Prompting-Based UCLID5 Code Generation Using an Intermediate Language

Haley Lepe[1], Federico Mora[2], Sanjit A. Seshia[2]
[1]MiraCosta College
[2]University of California, Berkeley

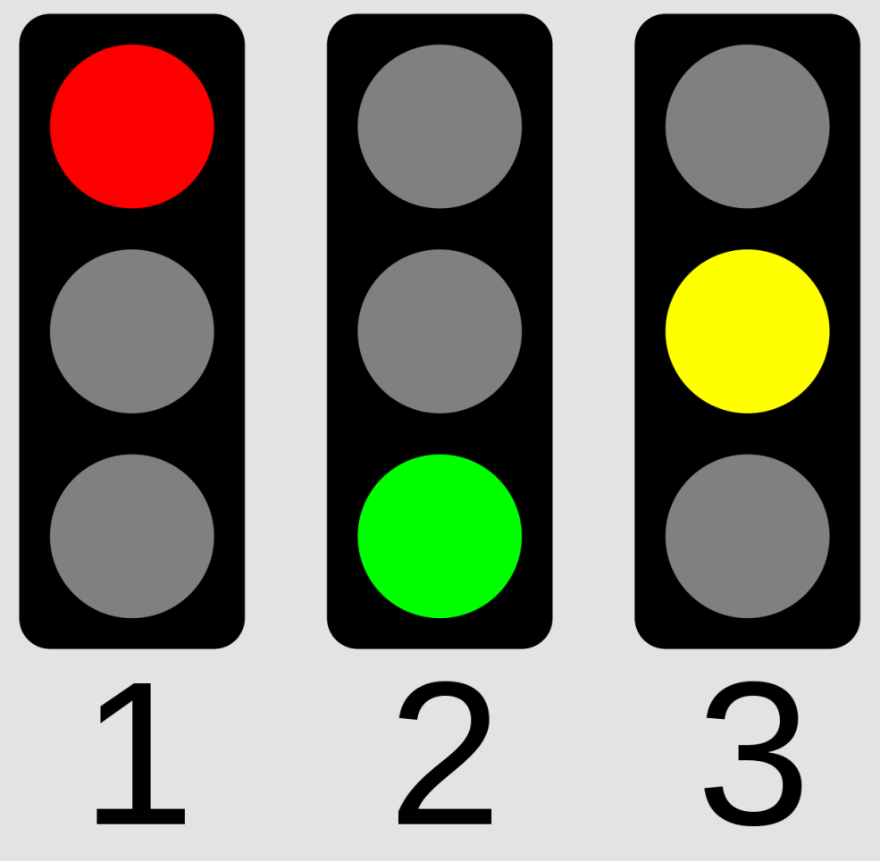
Abstract

UCLID5 is a Domain-Specific Language (DSL) for the formal modeling, specification, and verification of systems and programming in UCLID5, just as in any language, can be laborious. Although recent advancements in Large Language Models (LLM) have eased the manual burden of programming, their application to formal reasoning DSL like UCLID5 has been limited, with difficulties in generating even syntactically correct code.

In this work, we propose a new prompting technique that will help LLMs generate UCLID5 code from natural language queries. This technique uses LLMs ability to generate syntactically correct Python code to produce UCLID5 code.

Background

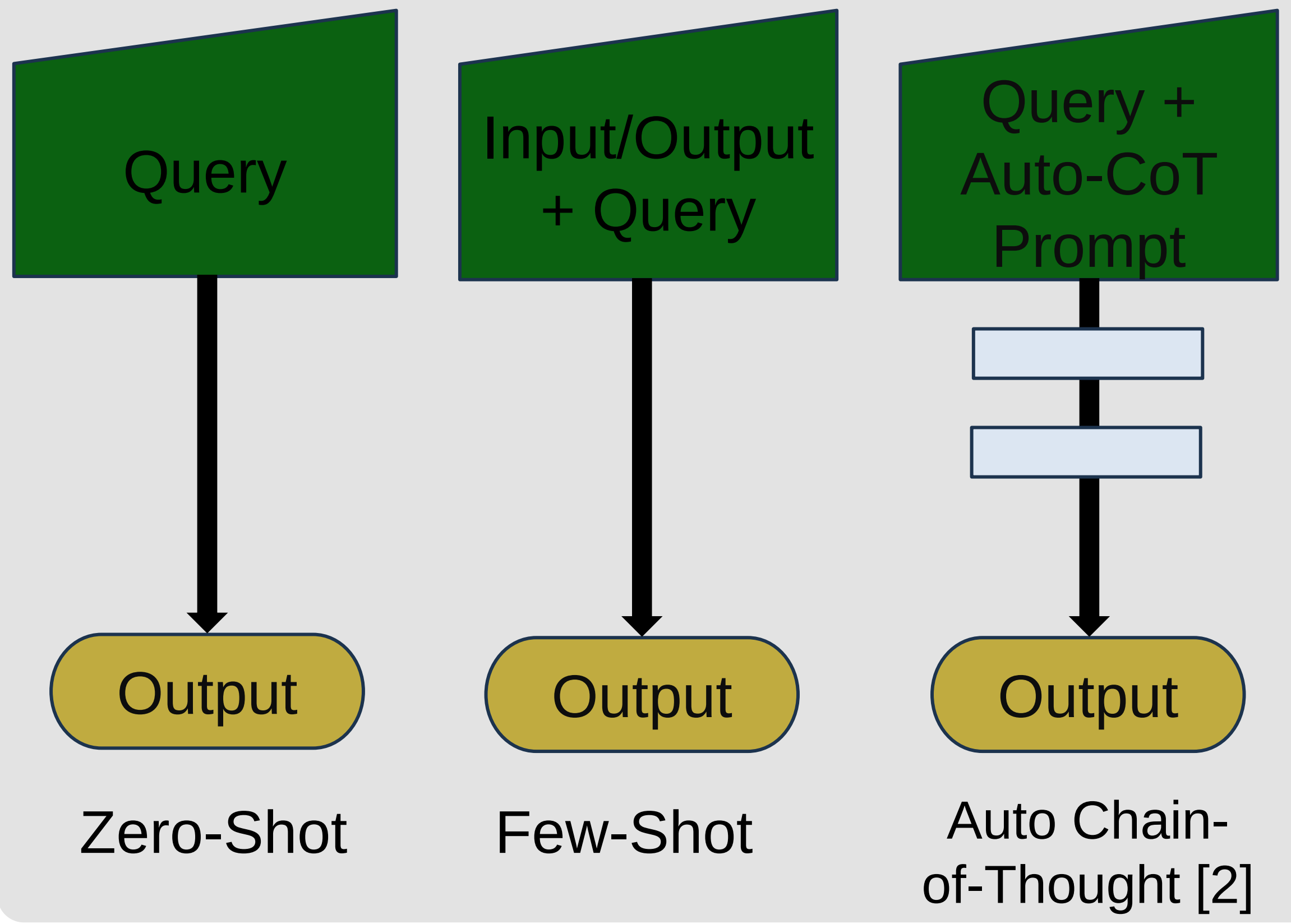
What is UCLID5?



- Users can:
  - Model hardware or software systems
  - Prove properties about their models
- EX: Traffic Light: Finding bugs is crucial!

Prompting

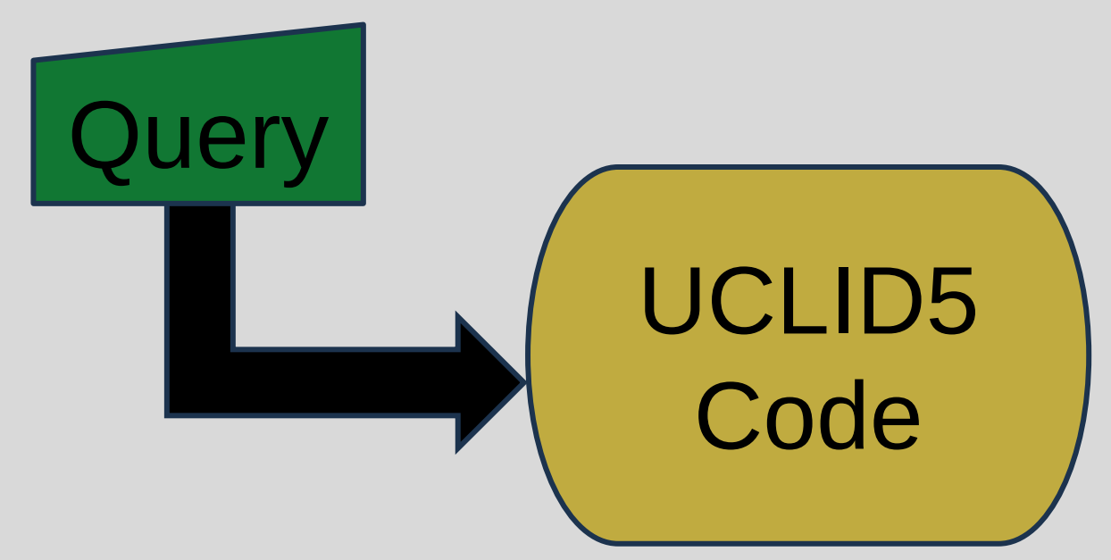
- Guide LLM outputs by modifying queries



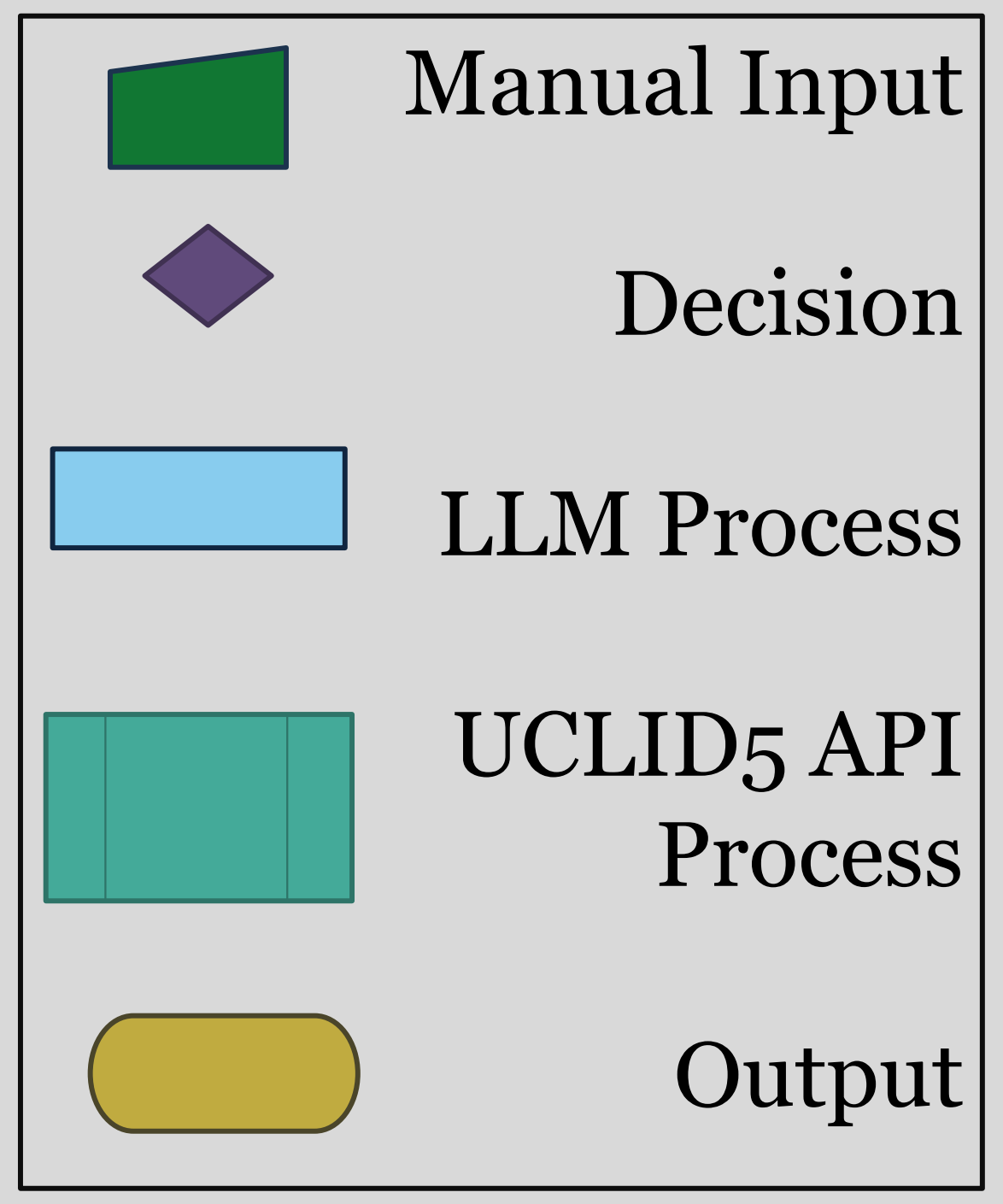
Contact Information- haleylepe@gmail.com

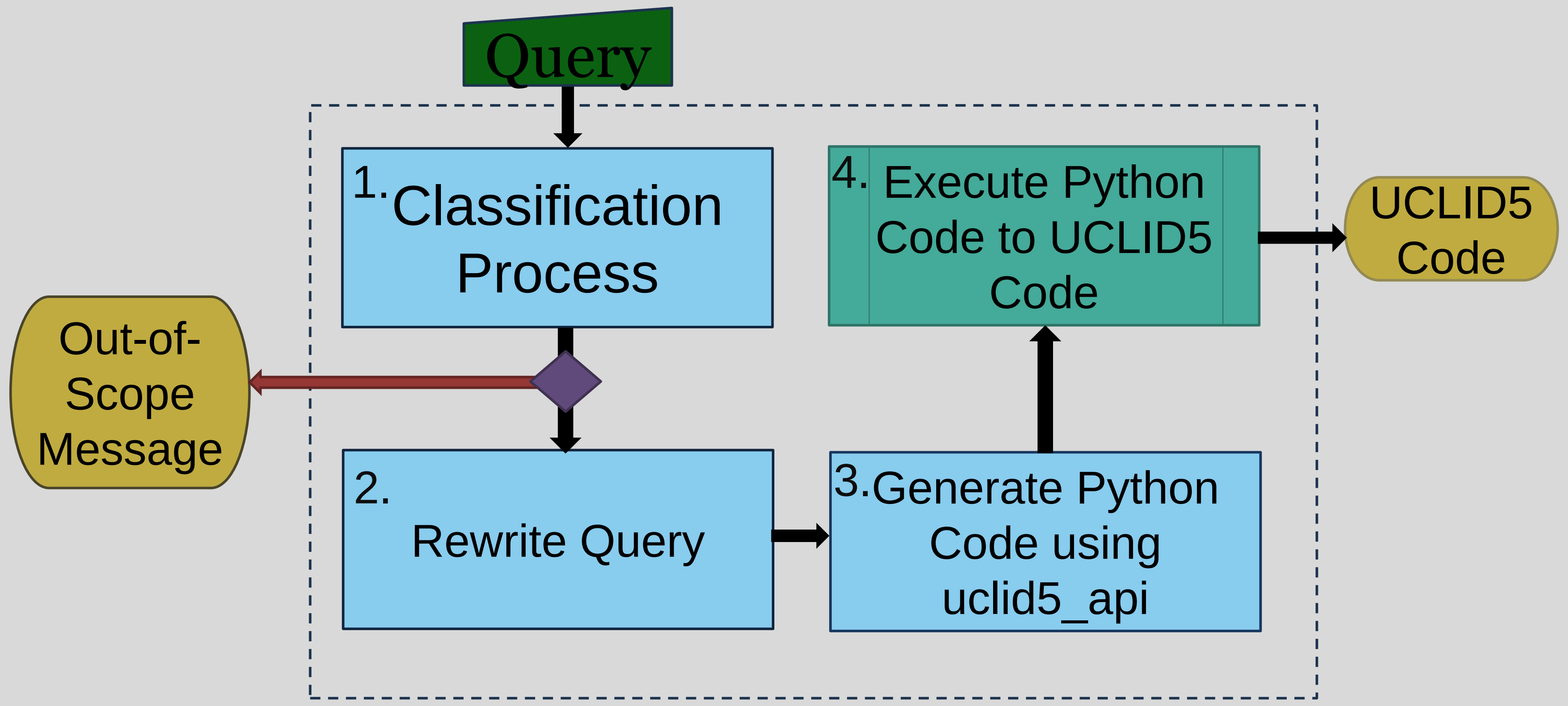
Objective: Query-to-Code

- Writing code can be difficult
- LLMs can assist with this process
- Minimal data for domain-specific languages limits LLMS
- Can prompting help LLMs produce UCLID5 code?



Methods: UPYGEN

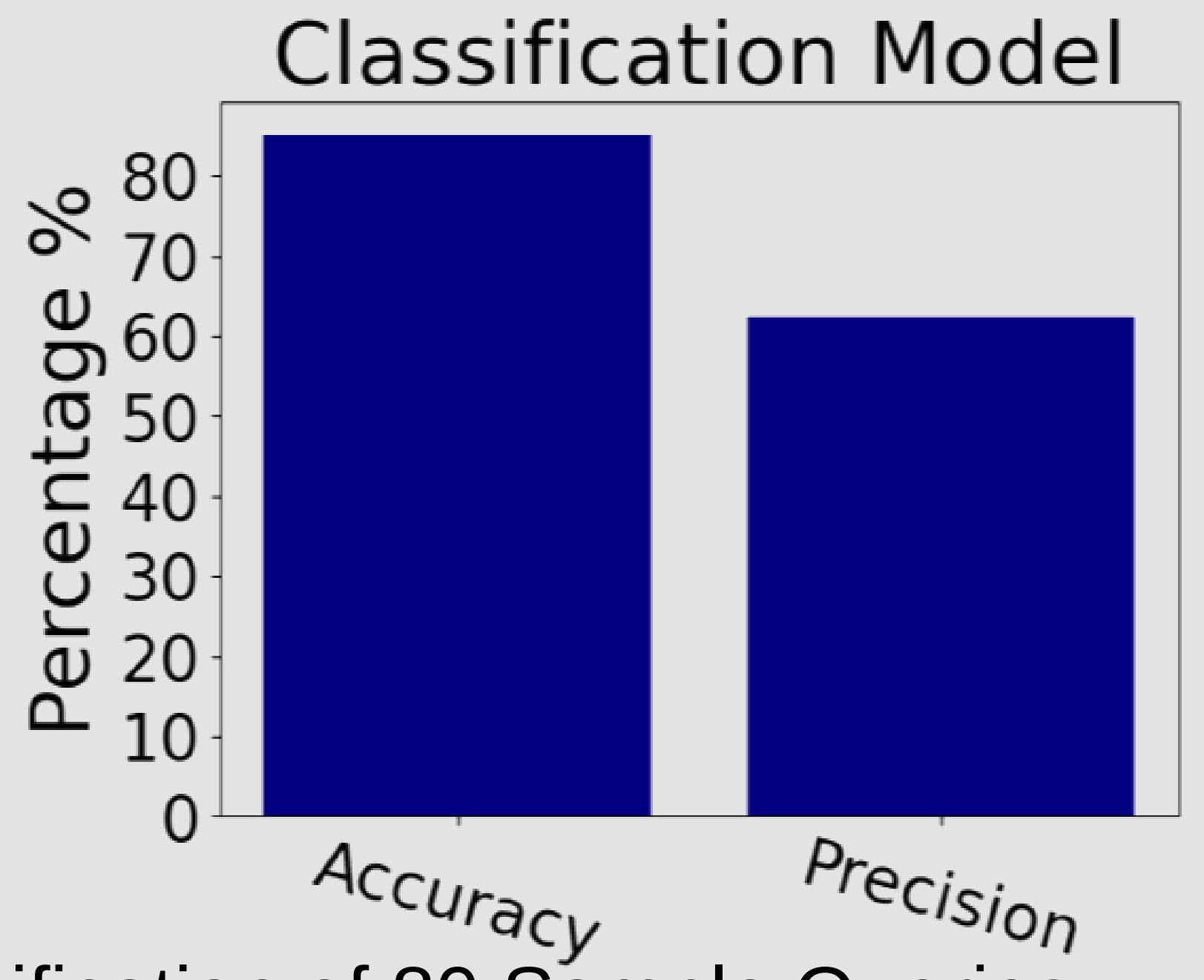



Query
1. Classification Process
Out-of-Scope Message
2. Rewrite Query
3. Generate Python Code using uclid5_api
4. Execute Python Code to UCLID5 Code
UCLID5 Code

Techniques

1. Few-shot used to classify if queries are suitable for the pipeline
2. Few-shot used in order to add information about the uclid5_api to the original query
3. Information about UCLID5_API and prompting the model to 'act' as an expert

Results: Classification



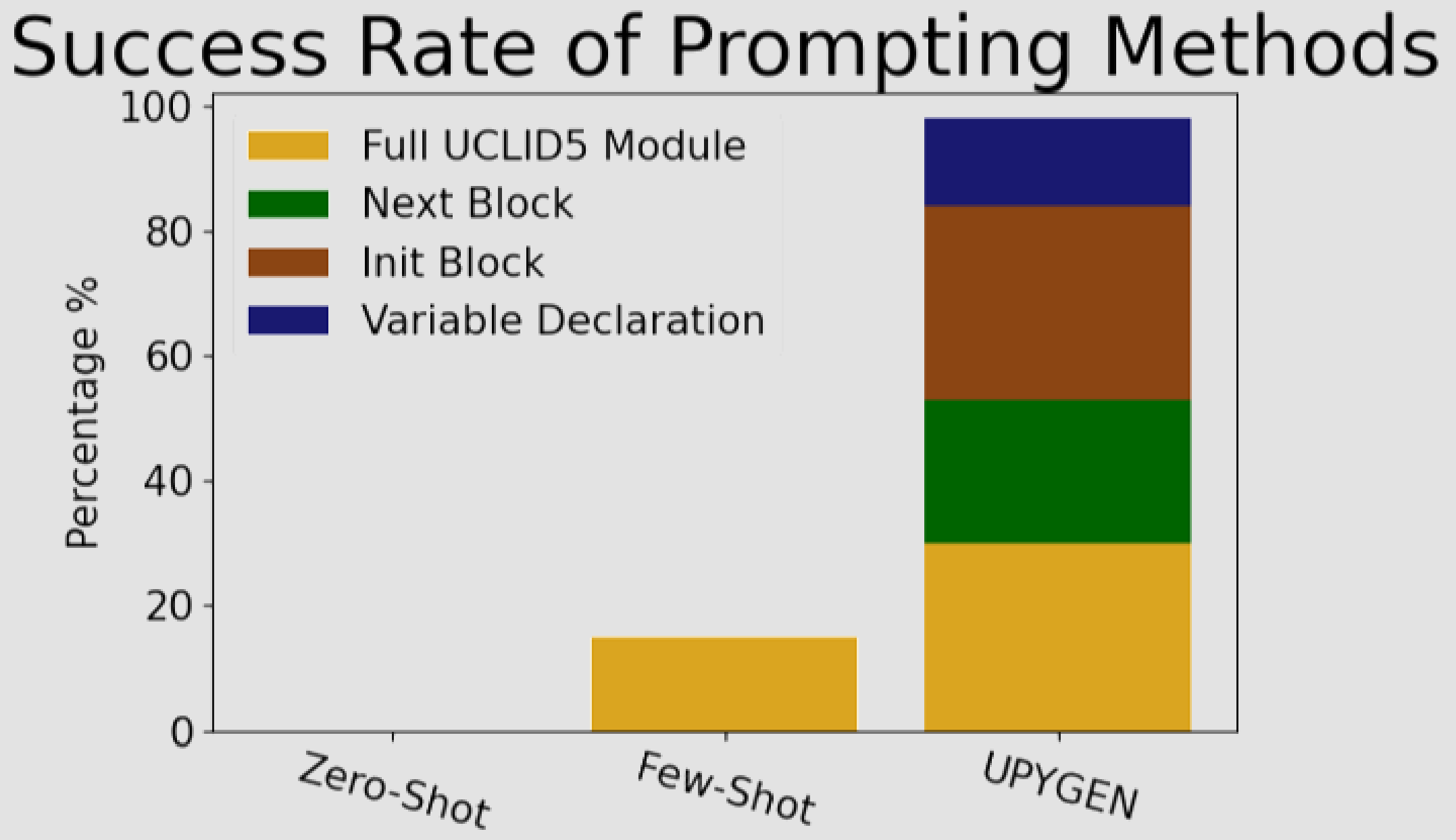
Classification of 80 Sample Queries

- 20 queries were valid
- 60 queries were invalid (20 adversarial)

Takeaways

- Model accepts most valid queries
- Model rejects most invalid queries
- Model struggled with adversarial queries, and allowing invalid queries to pass

Results: Syntax Analysis



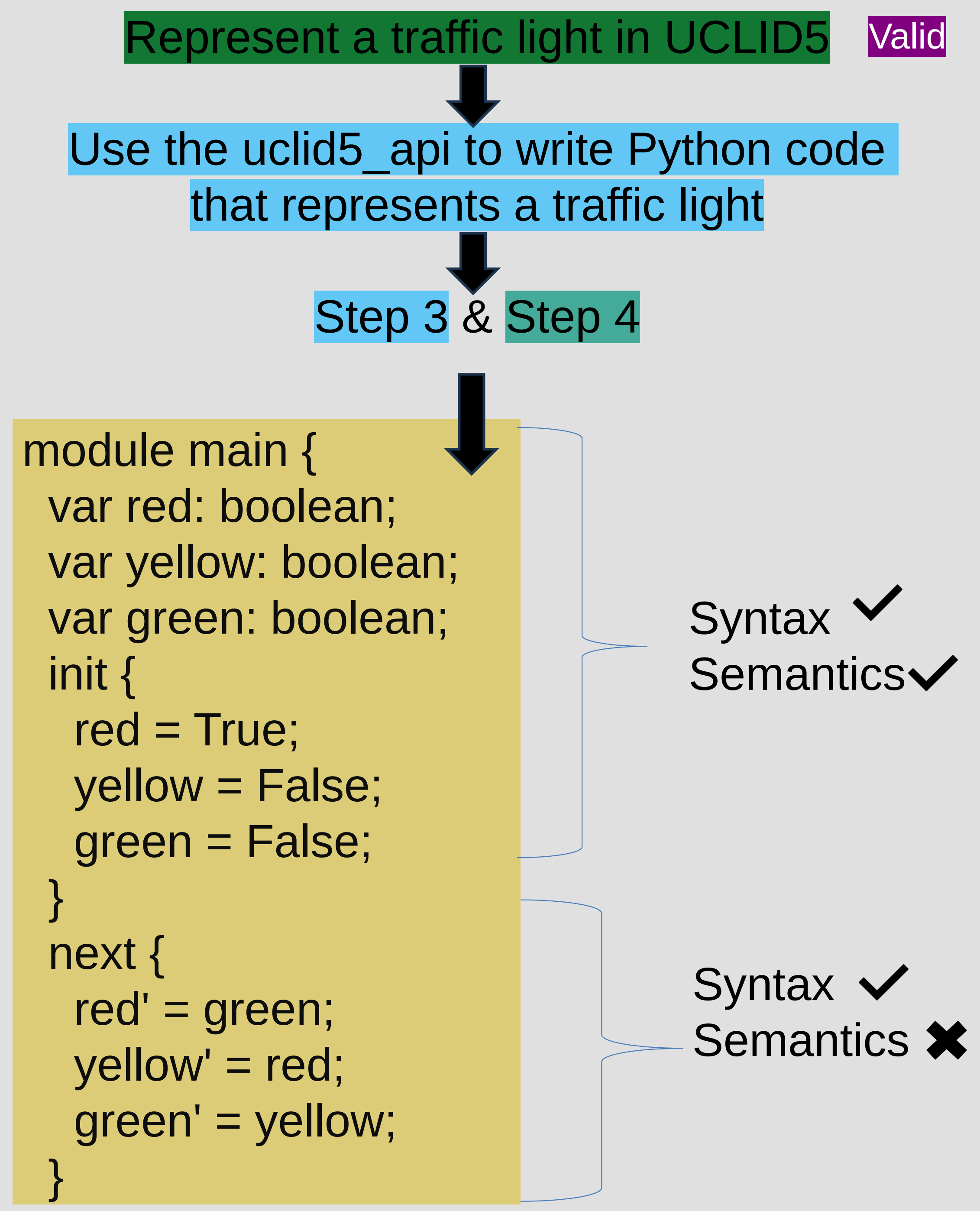
Five trials of 20 Valid Sample Queries

- Success- Syntactically correct UCLID5 code

Takeaways

- Zero shot was not successful for any trials
- UPYGEN outperforms few-shot by over 6x
- We can improve creating full modules

Results – Case Study



Represent a traffic light in UCLID5

Use the uclid5_api to write Python code that represents a traffic light

```
module main {
  var red: boolean;
  var yellow: boolean;
  var green: boolean;
  init {
    red = True;
    yellow = False;
    green = False;
  }
  next {
    red' = green;
    yellow' = red;
    green' = yellow;
  }
```

Conclusion & Future Work

- Improved LLMs ability in producing syntactically correct UCLID5 modules
- The prevalence LLM hallucinations exceeded initial expectations
- Semantic Struggles remain

Future Work

- Improve consistency of full module generation and semantic errors
- Give LLM multiple chances with loops

Acknowledgements

This work was funded by the Hopper Dean Foundation. I would like to thank my mentor, Federico Mora, for his guidance and support throughout this project. Additionally, I would also like to thank Sanjit A. Seshia for providing me with the opportunity to be supervised under his lab.

References

[1] J. White et al., "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT." arXiv, Feb. 21, 2023. Accessed: Jul. 07, 2023. [Online]. Available: http://arxiv.org/abs/2302.11382.
[2] Z. Zhang, A. Zhang, M. Li, and A. Smola, "Automatic Chain of Thought Prompting in Large Language Models." arXiv, Oct. 07, 2022. doi: 10.48550/arXiv.2210.03493.