proceedings of the Fourth Workshop on Fact Extraction and VERification (FEVER)*, pages 60–65.
- Jian Guan, Jesse Dodge, David Wadden, Minlie Huang, and Hao Peng. 2024. Language models hallucinate, but may excel at fact verification. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1090–1111, Mexico City, Mexico. 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. *arXiv preprint arXiv:2011.03088*.
- Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: A python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2356–2362.
- Alisa Liu, Swabha Swayamdipta, Noah A. Smith, and Yejin Choi. 2022. WANLI: Worker and AI collaboration for natural language inference dataset creation. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 6826–6847, Abu Dhabi, United Arab Emirates.
- Yixin Nie, Haonan Chen, and Mohit Bansal. 2019a. Combining fact extraction and verification with neural semantic matching networks. In *Proceedings of*

- the AAAI conference on artificial intelligence*, pages 6859–6866.
- Yixin Nie, Haonan Chen, and Mohit Bansal. 2019b. Combining fact extraction and verification with neural semantic matching networks. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 6859–6866, Honolulu, Hawaii, USA.
- 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 (ACL)*, pages 4885–4901, Online.
- 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. *arXiv preprint arXiv:2305.12744*.
- Alicia Parrish, William Huang, Omar Agha, Soo-Hwan Lee, Nikita Nangia, Alexia Warstadt, Karmanya Aggarwal, Emily Allaway, Tal Linzen, and Samuel R. Bowman. 2021. Does putting a linguist in the loop improve NLU data collection? In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4886–4901, Punta Cana, Dominican Republic.
- Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.
- Amir Soleimani, Christof Monz, and Marcel Worring. 2020. Bert for evidence retrieval and claim verification. In *Advances in Information Retrieval: 42nd European Conference on IR Research, ECIR 2020, Lisbon, Portugal, April 14–17, 2020, Proceedings, Part II 42*, pages 359–366. Springer.
- James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. Fever: a large-scale dataset for fact extraction and verification. *arXiv preprint arXiv:1803.05355*.
- David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. 2020. Fact or fiction: Verifying scientific claims. *arXiv preprint arXiv:2004.14974*.
- David Wadden, Kyle Lo, Lucy Lu Wang, Arman Cohan, Iz Beltagy, and Hannaneh Hajishirzi. 2021. Multivers: Improving scientific claim verification with weak supervision and full-document context. *arXiv preprint arXiv:2112.01640*.
- Haoran Wang and Kai Shu. 2023. Explainable claim verification via knowledge-grounded reasoning with large language models. *arXiv preprint arXiv:2310.05253*.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 1112–1122, New Orleans, Louisiana, USA.
- Xuan Zhang and Wei Gao. 2023. Towards llm-based fact verification on news claims with a hierarchical step-by-step prompting method. *arXiv preprint arXiv:2310.00305*.
- Chen Zhao, Chenyan Xiong, Corby Rosset, Xia Song, Paul Bennett, and Saurabh Tiwary. 2020. Transformer-xh: Multi-evidence reasoning with extra hop attention. In *International Conference on Learning Representations*.
- 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.
- 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. *arXiv preprint arXiv:1908.01843*.

Appendix

A Graph-Guided Fact-Checking

Here we give an algorithm-style graph-guided fact-checking process in algorithm 1. Based on the sorted triple list \mathcal{T} in Equation 4, we conduct a fact-checking process for each triplet in the claim graph G_c .

1. If the triple contains only grounded entities, we apply a graph match agent to verify its validity against $G_e^{(t)}$. Under this situation, if the validation process returns a *false*, then the fact-checking function returns a *false*.

2. Otherwise, if the triple contains an unknown entity, we apply a graph completion agent to find the correct entity and ground it. If grounding is successful, we update both the claim graph G_c and evidence graph G_e . If the function fails to find grounding results, the fact-checking function returns *false*.
3. If no *false* returns during the fact-checking process, then the fact-checking function returns a *true*.

Algorithm 1 Graph-Guided Fact-Checking

Require: claim graph G_c , evidence graph G_e , evidence set E , agents $\mathcal{A}_{gc}, \mathcal{A}_{gm}, \mathcal{A}_{gcp}$, and sorted triples \mathcal{T}

Ensure: fact-checking result Y (True or False)

```

 $Y \leftarrow \text{True}$ 
for each  $t$  in  $\mathcal{T}$  do
     $(s, p, o) \leftarrow t$ 
    if  $s \notin \mathcal{X}_c$  and  $o \notin \mathcal{X}_c$  then
         $Y_t \leftarrow \mathcal{A}_{gm}(t, G_e^{(t)}, E)$ 
        if  $Y_t == \text{False}$  then
             $Y \leftarrow \text{False}$ 
            break
        end if
    else
         $e, Y_t \leftarrow \mathcal{A}_{gcp}(t, G_e^{(t)}, E)$ 
        if  $Y_t == \text{False}$  then
             $Y \leftarrow \text{False}$ 
            break
        else
             $\mathcal{E}_c \leftarrow \mathcal{E}_c \cup e\{\text{Update Claim Graph}\}$ 
             $G_e \leftarrow G_e \cup \mathcal{A}_{gc}(E, [e])\{\text{Update Evidence Graph}\}$ 
        end if
    end if
end for
return  $Y$ 

```

B Baseline Details

We present detailed implementation for our baseline models across two categories: **Pre-trained/Fine-tuned Models:**

- **BERT-FC** (Soleimani et al., 2020): Employs bert-large-uncased (345M parameters) for binary classification, concatenating claims and evidence as '[CLS] claim [SEP] evidence'. Fine-tuned using 20 random examples from the dataset.

- **LisT5** (Jiang et al., 2021): Implements t5-large with listwise concatenation, processing all evidence sentences simultaneously for binary classification (Supports/Refutes).

- **RoBERTa-NLI** (Nie et al., 2019a): RoBERTa-large model fine-tuned on SNLI (Bowman et al., 2015), MNLI (Williams et al., 2018), FEVER-NLI (Nie et al., 2019b), ANLI (R1, R2, R3) (Nie et al., 2020) datasets, with additional fine-tuning using 20 task-specific examples from HOVER/FEVEROUS.

- **DeBERTaV3-NLI** (He et al., 2021): DeBERTaV3-large model trained on 885,242 NLI pairs from FEVER and other NLI datasets (MNLI, ANLI, LingNLI (Parrish et al., 2021), WANLI (Liu et al., 2022)).

- **MULTIVERS** (Wadden et al., 2021): Implements LongFormer (Beltagy et al., 2020) architecture to handle extended evidence sequences, specifically fine-tuned on FEVER (Thorne et al., 2018) dataset.

In-context Learning Models:

- **Codex** (Chen et al., 2021): Utilizes code-davinci-002 with 20 in-context examples.
- **FLAN-T5** (Chung et al., 2022): Leverages FLAN-T5-XXL (3B parameters) with 20 in-context examples, using structured prompts for claim verification.
- **ProgramFC** (Pan et al., 2023): Combines Codex and FLAN-T5 for generating reasoning programs. For fair comparison, we adopt their single reasoning chain configuration ($N = 1$).
- **Direct, Decomposition:** Implements gpt-3.5-turbo-0125 using two approaches: (1) zero-shot prompting (Direct) and (2) claim decomposition with 10 in-context examples (Decomposition). The latter approach first breaks down complex claims into independently verifiable textual sub-claims, then applies direct prompting to verify each sub-claim. The complete prompts are provided in Appendix C.

C Prompts

Below, we present the prompt templates used for the baseline methods (Direct, Decomposition) and GraphFC. For clarity, we have omitted the in-context-learning examples.

Listing 1: Direct Prompt

```
[[Evidence]]
Based on the above information, is it true that [[Claim]]?
True or false? The answer is:
```

Listing 2: Decomposition Prompt

```
## Task Description:
Break down the given into multiple sub claims that:
- Cannot be further divided into simpler meaningful
  statements
- Has a clear truth value (true or false)
- Contains a single subject and predicate
- Does not contain logical connectives (and, or, if-then,
  etc.)

## Input Format:
A complex claim or statement.

## Output Format:
Each atomic proposition should be presented on a new line,
separated by blank lines for clarity.

## Examples:
(... more in-context examples here ...)

## Real Data:
Input: [[Claim]]
Output:
```

Listing 3: Claim Graph Construction Prompt

```
# Knowledge Graph Construction Specification

You are an expert in knowledge graph construction. Your task
is to parse natural language claims into a formal
claim graph representation by following these
specifications:

## 1. Entity Types
- Person: Real individuals
- Location: Places, cities, regions
- Organization: Companies, institutions, groups
- Time: Dates, years, periods
- Number: Numerical values
- Concept: Abstract ideas, categories
- Object: Physical items
- Work: Creative works (books, songs, etc.)
- Event: Occurrences, happenings
- Species: Biological organisms

## 2. Constraint Types
- temporal: Time-related constraints (year, date, period)
- spatial: Location-related constraints (in, at, from)
- condition: Qualifying conditions or attributes
- context: Broader situational context

## 3. Atomic Proposition Rules

### Definition
An atomic proposition must:
- Express a single, indivisible fact
- Cannot be broken down into simpler meaningful statements
- Must preserve all relevant context
- Must maintain temporal and spatial relationships

### Decomposition Guidelines
1. Structural Analysis:
  - Split complex sentences at conjunction words (and, but, or)
  - Separate conditional statements (if/then) into distinct
    propositions
  - Identify dependent clauses and their relationships
  - Preserve modifiers and qualifiers with their related
    concepts

2. Semantic Preservation:
  - Maintain causal relationships
  - Preserve temporal order
  - Keep spatial relationships intact
  - Retain contextual qualifiers
```

```
## 4. Fuzzy Entity Identification Process

### Entity Reference Types
1. Direct Reference:
  - Uses proper name ("John", "Paris")
  - Specific numerical values
  - Well-defined concepts

2. Indirect Reference:
  - Uses descriptions ("the teacher", "that city")
  - Role-based references ("the founder", "the mother")
  - Attribute-based references ("the tall building")

3. Contextual Reference:
  - Requires information from other statements
  - Part of a collective reference
  - Implied entities

## 5. Fuzzy Entity Decision Tree

1. Initial Check:
  - Is the entity referred to by proper name? Not fuzzy
  - Is the entity a specific number or date? Not fuzzy
  - Is the entity a well-defined concept? Not fuzzy

2. Context Analysis:
  - Does the entity require contextual information? Fuzzy
  - Is the entity part of a group or collection? Fuzzy
  - Is the entity only described by role or attribute? Fuzzy
  - Is the entity referenced through relationships? Fuzzy

3. Coreference Resolution:
  - Track entities across multiple atomic propositions
  - Maintain consistent fuzzy entity IDs ($A$, $B$, etc.)
  - Document relationships between fuzzy entities

## 6. Output Format

### 1. Entity Definitions
```python
entities = [
 {
 "name": str, # Entity identifier
 "type": str, # One of the predefined entity
 types
 "is_fuzzy": bool # True if entity meets fuzzy
 criteria
 }
]
```

### 2. Relationship Quadruples with Atomic Propositions
```python
quads = [
 {
 "atomic_proposition": str, # The original atomic
 statement this quad represents
 "subject": str, # Entity name or
 identifier
 "predicate": str, # Relationship type
 "object": str, # Entity name, identifier
 , or value
 "constraint": { # Optional constraint
 "type": str, # One of the predefined
 constraint types
 "value": str # Constraint value
 }
 }
]
```

### 6. Validation Steps

### 1. Quad and Atomic Proposition Validation
For each quad:
- Verify its atomic proposition cannot be further decomposed
- Check if it captures all relevant context
- Ensure temporal/spatial information is preserved
- Validate the relationship between entities
- Check if all relevant constraints are captured

### 2. Semantic Consistency Check
For the entire graph:
- Verify all quads together preserve the original claim's
  meaning
- Check for logical gaps or inconsistencies
- Validate temporal/causal order is maintained
- Ensure entity coreferences are consistent

### 7. Example
(... more in-context examples here ...)

Now, please parse the following claim into the formal
representation described above:
Claim = [[Claim]]

Output =
```

Listing 4: Evidence Graph Construction Prompt

```
## Task:  
Extract semantic triples from the given evidence and ensure  
every extracted triple is context-independent.  
  
## Input:  
- Evidence: [Full text of the evidence]  
- Entities: [List of entities mentioned, including entity  
name and its type]  
  
## Output Format:  
Extract triples separated by newlines, with each triple in  
the format:  
`Subject | Predicate | Object`  
  
## Extraction Guidelines:  
1. Ensure that either **Predicate** and **Object**, or both  
in each triple are from the provided Entity Set.  
2. "Identify all relationships" from the Evidence Text that  
meet this requirement.  
3. Every triple should be context-independent. Use full  
forms or expanded phrases for relational references  
where necessary.  
  
## Examples  
(... more in-context examples here ...)  
  
## Real data  
Evidence: [[Evidence]]  
  
Entity Set:  
[[Entity Set]]  
  
Output:
```

Listing 5: Graph Match Prompt

```
Evidence:  
[[Evidence]]  
  
Using the provided evidence, determine whether the given  
quadruple is true or false.  
Quadruple:  
[[Quadruple]]  
  
Output(true/false):
```

Listing 6: Graph Completion Prompt

```
Evidence:  
[[Evidence]]  
  
Complete the fuzzy entity in the quadruple based on above  
evidence. If a suitable entity is found, output its  
name; otherwise, output "none".  
  
Quadruples:  
[[Quadruples]]  
  
Output:
```