Program Enhanced Fact Verification with Verbalization and Graph Attention Network

Xiaoyu Yang[†][*], **Feng Nie**[§][*], **Yufei Feng**[†], **Quan Liu**[‡], **Zhigang Chen**[‡], **Xiaodan Zhu**[†]

[†] ECE & Ingenuity Labs Research Institute, Queen's University

[§] Sun Yat-sen University

[‡] State Key Laboratory of Cognitive Intelligence, iFLYTEK Research

{xiaoyu.yang, xiaodan.zhu, yufei.feng}@queensu.ca

fengniesysu@gmail.com {quanliu, zgchen}@iflytek.com

Abstract

Performing fact verification based on structured data is important for many real-life applications and is a challenging research problem, particularly when it involves both symbolic operations and informal inference based on language understanding. In this paper, we present a **Prog**ram-enhanced **V**erbalization and **G**raph **AT**tention Network (**ProgVGAT**) to integrate *programs* and *execution* into textual inference models. Specifically, a *verbalization with program execution* model is proposed to accumulate evidences that are embedded in operations over the tables. Built on that, we construct the *graph attention verification networks*, which are designed to fuse different sources of evidences from verbalized program execution, program structures, and the original statements and tables, to make the final verification decision. To support the above framework, we propose a *program selection* module optimized with a new training strategy based on margin loss, to produce more accurate programs, which is shown to be effective in enhancing the final verification results. Experimental results show that the proposed framework achieves the new state-of-the-art performance, a 74.4% accuracy, on the benchmark dataset TABFACT. Our code is available at https://github.com/arielsho/Program-Enhanced-Table-Fact-Checking.

1 Introduction

With the overwhelming information available on the Internet, fact verification has become crucial for many applications such as detecting fake news, rumors, and political deception (Rashkin et al., 2017; Thorne et al., 2018; Goodrich et al., 2019; Vaibhav et al., 2019; Kryściński et al., 2019), among others. Existing research has mainly focused on collecting

---

Table with title 'Ji-young Oh'

Year	Tournaments Played	Avg. Score	Scoring Rank
2007	22	72.46	81
2008	29	71.65	22
2009	25	71.90	34
2010	18	73.42	92
2011	11	74.42	125

Statement	Ji-young Oh played more tournament in 2008 than any other year.
Label	ENTAILED
Program	*eq { max { all_rows ; tournaments played } ; hop { filter_eq { all_rows ; year ; 2008 } ; tournaments played } }* = True

Figure 1: An example of fact verification over tables.

and processing evidences from unstructured text data (Liu et al., 2020; Nie et al., 2019; Hanselowski et al., 2018; Yoneda et al., 2018), which is only one type of data where important facts exist. Structured and semi-structured data, e.g., tables in relational databases or in the HTML format is also ubiquitous. Performing fact validation based on structured data is important yet challenging and further study is highly desirable. Fig. 1 depicts a simplified example in which systems are expected to decide whether the facts in the table support the natural language statement.

In addition to its importance in applications, the task presents research challenges of fundamental interests—the problem inherently involves both informal inference based on language understanding (Dagan et al., 2005; MacCartney and Manning, 2009, 2008; Bowman et al., 2015, 2016) and symbolic operations such as mathematical operations (e.g., *count* and *max*). Recently, pre-trained language models such as BERT (Devlin et al., 2019) have shown superior performances in natural language inference by leveraging knowledge from large text datasets and can capture complicated semantic and syntactic information among premises and hypotheses (Radford, 2018; Radford et al., 2019; Liu et al., 2019; Dong et al., 2019b).

However, such methods tend to fail when verification requires the joint modelling of both symbolic operations and language inference (Gupta et al., 2020) such as the case depicted in Fig. 1.

To effectively enable symbolic operations and integrate them into language-based inference models, we propose a framework centered around programs, i.e., logical forms that can be executed to find evidences from structured data. Our model starts with a *program selection* module, for which we propose a new training strategy based on margin loss to find programs that can accurately extract supporting facts from tables. To bridge the semantic gap between structured programs and tables as well as to leverage the structures of programs, we propose a novel model based on *verbalization with program execution*. The verbalization algorithm interweaves with program execution in order to accumulate evidences inherently embedded in operations, and the algorithm recursively converts executed operations in programs into natural language sentences. Built on that, we propose *graph-based verification network* to fuse different sources of evidences from verbalized program execution, together with the original statements and tables, to support the final verification decision.

We conduct experiments on the recently proposed large scale benchmark dataset TAB-FACT (Chen et al., 2020). Experimental results show that our proposed framework achieves new state-of-the-art performance, an accuracy of 74.4%, substantially improving the previously reported best performance with the accuracy of 71.7%. Our detailed analysis shows the effectiveness of verbalization and graph-based verification network in utilizing programs to achieve the improvement. The analysis also demonstrates that the program selection optimized with the proposed training strategy based on margin loss effectively improves the final verification results.

2 Related Work

Fact Verification. Existing work on fact verification is mainly based on collecting and using evidences from unstructured text data (Liu et al., 2020; Nie et al., 2019; Hanselowski et al., 2018; Yoneda et al., 2018). FEVER (Thorne et al., 2018) is one of the most influential benchmark datasets built to evaluate systems in checking claims by retrieving Wikipedia articles and extracting evidence sentences. Recent proposed FEVER 2.0 (Thorne

et al., 2019) has a more challenging dataset to verify factoid claims and an adversarial attack task. Some previous models are developed on the official baseline (Thorne et al., 2018) with three step pipeline (Chen et al., 2017a) for fact verification (Hanselowski et al., 2018; Yoneda et al., 2018; Yin and Roth, 2018; Nie et al., 2019). Others formulates fact verification as graph reasoning (Zhou et al., 2019; Liu et al., 2020). Natural language inference (NLI) task is also a verification problem which is fully based on unstructured text data (Dagan et al., 2005, 2010; Bowman et al., 2015; Parikh et al., 2016; Chen et al., 2017c; Ghaeini et al., 2018; Peters et al., 2018). Neural models proposed for NLI have been shown to be effective (Parikh et al., 2016; Chen et al., 2017d,e; Ghaeini et al., 2018; Peters et al., 2018), including models incorporating external knowledge (Chen et al., 2017b; Yang et al., 2019). Our work focuses on fact verification based on structured tables (Chen et al., 2020).

For verification performed on structured data, Chen et al. (2020) propose a typical baseline (Table-BERT), which is a semantic matching model taking a linearized table T and statement S as input and employs BERT for verification. The other model (LPA) proposed in (Chen et al., 2020) uses Transformer blocks to compute semantic similarity between a statement and program. A contemporaneous work (Zhong et al., 2020) proposes Logical-FactChecker aiming to leverage programs for fact verification. LogicalFactChecker utilizes inherent structures of programs to prune irrelevant information in evidence tables and modularize symbolic operations with module networks. Different from theirs, our proposed framework verbalizes the accumulated evidences from program execution to support the final verification decision with graph attention networks.

Semantic Parsing. A line of work uses program synthesis or logic forms to address different natural language processing problems, such as question answering (Berant et al., 2013; Berant and Liang, 2014), code generation (Yin and Neubig, 2017), SQL synthesis (Zhong et al., 2017; Yu et al., 2018) and mathematical problem solving (Kushman et al., 2014; Shi et al., 2015). Traditional semantic parsing methods greatly rely on rules and lexicons to parse texts into structured representations (Zettlemoyer and Collins, 2005; Berant et al., 2013; Artzi and Zettlemoyer, 2013). Recent semantic parsing methods strives to leverage the power of neural
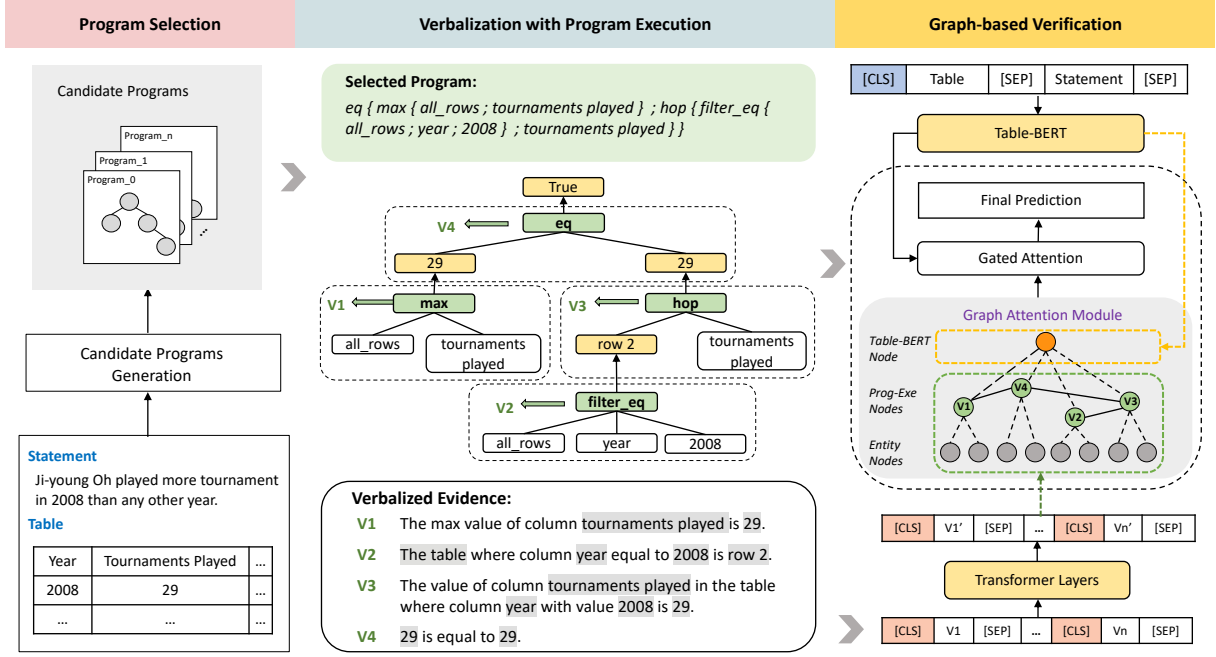
Figure 2: An overview of the proposed framework.

networks (Neelakantan et al., 2016; Jia and Liang, 2016; Liang et al., 2017; Yu et al., 2018; Dong et al., 2019a). Our work leverages symbolic operations inherited in programs produced by neural semantic parsers to enhance fact verification over structured data.

3 Model

We present the proposed framework (ProgVGAT), which centers around *programs* and *execution* to integrate symbolic operations for fact verification. Fig. 2 depicts the overall architecture. This section is organized as follows. We first introduce the task formulation along with program representations in Sec. 3.1. Then, we describe in Sec. 3.2 the program selection module that aims to obtain a semantically relevant program for verification. Built on that, we present our proposed verbalization algorithm and graph attention network to dynamically accumulate evidences embedded in symbolic operations of programs for final verification in Sec. 3.3 and Sec. 3.4.

3.1 Task Formulation and Notations

Formally, given a structured evidence table T and a statement S, the fact verification task aims to predict whether T *entails* S or *refutes* it. The evidence table $T = \{T_{i,j} | i \leq R, j \leq C\}$ has R rows and C columns, and $T_{i,j}$ is the value in the (i, j)-th cell. $T_{i,j}$ can be of different data types, e.g., a word,

number, phrase, or even natural language sentence.

Program representation. Given a statement S, a semantic consistent program $z = \{op_i\}_{i=1}^M$ is a tree consisting of multiple executable symbolic operations op_i. An example of programs is shown in the center of Fig. 2. An operation $op_i = (op_i.t, op_i.arg)$ contains an operator $op_i.t$ (e.g., *max* in the figure) and arguments $op_i.arg$ relevant to table T (e.g., *all rows* and *tournaments played*), and the execution of an operation yields an output/answer ans (e.g., *29*). Before building the model, we follow the previous work (Chen et al., 2020) and perform rule-based entity linking and latent program search to obtain a set of candidate programs $\mathcal{Z} = \{z_i\}_{i=1}^N$. Specifically, entity linking (Nie et al., 2018) detects relevant entities (i.e., cells in evidence table T) in statement S using a set of string matching rules. And the latent program search algorithm finds all valid combinations of pre-defined operations and detected entities by traversing and executing them recursively through the evidence table T.

3.2 Program Selection

Given a statement S, program selection aims to obtain a high quality program z^* from a set of candidate programs $\mathcal{Z} = \{z_i\}_{i=1}^N$. Previous work (Chen et al., 2020) optimizes the model via a cross

entropy loss:

$$J(\theta) = -\sum_{z \in \mathcal{Z}} \mathbb{1}_{[z]_y} \log p_\theta(z|S,T) \qquad (1)$$

where $\mathbb{1}_{[z]_y}$ is an indicator function which takes the value 1 if the execution result of a program z (i.e., the output of the root; *"True"* in Fig. 2) is consistent with the ground-truth verification label y, otherwise 0. The former type of programs are called *label-consistent* programs and the latter *label-inconsistent* programs. Despite being simple, it ignores that only one of the label-consistent programs is correct and can potentially assign too much credit to spurious programs that execute to get the correct verification labels with incorrect operations during training. Meanwhile, the loss function considers every program in $\mathcal{Z}$ during training and there is only one most relevant program z^* selected in testing phase, creating discrepancies between training and testing.

To remedy these issues, we introduce a margin loss which encourages to select a most positive program (i.e., positive program with maximum semantic similarity score) while maintaining a margin with label-inconsistent programs:

$$J(\theta) = \max\left(p_\theta(z'_{neg}|S,T) - p_\theta(z'_{pos}|S,T) + \gamma, 0\right) \qquad (2)$$

where z'_{neg} and z'_{pos} refer to the label-inconsistent program and the label-consistent program with the highest probability in their own categories, respectively. γ is a hyperparameter controlling the margin between the positive instances and negative ones. To measure the semantic similarity between a candidate program z and the corresponding statement S, we leverage pre-trained language model BERT (Devlin et al., 2019) instead of simply training transformer layers as proposed in (Chen et al., 2020). Specifically, given a (S, z) pair, it is prefixed with the [CLS] token and suffixed with the [SEP] token to form the input of BERT. Then a 1-layer MLP with a sigmoid layer is applied on top of the BERT model to produce the final probability $p_\theta(z|S,T) = \sigma(W_r\boldsymbol{h})$.

Instead of selecting the top program based on Eq. 2, we also tried the exploration strategy proposed in (Guu et al., 2017) to sample a non-top label-consistent program with a small probability. However, this does not further improve the verification performance on the development dataset. We

---

Algorithm 1 Verbalization

Require Statement and evidence table pair (S,T), and parsed program $z^* = \{op_i\}_{i=1}^M$; Pre-defined operator $P = \{p_i\}_{i=1}^R$; A template function $\mathcal{F}(.)$ maps operation and operation results into sentences.

1: **function** VERBALIZATION(op, ret)
2:     $args = \{\}, verb\_args = \{\}$
3:     **for** a_j in arguments of operation op **do**
4:         **if** a_j is an operator in P **then**
5:             $arg_ans, verb_arg = $ VERBALIZATION(a_j, ret)
6:             $args \leftarrow args \cup arg\_ans$
7:             $verb\_args \leftarrow verb\_args \cup verb\_arg$
8:         **else**
9:             $args \leftarrow args \cup a_j$
10:             $verb\_args \leftarrow verb\_args \cup str(a_j)$
11:         **end if**
12:     **end for**
13:     Apply operation $(op.t, args)$ over evidence table T, obtain operation result ans
14:     Apply $\mathcal{F}(op.t, verb\_args, ans)$, obtain verbalized operation result $verb\_ans$ and verbalized operation $verb\_op$
15:     Update $ret \leftarrow ret \cup verb\_ans$
16:     **Return** $ans, verb\_op$
17: **end function**
Set verbalized program execution $ret = \{\}$
VERBALIZATION(op_1, ret)
Return ret

---

therefore use the first ranked program in the remainder of this paper. Our proposed method relies on the program produced by this section. We further conclude the importance of the program quality in the experimental sections (i.e., Sec.5).

3.3 Verbalization with Program Execution

With the derived program z^* for each (S,T) pair, we propose to verbalize program execution—with operations in a program being recursively executed over the evidence table with a post-order traversal along the program tree. The verbalization algorithm works to convert the execution, including operators, arguments, and execution output, into natural language sentences, to accumulate evidences that are inherently embedded in operations.

Formally, an operation $op_i = (op_i.t, op_i.arg)$ contains an operator $op_i.t$ and arguments $op_i.arg$, and its execution yields an output/answer ans_i. Al-

Operation	Templates	
	Operation	Operation Results
count, [verb_arg1], ans:string/number	the number of [verb_arg1]	the number of [verb_arg1] is [to_string(ans)]
greater, [verb_arg1, verb_arg2], ans:true/false	[verb_arg1] greater than [verb_arg2]	[arg1's template] [true:is]/[false:is not] greater than [arg2's template]
filter_eq, [verb_arg1, verb_arg2, verb_arg3], ans:rows	[verb_arg1] where column [verb_arg2] equal to [verb_arg3]	[verb_arg1] where column [verb_arg2] equal to [verb_arg3] is row [indices of ans]

Table 1: Examples of generation templates for different operations.

gorithm 1 describes the verbalization procedure. The post-order traversal over program z^* and the execution of operations can be found in line 3 to line 13. The template-based generation that converts an executed operation (its operation, the arguments, and output) into natural language sentences can be found in line 14. As such, the execution of each operation $\{op_i\}_{i=1}^M$ in the program z^* is converted into an evidence sentence $V = \{v_i\}_{i=1}^M$. Table 1 lists a few operation templates.* Note that the proposed verbalization can be easily generalized to other domains by extending templates. We leave it as future work for exploring different generation methods, although for structured programs with fixed operations, template-based methods are often very effective already. Fig. 2 gives an example produced by our verbalization algorithm.

3.4 Graph-based Verification Network

We propose *graph attention verification networks*, which is designed to fuse different sources of evidences from verbalized program execution, program structures, together with the original S and table T, to make the final verification decision, shown on the right subfigure of Fig. 2.

3.4.1 Graph Definition

Nodes The graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ contains three types of nodes $\mathcal{V}$ and three types of edges $\mathcal{E}$. The first type of nodes, $(n_0, \ldots, n_{M-1})$, encode verbalized program executions obtained above, called *Prog-Exec* nodes, shown as green nodes on the right of Fig. 2. M is the number of operations in a program. The second type encodes program entities, called *entity nodes*, shown as grey nodes. As each operation execution o_i consists of arguments and execution output, we construct nodes $(n_M, \ldots, n_{K-2})$ for these entities. The third type of nodes utilize information in original tables and statements. We design a *Table-BERT* node, n_{K-1}, initialized with the output of Table-BERT proposed

in (Chen et al., 2020), denoted as the orange node in Fig. 2. In total, we have K nodes, where K varies for different (S, T, z^*) triples.

Edges For a graph $\mathcal{G}$ with K nodes, the adjacency matrix $A \in K \times K$ reflects their connectivity, where $A_{i,j}$ is set to 1 if node n_i and n_j are connected with an edge. Similar to graph nodes, we have three types of edges. We design different attention heads to handle different types of edges/connections as detailed in Section 3.4.3.

The first type of edges connect the nodes of verbalized program execution V based on the program tree structure—we connect node n_i and node n_j, if the corresponding operation o_i is a father or child operation of operation o_j in a program z.† The second type of edges connect *Prog-Exec* nodes $\{n_i\}_{i=0}^{M-1}$ with the corresponding entity nodes $(n_M, \ldots, n_{K-2})$. The third type connects *Prog-Exec* nodes, i.e., verbalized symbolic operations and executions, with *Table-BERT node*, which is NLI-based verification performed directly on statement S and T.

3.4.2 Graph Construction and Initialization

For *Table-BERT* node, we utilize the Table-BERT model proposed in (Chen et al., 2020) to obtain the representation:

$$\boldsymbol{h}_{K-1} = f_{BERT}([\widetilde{\boldsymbol{T}}; S]), \quad (3)$$

where $\widetilde{\boldsymbol{T}}$ linearizes table T into sentences; $\boldsymbol{h}_{K-1} \in \mathbb{R}^{F \times 1}$ and F are the number of features in node. We recommend readers to refer to (Chen et al., 2020) for details.

For *Prog-Exec* nodes, instead of directly using the verbalized program executions, the nodes are constructed and initialized as follows to consider

context. Given a program z^*, verbalization proposed in Sec. 3.3 generates M sentences $\{v_i\}_{i=1}^M$. We use document-level BERT proposed in (Liu and Lapata, 2019) to encode these sentences by first inserting a [CLS] token at the beginning of each v_i and a [SEP] token at the end of it. The segment embedding is set to E_A when i is odd and E_B when i is even. After applying BERT, we take the corresponding [CLS] vector at the top layer (e.g., the [CLS] inserted before v_2) to be the representation for the corresponding *Prog-Exec* node (e.g., n_2).

For *entity* nodes, we take the contextualized embeddings at positions corresponding to the entities in the top layer of BERT model as the node representation. For entities with multiple words, an average pooling is applied to produce the final entity representation.

3.4.3 Reasoning with Graph Attentions

Graph Propagation Unlike the standard graph propagation (Velickovic et al., 2018), we model different types of edges in the propagation process. Specifically, we use the following formula to update each node representation $\boldsymbol{h}_i$ in graph $\mathcal{G}$:

$$\boldsymbol{h}_i^{new} = f\big(\,\big\|_{d=1}^{D}\, \sigma(\sum_{j \in \mathcal{N}_i^d} \alpha_{ij}^d \boldsymbol{W}\boldsymbol{h}_j)\big) \quad (4)$$

where

$$e_{ij} = a(\boldsymbol{U}\boldsymbol{h}_i, \boldsymbol{U}\boldsymbol{h}_j), \quad (5)$$

$$\alpha_{ij}^d = \frac{exp(e_{ij})}{\sum_{k=1}^K A_{i,k}^d exp(e_{ik})} \quad (6)$$

where $\boldsymbol{U} \in \mathbb{R}^{F \times L}, \boldsymbol{W} \in \mathbb{R}^{F \times F}$ are trainable parameters and $a(.)$ denotes shared attention mechanism $\mathbb{R}^L \times \mathbb{R}^L \to \mathbb{R}$. Note that $\|$ refers to the concatenation operation and D denotes number of different types nodes (D is set to 3 in this paper).

To propagate along different types of edges, we extend masked attention with a multi-head mechanism to encode different types of edges in different heads. Particularly, *masked attention* in self-attention mechanism is performed for each type of edges d. The masked attention computes normalized attention coefficients $\alpha_{i,j}^d$ between node n_i and its neighbor n_j under edge type d (i.e., $A_{i,j}^d = 1$ means node i and node j is connected with the edge type d. A^d is the adjacency matrix we constructed above). To aggregate node representation from each head, we concatenate D updated nodes with a feed-forward layer in Eq. 4, yielding the final node representation $\boldsymbol{h}_i^{new*}$.

Gated Attention To aggregate information in a graph, we employ a gated attention mechanism to obtain final aggregated representation $\boldsymbol{h}_{final}$ and predict final verification label y as follows:

$$\boldsymbol{h}_{final} = \sum_{i=0}^{M-1} p_i \boldsymbol{h}_i^{new}; p_i = \sigma(\boldsymbol{h}_{K-1}^T \boldsymbol{h}_i^{new}), \quad (7)$$

$$y = \sigma(\boldsymbol{W_f}([\boldsymbol{h}_{final} \| \boldsymbol{h}_{K-1}])) \quad (8)$$

where $\boldsymbol{W_f}$ are trainable parameters, σ is the sigmoid function, and $\|$ the concatenation operation.

4 Experiment Setup

Data and Evaluation Metric As discussed above, although (semi-)structured and unstructured text data are ubiquitous in our daily life, performing fact verification across these different formats is relatively new. We conduct our experiments on recently released large-scale dataset TAB-FACT (Chen et al., 2020).

TABFACT contains 92,283, 12792, and 12779 table-statement pairs for training, validation and testing respectively. Verification on some statements requires higher-order semantics such as *argmax*, the test set is further split into a *simple* and *complex* subset according to verification difficulty. A *small* subset is provided in which the human upper-bound performance is given. Following the existing work, we use *accuracy* as the evaluation metric. Detailed data statistics are listed in Appendix A.1.

Training Details For parameters in all BERT models, the hidden size is set to 768, we use Adam optimizer (Kingma and Ba, 2014) with learning rate 2e-5, warmup step 3000, dropout 0.1. For parameters in graph attention network, the hidden feature dimensions F and L are all set to 768. All codes are implemented with PyTorch (Paszke et al., 2019). All hyper-parameters are decided according to the validation performance.

Compared Systems We compare our models with typical baselines proposed in (Chen et al., 2020). We also present results of a contemporaneous work LogicalFactChecker (Zhong et al., 2020) which reports the best performance in the literature. Details of baselines are discussed in related work (Section 2).

5 Results

Overall Performance Table 2 presents the results of different verification models. Our proposed

Model	Val	Test	Test (simple)	Test (complex)	Small Test
Human Performance	-	-	-	-	92.1
Table-BERT-Horizontal-S+T-Concatenate	50.7	50.4	50.8	50.0	50.3
Table-BERT-Vertical-S+T-Template	56.7	56.2	59.8	55.0	56.2
Table-BERT-Vertical-T+S-Template	56.7	57.0	60.6	54.3	55.5
Table-BERT-Horizontal-S+T-Template	66.0	65.1	79.0	58.1	67.9
Table-BERT-Horizontal-T+S-Template	66.1	65.1	79.1	58.2	68.1
LPA-Voting w/o Discriminator	57.7	58.2	68.5	53.2	61.5
LPA-Weighted-Voting	62.5	63.1	74.6	57.3	66.8
LPA-Ranking w/ Discriminator	65.2	65.0	78.4	58.5	68.6
LogicalFactChecker (Zhong et al., 2020)	71.8	71.7	85.4	65.1	74.3
ProgVGAT	**74.9**	**74.4**	**88.3**	**67.6**	**76.2**

Table 2: Performance (accuracy) of different models on TABFACT. For `Table-BERT` baseline, different strategies of linearizing tables to bridge semantic gap with statements. *Horizontal* and *Vertical* refer to horizontally or vertically traverse items in tables respectively. *S* denotes statements, *T* denotes tables, + indicates concatenation order between *S* and *T*. *Concatenate* refers to directly concatenating items in tables. *Template* convert items in tables into sentences with pre-defined templates. For `LPA` baseline, to select one program among all candidates for each statement, they take either a (weighted) voting strategy or a discriminator.

method obtains accuracy of 74.4% on the test set, achieving new state-of-the-art in this dataset.

For `Table-BERT` baseline, it leverages pre-trained language models to measure semantic similarities for table-statement pairs. `LPA` derives a synthesized program best describing the statement-table pair, and executes derived program against semi-structured tables for verification. Our proposed method proposes a verbalization and graph attention network for fact verification. It integrates execution of programs into pre-trained language models, outperforming `Table-BERT` and `LPA` with a large margin.

Compared with `LogicalFactChecker`, our proposed method is built to leverage operation execution evidences and the inherent structures information with verbalization and graph attentions. While `LogicalFactChecker` focuses on pruning irrelevant rows and columns in evidence table with programs and utilizing structures of operations with module networks. Our proposed method achieves better results (74.4%) than `LogicalFactChecker` (71.7%). The result suggests the effectiveness of our proposed method by introducing executed operation results. Symbolic operations performed on evidence tables provide useful information for verification.

Although the proposed model improves the state-of-the-art performance on the entire dataset as well as all subsets, we can see that the *complex* subset of the problem remains hard to solve. On the

small test set where the human upper-bound performance is provided (92.1%), there is still a large gap between the system and human performance. We hope our work will serve asOther for base for further work on this problem.

Model	Val	Test
Table-BERT w/ prog	70.3	70.0
LogicalFactChecker	71.8	71.7
Table-BERT w/ verb. prog	71.8	71.6
Table-BERT w/ verb. prog exec	72.4	72.2
ProgVGAT	**74.9**	**74.4**

Table 3: Results of different ways of using operations.

Model	Val	Test
ProgVGAT w/o graph attention	73.6	73.4
ProgVGAT	**74.9**	**74.4**

Table 4: Ablation results (accuracy) that shows the effectiveness of our graph attention component.

Effect of Program Operations A key component of our proposed framework is utilizing the executed operation results for verification, in which we introduce a verbalization algorithm to transform the recursive execution of symbolic operations into evidence sentences. We further investigate different ways of leveraging operations: (1) `Table-BERT w/ prog` directly concatenates

			Final Verification		
			Val	Test	ΔTest
LPA	Val	Test	73.3	72.8	-
w/ CE	65.2	65.0			
LPA+ BERT	Val	Test	73.9	73.4	+0.6
w/ CE	67.7	67.3			
LPA +BERT	Val	Test	74.9	74.4	**+1.6**
w/ Margin loss	**69.4**	**68.5**			

Table 5: Accuracy of different program selection models and corresponding final verification performance based on verbalized evidence derived from each program selection model.

the derived program z^* with the original table as new evidence, and employs `Table-BERT` on the new evidence-statement pair; (2) `Table-BERT w/ verb. prog` differs from `Table-BERT w/ prog` in converting the derived program z into sentences with templates proposed in verbalization algorithm in Sec. 3.3 for verification (3) `Table-BERT w/ verb. prog exec` verbalizes program along with *execution* results using the algorithm proposed in Sec. 3.3.

Table 3 shows the results. Compared to directly concatenate structured program with original Table-BERT, `Table-BERT w/ verb. prog` converts the structured program into natural sentences, and achieves better results. The result demonstrates the importance of eliminating the semantic discrepancy between structured data and natural language in BERT based verification model. `Table-BERT w/ verb. prog exec` leverages executed operation results with the verbalization algorithm and outperforms `Table-BERT w/ verb. prog` as well as `LogicalFactChecker`, The execution output provides complementary clues from evidence tables and improves the verification results. Our proposed `ProgVGAT` further leverages structures in program execution and boost the performance. The results confirm the effectiveness of leveraging symbolic operations with our method.

Effect of Graph Attention We investigate the necessity of leveraging structure information in program execution for verification. We present a simpler version of our model by removing the graph attention module and integrating verbalized program execution with gated attention mechanism in Eq. 7. Table 4 presents the results. By removing the graph attention network, the verification accuracy drops 1.3% on the validation set and 1.0% on the test set, respectively. The results show that

integrating the structures of programs in the graph attention network is important for verification.

Effect of Derived Programs We investigate the effectiveness of accurate program selection models for final verification. Table 5 represents the results of our model on leveraging different programs produced by different program selection models.

We start by introducing different program selection models in Table 5. `LPA w/ CE` (Chen et al., 2020) applies Transformer encoders without pre-training stage (i.e., BERT) to compute semantic similarity between candidate programs and statements, and optimizes via a cross entropy loss in Eq. 1. It achieves a 65.0% accuracy on the test set. Our proposed program selection denoted as `LPA+BERT w/ Margin loss`, replacing transformer encoders with BERT and optimizing the model with our proposed margin loss, can effectively improve the program accuracy to 68.5%. Comparing with the accuracy of 67.3% obtained by `LPA+BERT w/ CE`, which is optimized with cross entropy loss instead of margin loss, we can conclude that our proposed margin loss plays a positive role in program selection.

Accordingly, we compare the final verification results based on different programs selected by the above models. Our proposed method leverages programs produced by `LPA+BERT w/ Margin loss` obtains better verification results (e.g., 74.4% on test set) compared with using programs derived by `LPA` (e.g., 72.8% on test set). The results indicate more accurate programs can provide more useful operation information and is beneficial for the final verification.

Qualitative Analysis We provide an example, where integrating program execution information with our proposed verbalization and graph attention network can yield the right label. In Fig. 3, the

Table	Year	Gold	Silver	Bronze
	2002	Gunilla Svärd	Brigitte Wolf	Birgitte Husebye
	2004	Hanne Staff	Dainora Alšauskaitė	Tatiana Ryabkina
	2006	Minna Kauppi	Marianne Andersen	Heli Jukkola
	2008	Heli Jukkola	Merja Rantanen	Minna Kauppi
	2010	Simone Niggli	Signe Soes	Lena Eliasson
	2012	Simone Niggli	Minna Kauppi	Tatiana Ryabkina

Statement Tatiana Ryabkina has won the bronze medal more times than Lena Eliasson has.

Label ENTAILED

Program *greater { count { filter_eq { all_rows ; bronze ; tatiana ryabkina } } ; count { filter_eq { all_rows ; bronze ; lena eliasson } } }*

Verbalized Evidence:

V1 The table where column bronze equal to Tatiana Ryabkina are row 2 , row 6.
V2 The number of the column bronze equal to Tatiana Ryabkina is 2.
V3 The table where column bronze equal to Lena Eliasson is row 5.
V4 The number of the column bronze equal to Lena Eliasson is 1.
V5 2 is greater than 1.

Figure 3: An example in qualitative analysis.

statement requires symbolic manipulation on counting bronze medals of two players and compare the number of medals of them. First, program selection produces a semantic-consistent program for the statement and then correctly captures the semantic correlations between the phrase "more times than" and operations $greater, count$. With the derived program, a verbalization algorithm executes operations over the table and produces sentences describing useful operation results. For example, "the number of the column bronze equal to Tatiana Ryabkina is 2", "the number of the column bronze equal to Lena Eliasson is 1", and "2 is greater than 1". Then the sentences are integrated into the verification model with the graph attention network to perform the final verification.

6 Conclusions

In this paper, we propose a framework centered around *programs* and *execution* to provide symbolic manipulations for table fact verification. We propose a verbalization technique together with a graph-based verification network to aggregate and fuse evidences inherently embedded in programs and the original tables for fact verification. Moreover, we design a new training strategy adapting margin loss for the program selection module to derive more accurate programs. The experiments show that the proposed model improves the state-of-the-art performance to a 74.4% accuracy on the benchmark dataset TABFACT. Our studies also reveal the importance of accurate program acquisition for improving the performance of table fact verification. In the future, we will investigate the

properties of our proposed method on verifying statements with more complicated operations and explore the explainability of the model.

References

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1533–1544.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1415–1425, Baltimore, Maryland. Association for Computational Linguistics.

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 2015, Lisbon, Portugal, September 17-21, 2015*, pages 632–642. The Association for Computational Linguistics.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. 2016. A fast unified model for parsing and sentence understanding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017a. Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, Vancouver, Canada. Association for Computational Linguistics.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Diana Inkpen, and Si Wei. 2017b. Neural natural language inference models enhanced with external knowledge. *arXiv preprint arXiv:1711.04289*.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2017c. Enhanced LSTM for natural language inference. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1657–1668, Vancouver, Canada. Association for Computational Linguistics.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2017d. Enhanced LSTM for natural language inference. pages 1657–1668.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2017e. Recurrent neural network-based sentence encoder with gated attention for natural language inference. *arXiv preprint arXiv:1708.01353*.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020. Tabfact: A large-scale dataset for table-based fact verification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Ido Dagan, Bill Dolan, Bernardo Magnini, and Dan Roth. 2010. Recognizing textual entailment: Rational, evaluation and approaches. *Journal of Natural Language Engineering*, 4.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190. Springer.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. 2019a. Neural logic machines. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019b. Unified language model pre-training for natural language understanding and generation. In *Advances in Neural Information Processing Systems*, pages 13042–13054.

Reza Ghaeini, Sadid A. Hasan, Vivek Datla, Joey Liu, Kathy Lee, Ashequl Qadir, Yuan Ling, Aaditya Prakash, Xiaoli Fern, and Oladimeji Farri. 2018. DR-BiLSTM: Dependent reading bidirectional LSTM for natural language inference. 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 1460–1469, New Orleans, Louisiana. Association for Computational Linguistics.

Ben Goodrich, Vinay Rao, Peter J Liu, and Mohammad Saleh. 2019. Assessing the factual accuracy of generated text. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 166–175.

Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. 2020. Neural module networks for reasoning over text. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. pages 1051–1062.

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.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Wojciech Kryściński, Bryan McCann, Caiming Xiong, and Richard Socher. 2019. Evaluating the factual consistency of abstractive text summarization. *arXiv*, pages arXiv–1910.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 271–281, Baltimore, Maryland. Association for Computational Linguistics.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural symbolic machines: Learning semantic parsers on Freebase with weak supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23–33, Vancouver, Canada. Association for Computational Linguistics.

Yang Liu and Mirella Lapata. 2019. Text summarization with pretrained encoders. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3728–3738. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

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.

Bill MacCartney and Christopher D Manning. 2008. Modeling semantic containment and exclusion in natural language inference. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 521–528.

Bill MacCartney and Christopher D Manning. 2009. *Natural language inference*. Citeseer.

Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. 2016. Neural programmer: Inducing latent programs with gradient descent. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Feng Nie, Yunbo Cao, Jinpeng Wang, Chin-Yew Lin, and Rong Pan. 2018. Mention and entity description co-attention for entity disambiguation. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 5908–5915. AAAI Press.

Yixin Nie, Haonan Chen, and Mohit Bansal. 2019. Combining fact extraction and verification with neural semantic matching networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6859–6866.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255, Austin, Texas. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. 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 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

A. Radford. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Hannah Rashkin, Eunsol Choi, Jin Yea Jang, Svitlana Volkova, and Yejin Choi. 2017. Truth of varying shades: Analyzing language in fake news and political fact-checking. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2931–2937.

Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. 2015. Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1132–1142, Lisbon, Portugal.

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, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 809–819. Association for Computational Linguistics.

James Thorne, Andreas Vlachos, Oana Cocarascu, Christos Christodoulopoulos, and Arpit Mittal. 2019. The fever2. 0 shared task. In *Proceedings of the Second Workshop on Fact Extraction and VERification (FEVER)*, pages 1–6.

Vaibhav Vaibhav, Raghuram Mandyam Annasamy, and Eduard H. Hovy. 2019. Do sentence interactions matter? leveraging sentence level representations for fake news classification. In *Proceedings of the Thirteenth Workshop on Graph-Based Methods for Natural Language Processing, TextGraphs@EMNLP*

2019, Hong Kong, November 4, 2019, pages 134–139. Association for Computational Linguistics.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks.

Xiaoyu Yang, Xiaodan Zhu, Huasha Zhao, Qiong Zhang, and Yufei Feng. 2019. Enhancing unsupervised pretraining with external knowledge for natural language inference. In *Canadian Conference on Artificial Intelligence*, pages 413–419. Springer.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada. Association for Computational Linguistics.

Wenpeng Yin and Dan Roth. 2018. TwoWingOS: A two-wing optimization strategy for evidential claim verification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 105–114, Brussels, Belgium. Association for Computational Linguistics.

Takuma Yoneda, Jeff Mitchell, Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. 2018. Ucl machine reading group: Four factor framework for fact finding (hexaf). In *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)*, pages 97–102.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005*, pages 658–666. AUAI Press.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Wanjun Zhong, Duyu Tang, Zhangyin Feng, Nan Duan, Ming Zhou, Ming Gong, Linjun Shou, Daxin Jiang, Jiahai Wang, and Jian Yin. 2020. LogicalFactChecker: Leveraging logical operations for fact checking with graph module network. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6053–6065, 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 Appendices

A.1 Statistics of TABFACT Dataset

Table 6 provides the statistics of TABFACT (Chen et al., 2020), a recent large-scale table-based fact verification dataset on which we evaluate our method. Each evidence table comes along with 2 to 20 statements, and consists of 14 rows and 5-6 rows in average.

A.2 Pre-defined Operations in Program Selection

Programs consists of operations, and the definition of operations are listed in Table 7, mainly following (Chen et al., 2020).

A.3 Pre-defined Templates for Verbalization

In our proposed framework, there are 50 pre-defined operations, details are in Table 7. For each operation, we define templates for operation and its executed result. There are three types of executed results: (1) string or number type; (2) boolean type; (3) view or row type, where it is a sub-table or rows extracted from the evidence table.

We represent templates for different types of operations accordingly. The detailed templates are listed in the following Table 8, Table 9 and Table 10.

Split	Sentence	Table	Row	Col
Train	92,283	13,182	14.1	5.5
Val	12,792	1,696	14.0	5.4
Test	12,779	1,695	14.2	5.4

Table 6: Statistics of TABFACT and the split of Train/Val/Test.

Operations	Arguments	Output	Function
count	View	Number	Return the number of rows in the View
	View, Header String, Cell String/ Number	Bool	Return whether the cell string/number exists under the Header Column of the given View
without	View, Header String, Cell String/ Cell Number	Bool	Return whether the cell string/number does not exist under the Header Column of the given view
none	String	Bool	Whether the string represents None, like "None", "No", "-", "No information provided"
before/after	Row, Row	Row	Returns whether row1 is before/after row2
first/second/ third/fourth	View, Row	Bool	Returns whether the row is in the first/second/ third position of the view
avg/sum/ max/min	View, Header String	Number	Returns the average/summation/max/min value under the Header Column of the given view
argmin/ argmax	View, Header String	Row	Returns the row with the max/min value under the Header Column of the given view
Hop	Row, Header String	Number/ String	Returns the cell value under the Header Column of the given row
diff/add	Number, Number	Number	Perform arithmetic operations on two numbers
greater/less	Number, Number	Bool	Returns whether the first number is greater/less than the second number
Equal/ Unequal	String, String/ Number, Number	Bool	Compare two numbers or strings to see whether they are the same
filter_eq/ filter_greater/ filter_less/ filter_greater_or_equal/ filter_less_or_equal	View, Header String, Number	View	Returns the subview of the given with the cell values under the Header column greater/less/eq/... against the given number
all_eq/all_greater/ all_less/ all_greater_or_equal/ all_less_or_equal	View, Header String, Number	Bool	Returns the whether all of the cell values under the Header column are greater/less/eq/... against the given number
and/or	Bool, Bool	Bool	Returns the Boolean operation results of two inputs

Table 7: Details of pre-defined operations.

Operation operator, [arguments], answer	Templates	
	Operation	Operation Results
count, [verb_arg1], ans	the number of [verb_arg1]	the number of [verb_arg1] is [string(ans)]
avg/sum/max/min, [verb_arg1,verb_arg2], ans	average/sum/maximum/minimum [verb_arg1] where column [verb_arg2]	average/sum/maximum/minimum [verb_arg1] where column [verb_arg2] is [string(ans)]
add/diff, [verb_arg1, verb_arg2], ans	sum/difference of [verb_arg1] and [verb_arg2]	sum/difference of [verb_arg1] and [verb_arg2] is [string(ans)]
uniq_num/uniq_string, [verb_arg1, verb_arg2],ans	the unique value of [verb_arg1] in column [verb_arg2]	the unique value of [verb_arg1] in column [verb_arg2] is [string(ans)]
most_freq, [verb_arg1, verb_arg2],ans	the most frequent value of [verb_arg1] in column [verb_arg2]	the most frequent value of [verb_arg1] in column [verb_arg2] is [string(ans)]
half/one_third, [verb_arg1], ans	half/one third of value in [verb_arg1]	half/one_third of value in [verb_arg1] is [string(ans)]
num_hop, [verb_arg1, verb_arg2],ans	the first value of [verb_arg1] where column [verb_arg2]	the first value of [verb_arg1] where column [verb_arg2] is [string(ans)]

Table 8: Templates for operations with string or number type executed results.

Operation operator, [arguments], answer	Templates	
	Operation	Operation Results
only, [verb_arg1], ans:true/false	number of rows in [verb_arg1]	number of rows in [verb_arg1] [true:is/false:is not] one
several, [verb_arg1], ans:true/false	number of rows in [verb_arg1]	number of rows in [verb_arg1] [true:is/false:is not] more than one
zero/none, [verb_arg1], ans:true/false	the [verb_arg1]	the [verb_arg1] [true:is/false:is not] zero/none
first/second/third/fourth, [verb_arg1, verb_arg2], ans:true/false	the first/second/third/fourth row [verb_arg2] in [verb_arg1]	the first/second/third/fourth row in [verb_arg1] [true:is/false:is not] row [verb_arg2]
and/or, [verb_arg1, verb_arg2], ans:true/false	[verb_arg1] and/or [verb_arg2]	[verb_arg1] and/or [verb_arg2] [true:is/false:is not] true
greater/less, [verb_arg1, verb_arg2], ans:true/false	[verb_arg1] greater/less than [verb_arg2]	[verb_arg1] [true:is/false: is not] greater/less than [verb_arg2]
equal, [verb_arg1, verb_arg2], ans:true/false	[verb_arg1] equal to [verb_arg2]	[verb_arg1] [true:is/false: is not] equal to [verb_arg2]
unequal, [verb_arg1, verb_arg2], ans:true/false	[verb_arg1] not equal to [verb_arg2]	[verb_arg1] [true:is/false: is not] not equal to [verb_arg2]
with/without, [verb_arg1, verb_arg2, verb_arg3], ans:true/false	[verb_arg1] where column [verb_arg2] with/without value [verb_arg3]	[verb_arg1] where column [verb_arg2] with/without value [verb_arg3] [true:is/false:is not] true
all_equal, [verb_arg1, verb_arg2, verb_arg3], ans:true/false	[verb_arg1] where column [verb_arg2] all equal to value [verb_arg3]	[verb_arg1] where column [verb_arg2] all equal to value [verb_arg3] [true:is/false:is not] true
all_less/all_greater, [verb_arg1, verb_arg2, verb_arg3], ans:true/false	[verb_arg1] where column [verb_arg2] all less/greater than value [verb_arg3]	[verb_arg1] where column [verb_arg2] all less/greater than value [verb_arg3] [true:is/false:is not] true
all_less_or_equal/ all_greater_or_equal, [verb_arg1, verb_arg2, verb_arg3], ans:true/false	[verb_arg1] where column [verb_arg2] all less/greater than or equal to value [verb_arg3]	[verb_arg1] where column [verb_arg2] all less/greater than or equal to value [verb_arg3] [true:is/false:is not] true

Table 9: Templates for operations with boolean type executed results.

Operation operator, [arguments], answer	Templates	
	Operation	Operation Results
before/after, [verb_arg1, verb_arg2], ans	[verb_arg1] before/after [verb_arg2]	[verb_arg1] before/after [verb_arg2] is row [indices of ans]
argmax/argmin, [verb_arg1, verb_arg2], ans	row where column [verb_arg2] with maximum/minimum value in [verb_arg1]	row where column [verb_arg2] with maximum/minimum value in [verb_arg1] is row [indices of ans]
filter_eq, [verb_arg1, verb_arg2, verb_arg3], ans	[verb_arg1] where column [verb_arg2] equal to value [verb_arg3]	[verb_arg1] where column [verb_arg2] equal to value [verb_arg3] is row [indices of ans]
filter_less/filter_greater, [verb_arg1, verb_arg2, verb_arg3], ans:true/false	[verb_arg1] where column [verb_arg2] less/greater than value [verb_arg3]	[verb_arg1] where column [verb_arg2] less/greater than value [verb_arg3] is row [indices of ans]
less_or_equal/ greater_or_equal, [verb_arg1, verb_arg2, verb_arg3], ans:true/false	[verb_arg1] where column [verb_arg2] less/greater than or equal to value [verb_arg3]	[verb_arg1] where column [verb_arg2] less/greater than or equal to value [verb_arg3] is row [indices of ans]

Table 10: Templates for operations with view or row type executed results.