Fact-Checking Complex Claims with Program-Guided Reasoning

Liangming Pan[1,2], Xiaobao Wu[3], Xinyuan Lu[4], Anh Tuan Luu[3], William Yang Wang[1], Min-Yen Kan[4], Preslav Nakov[2]

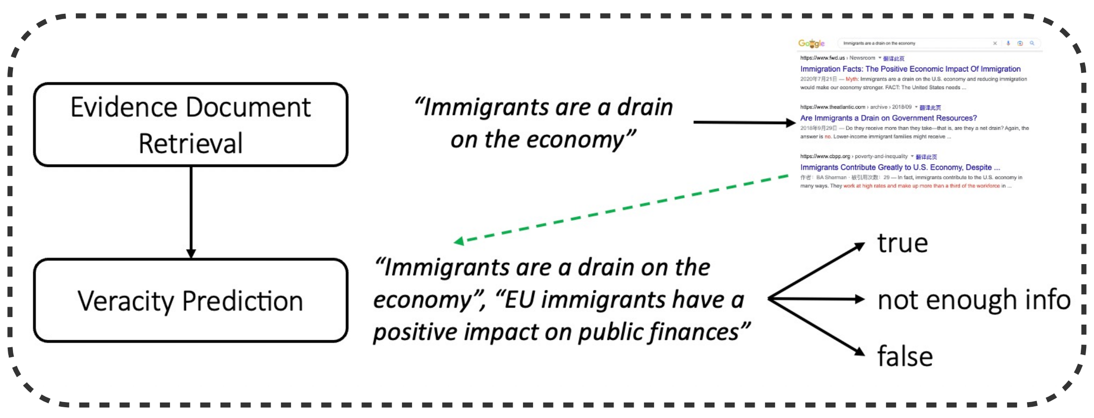
[1]University of California, Santa Barbara [2]MBZUAI [3]Nanyang Technological University [4]National University of Singapore

liangmingpan@ucsb.edu, william@cs.ucsb.edu, kanmy@comp.nus.edu.sg, preslav.nakov@mbzuai.ac.ae

Introduction

What is Fact-Checking?

Given a claim made by a claimant, to find a collection of evidence and provide a verdict about the claim's veracity label based on the evidence.



Settings:

1) Gold Evidence: the ground-truth evidence is given.
2) Open-book: a large textual corpus is given as the source of evidence.
3) Closed-book: no source of evidence is available.

Challenges:

Data Efficiency

- Human annotation is often time-consuming and costly.
- Fact-checking with minimal or no training data.

Explanablility

- The system should not only predict the veracity of the claim, but it should also provide a clear explanation of its reasoning process to help users understand and trust the results.

Deep Reasoning

- Evaluating the veracity of real-world claims often involves collecting multiple pieces of evidence and applying complex reasoning.

Links

Paper

@PanLiangming

Data & Codes

Datasets

HOVER (Jiang et al., 2020)

- 1,126 two-hop claims
- 1,835 three-hop claims
- 1,039 four-hop claims

FEVEROUS (Aly et al., 2021)

- We selected 2,962 claims that require exclusively textual evidence.

Limitations

Decomposition can be hard

- For many real-world claims, the reasoning is implicit.

"Aristotle couldn't have used a laptop"

answer_1 = Question("When did Aristotle live?");
answer_2 = Question("When was the laptop invented?");
fact_1 = Verify("answer_1 is before answer_2.");
label = Predict(fact_1)

Out-of-domain generalization

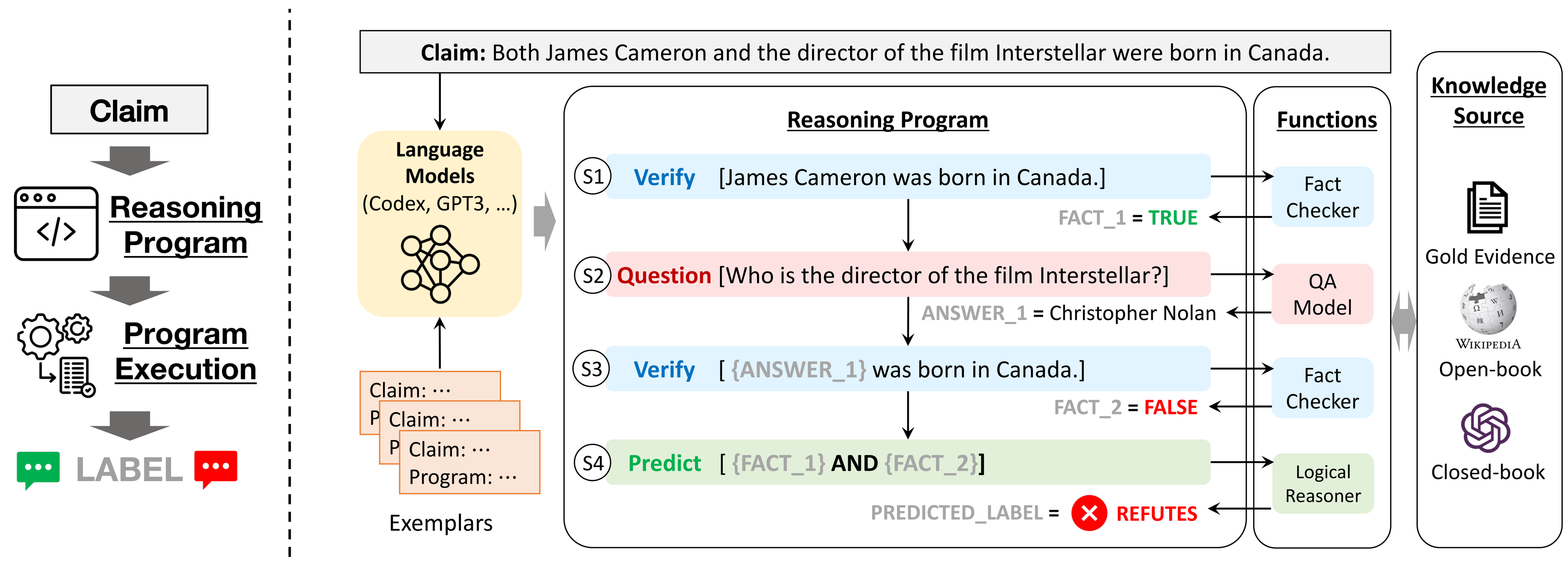
- A fixed set of in-context examples is insufficient to teach model how to decompose every possible claim in real world.

Computation efficiency

- Computational cost of ~4-5x higher than end-to-end FLAN-T5 model.

Approach: Program-Guided Fact-Checking

General Framework: Program-guided Fact-Checking (ProgramFC)

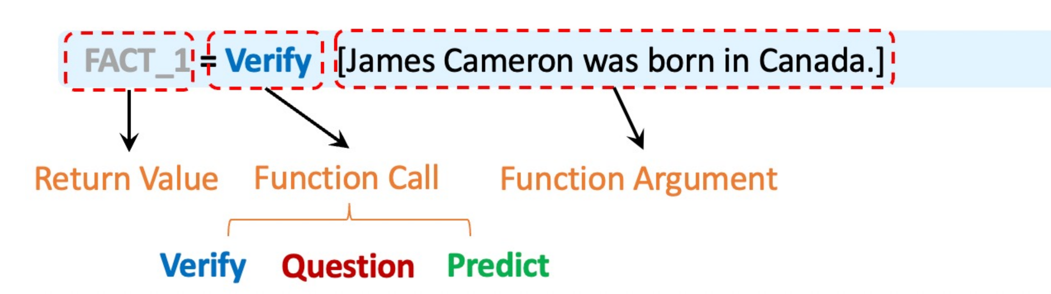


Program Generation

- Given the input claim C, a planner P generates a reasoning program $P = [S_1, \cdots, S_n]$, which consists of n sequentially ordered reasoning steps S_i.

Reasoning Step

- Each reasoning step is defined as a tuple $S_i = (f_i, A_i, V_i)$
- f_i specifies the sub-task function $f_i \in \mathcal{F}$
- A_i is the arguments passed to the function f_i
- V_i is the variable that stores the returned result from the function call $f_i(A_i)$



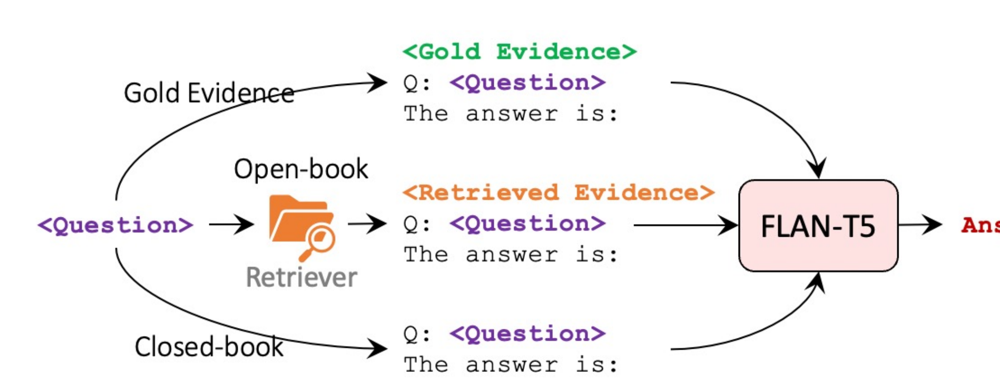
In-context Learning

- We base our program generator on Codex and GPT-3.5.
- We utilize their few-shot generalization ability to learn our grammar from a small number of in-context examples.
- Aggregating Reasoning Programs: We generate a diverse set of N candidate reasoning programs, since there might be multiple reasoning paths that can reach the final veracity label.

```
'''Generate a python-like program that describes the reasoning steps
   required to verify the claim step-by-step. You can call three functions
   in the program: 1. Question() to answer a question; 2. Verify() to
   verify a simple claim; 3. Predict() to predict the veracity label.'''

# The claim is that Both James Cameron and the director of the film
   Interstellar were born in Canada.
def program():
    fact_1 = Verify("James Cameron was born in Canada.")
    Answer_1 = Question("Who is the director of the film Interstellar?")
    fact_2 = Verify("{Answer_1} was born in Canada.")
    label = Predict(fact_1 and fact_2)

(··· more in-context examples here ···)

# The claim is that <input_claim>
def program():
```

Program Execution

- During execution, we sequentially parses the reasoning steps in P with a program interpreter.
- For each step $S_i = (f_i, A_i, V_i)$, the interpreter calls the corresponding off-the-shelf sub-task function f_i.
- We base the sub-functions on the FLAN-T5 model.



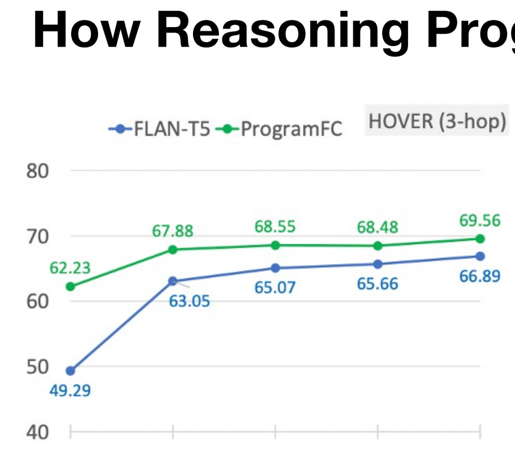
Experimental Results

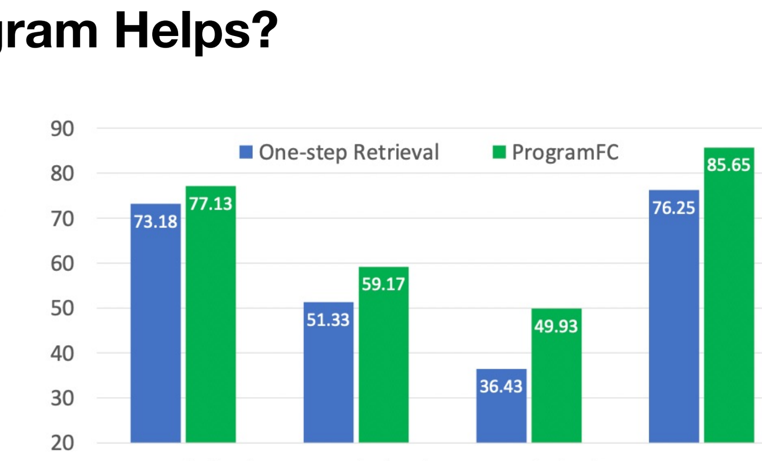
Main Results

- ProgramFC achieves the best performance on 7 out of 8 evaluations.
- ProgramFC is more effective on deeper claims.
- Aggregating reasoning programs is helpful.

	Few-shot learning models	HOVER (2-hop)		HOVER (3-hop)		HOVER (4-hop)		FEVEROUS-S	
		Gold	Open	Gold	Open	Gold	Open	Gold	Open
I	BERT-FC (Soleimani et al., 2020)	53.40	50.68	50.90	49.86	50.86	48.57	74.71	51.67
	ListT5 (Jiang et al., 2021)	56.15	52.56	53.76	51.89	51.67	50.46	77.88	54.15
II	RoBERTa-NLI (Nie et al., 2020)	74.62	63.62	62.23	53.99	57.98	52.40	88.28	57.80
	DeBERTaV3-NLI (He et al., 2021)	**77.22**	68.72	65.98	60.76	60.49	56.00	91.98	58.81
	MULTIVERS (Wadden et al., 2022b)	68.86	60.17	59.87	52.55	55.67	51.86	86.03	56.61
III	GPT3-Codex (Chen et al., 2021)	70.63	65.07	66.46	56.63	63.49	57.27	89.77	62.58
	FLAN-T5 (Chung et al., 2022)	73.69	69.02	65.66	60.23	58.08	55.42	90.81	63.73
IV	ProgramFC (N=1)	74.10	69.36	66.13	60.63	65.69	**59.16**	91.77	67.80
	ProgramFC (N=5)	75.65	**70.30**	**68.48**	**63.43**	**66.75**	57.74	**92.69**	**68.06**

How Reasoning Program Helps?





- The performance decrease is less obvious for ProgramFC with decreasing model size. The high-level planning offered by reasoning programs alleviates the demand on strong, large-scale models.
- In the open-book setting, ProgramFC significantly outperforms one-step retrieval. Iteratively retrieving information guided by the reasoning program leads to better results.

Error Analysis

Error Type	Proportion (%)		
	2-hop	3-hop	4-hop
Syntax error	**0%**	**0%**	**0%**
Semantic error	**29%**	**38%**	**77%**
Token	8%	20%	18%
Structure	19%	13%	57%
Subtask	2%	5%	2%
Incorrect execution	**71%**	**62%**	**23%**

Semantic Error — Subtask: missing / redundant / incorrect sub-task calls

```
Example 5:
The musician, who founded Morningwood with Max Green, is older than Max Green.

Predicted Program:
answer_1 = Question("Who founded Morningwood with Max Green?")
answer_2 = Question("When was Max Green born?")
~~answer_3 = Question("When was the musician born?")~~
fact_1 = Verify("{answer_3} is older than {answer_2}.") → {answer_1} is older than {answer_2}.
label = Verify(fact_1)
```