AI x Smart Contract：

靜態分析工具做不到的，交給 Prompt Engineering！

Tim Ou

DEFIHACKLABS

Let's make web3 more secure!

DEFIHACKLABS
Let's make web3 more secure!

Tim Ou

- Senior @NCKU CSIE
- Web3 Engineering Intern @OneSavie Lab
- Member @DeFiHackLabs
- Focus on blockchain & AI
- X @T1m0uBruh

Agenda

DEFIHACKLABS
Let's make web3 more secure!

- Background - From Static Analysis to AI-based Auditing
- Methodology - Prompting LLMs to Think Like An Auditor
- Conclusion - Future Work & Takeaway

DEFIHACKLABS
Let's make web3 more secure!

Bastet: Let’s Build a Safer Web3 Together!

- Bastet is a dataset of DeFi smart contract vulnerabilities, paired with an AI-driven detection process.
- We're building tools to enhance vulnerability detection and optimize security lifecycle management.

Background

Importance of Smart Contract Security

Rapid Market Growth

$232M → $1.98B by 2034

(Statifacts, 2024)

Security Breaches in 2024

$2.01B lost in 410 attacks

(Slowmist, 2024)

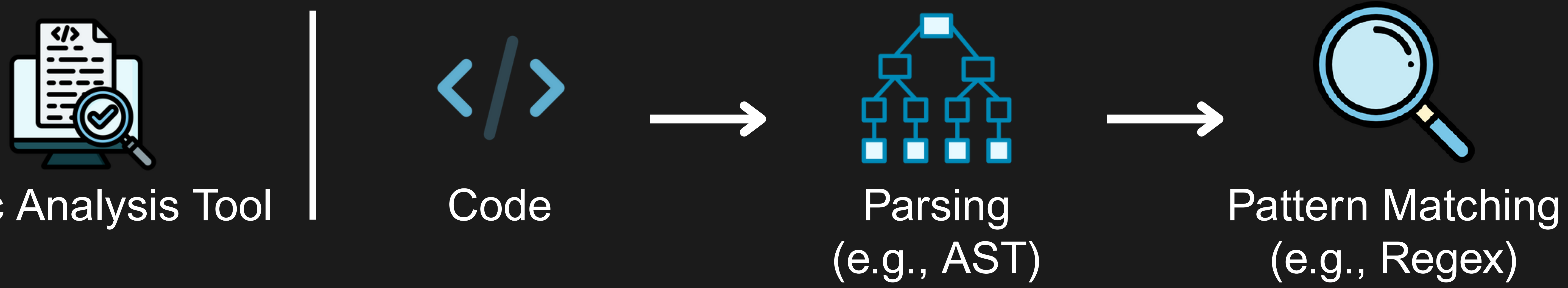
Security as a Core Requirement.

The expanding DeFi landscape is driving stronger demand for robust smart contract protection.

DEFIHACKLABS
Let's make web3 more secure!

Limitations of Static Analysis Tools

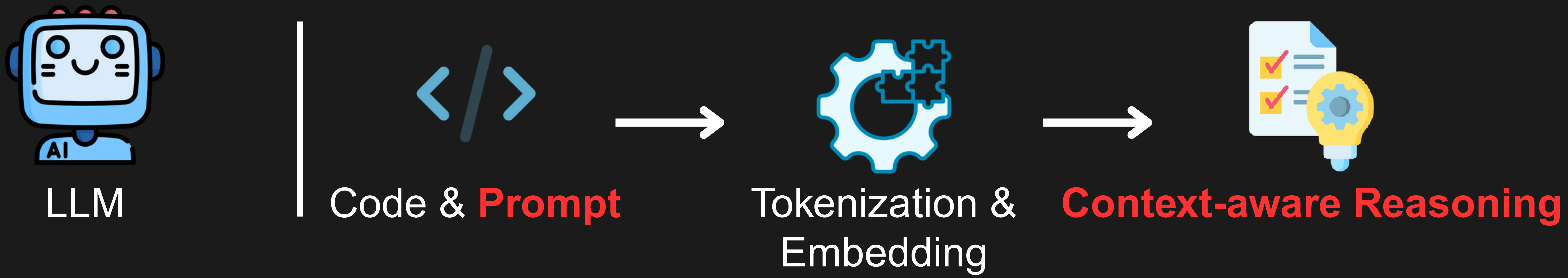
- **Rules** - AST or Regex.
- **Lack of Customization** - Not perfectly align with the specific needs.
- **High False Positives** - Flags harmless code with limited context.
- **Limited Scope** - Fails to capture full contract logic beyond single functions.



DEFIHACKLABS
Let's make web3 more secure!

Advantages of LLM

- Rules - Prompt
- Easy Adjustment - Easily adapts to new code and patterns.
- Cross-language - Works across languages, no tool lock-in.
- Contextual & Semantic Awareness - LLMs capture semantic patterns beyond the reach of static analysis tools.



How Do LLMs "Understand" Code?

Pretrained on
Massive Code Datasets

Tokenization &
Embedding to Capture the
Meaning & relationships

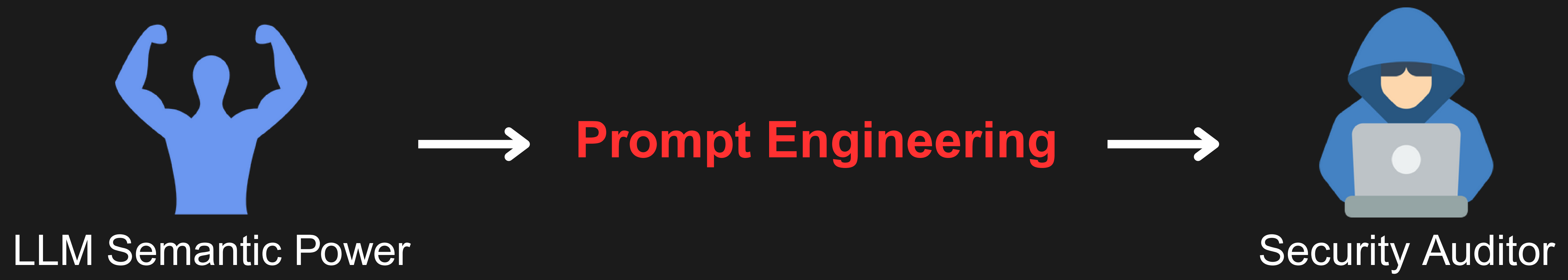
Self-Attention
Enables Context
Awareness

LLMs don't truly understand code; instead, they generate outputs by statistically predicting the most probable next token based on their training data.

LLMs “Understand”. But Can They Audit?

DEFIHACKLABS
Let's make web3 more secure!

- LLMs are generalists - Without task-specific guidance, they may hallucinate or misinterpret code logic.
- **Prompt engineering bridges the gap between understanding and auditing.**



Why Prompt Engineering Over Fine-tuning

DEFIHACKLABS
Let's make web3 more secure!

	Prompt Engineering	Fine-tuning
Cost	**Low** - no training needed	High - training required
Experiment	**Fast** - prompts can be tested and iterated quickly	Slow - testing requires time-consuming retraining
Flexibility	**High** - fine-grained control with prompt adjustments	Low - retraining needed for new tasks or changes

Methodology

DEFIHACKLABS
Let's make web3 more secure!

DEFIHACKLABS
Let's make web3 more secure!

Vulnerability: `slippage_min_amount`

- Swap functions must allow users to specify a minimum amount they want to receive, and this variable shouldn't be hardcoded or unused.
- If not followed, users are vulnerable to sandwich attacks, leading to poor trade execution and fewer tokens received.

Attacker detects a pending txn (X → Y) → Attacker frontruns to buy Y → Victim txn executes, buying Y at higher → Attacker backruns to sell Y, making profit

`slippage_min_amount` Example 1

DEFIHACKLABS

Let's make web3 more secure!

```
function addLiquidity(
    IERC20 tokenA,
    IERC20 tokenB,
    uint256 amountADesired,
    uint256 amountBDesired,
    uint256, // amountAMin = unused
    uint256, // amountBMin = unused
    address to,
    uint256 deadline
) external override returns (uint256 liquidity) {
    return
        addLiquidity(
            tokenA,
            tokenB,
            amountADesired,
            amountBDesired,
            to,
            deadline
        );
}
```

<u>did not use `amountAMin` & `amountBMin`</u>

`slippage_min_amount` Example 2

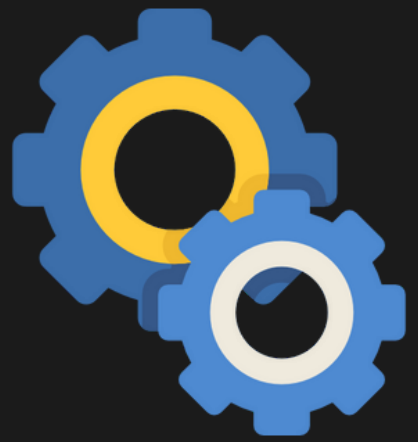
DEFIHACKLABS
Let's make web3 more secure!

```
if (amountOut > currentBalance) {
    ICurveSwap(CVX_CRV_CRV_CURVE_POOL).exchange(
        _CRV_INDEX,
        _CVX_CRV_INDEX,
        currentBalance,
        0
    );
```

`min_amount_out` set to 0 (hardcoded)

DEFIHACKLABS
Let's make web3 more secure!

Why static analysis tools can't detect it well



Various Functions Related to Swap

Naming Variations of Slippage Parameter Across Protocols

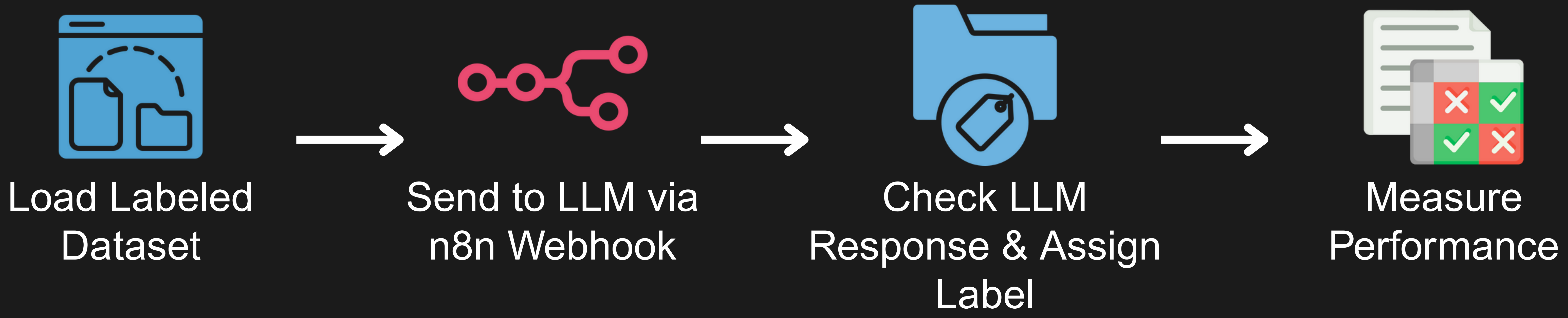
Cross-language Detection Limitation

Dataset for `slippage_min_amount`

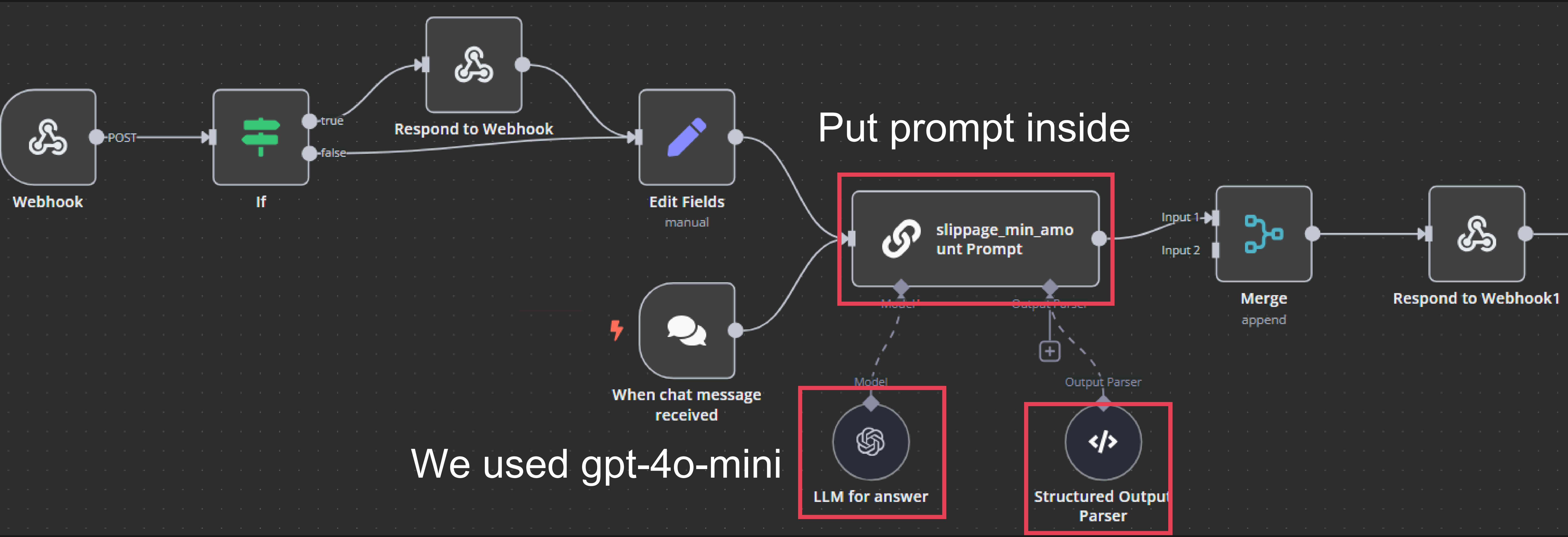
Positive Data (vuln) = code4rena + On-chain Contracts = 29

Negative Data (no vuln) = Secure Contracts (e.g., OZ, top protocols) = 29

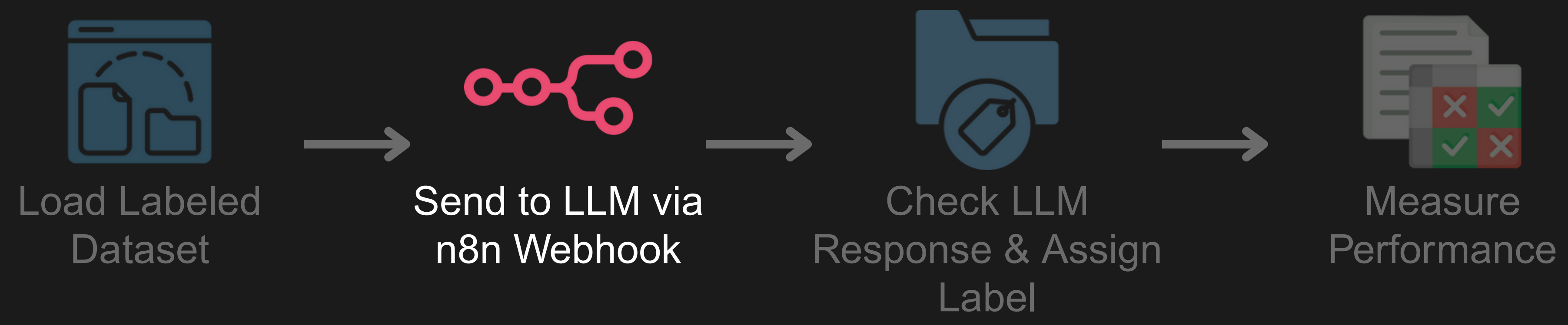
Evaluation Workflow



n8n Workflow



Fix LLM’s output



DEFIHACKLABS
Let's make web3 more secure!

Output Parser

- To convert LLM's output into structured and usable data, we designed a JSON output parser.
- Each JSON object corresponds to a single detected vulnerability in the LLM's final array output.

```
{
  "summary": "...",
  "severity": ["high", "medium", "low"],
  "vulnerability_details": {
    "function_name": "...",
    "description": "..."
  },
  "code_snippet": ["..."],
  "recommendation": "..."
}
```

Evaluation Method & Output

- Evaluation Method - If output = `[]`, predict 0 (safe), else 1 (vulnerable).
- Metrics - Confusion Matrix.

	Actually Vulnerable	Actually Safe
Predicted Vulnerable	True Positive	False Positive
Predicted Safe	False Negative	True Negative

```
+-----------------+---------+
| Metric          |   Value |
+=================+=========+
| True Positive   |      28 |
+-----------------+---------+
| True Negative   |       0 |
+-----------------+---------+
| False Positive  |      29 |
+-----------------+---------+
| False Negative  |       1 |
+-----------------+---------+
accuracy: 0.4827586206896552
precision: 0.49122807017543857
recall: 0.9655172413793104
f1: 0.6511627906976745
```

Evaluation Method & Output

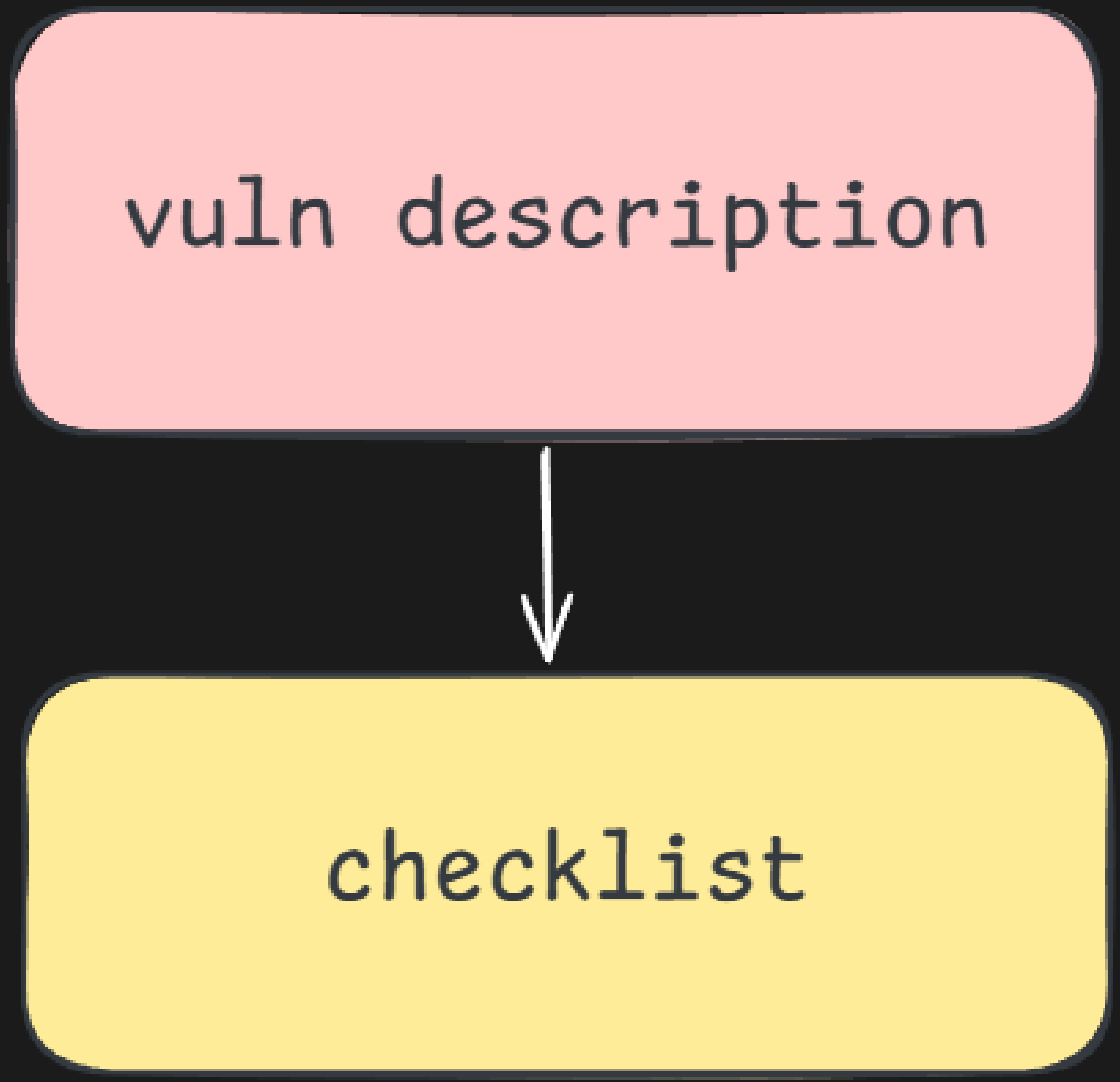
- Performance Indicators
  - Accuracy - Overall correctness.
  - Precision - Correctness of positive predictions.
  - Recall - Coverage of actual positives.
  - F1 Score - Average of precision and recall.

Metric	Value
True Positive	28
True Negative	0
False Positive	29
False Negative	1

```
accuracy: 0.4827586206896552
precision: 0.49122807017543857
recall: 0.9655172413793104
f1: 0.6511627906976745
```

Prompt Ver. 1 - Listing

- Outline common patterns that cause this vulnerability.
- checklist:
  - should not be set to 0.
  - should not be unused.
  - should not be hardcoded.



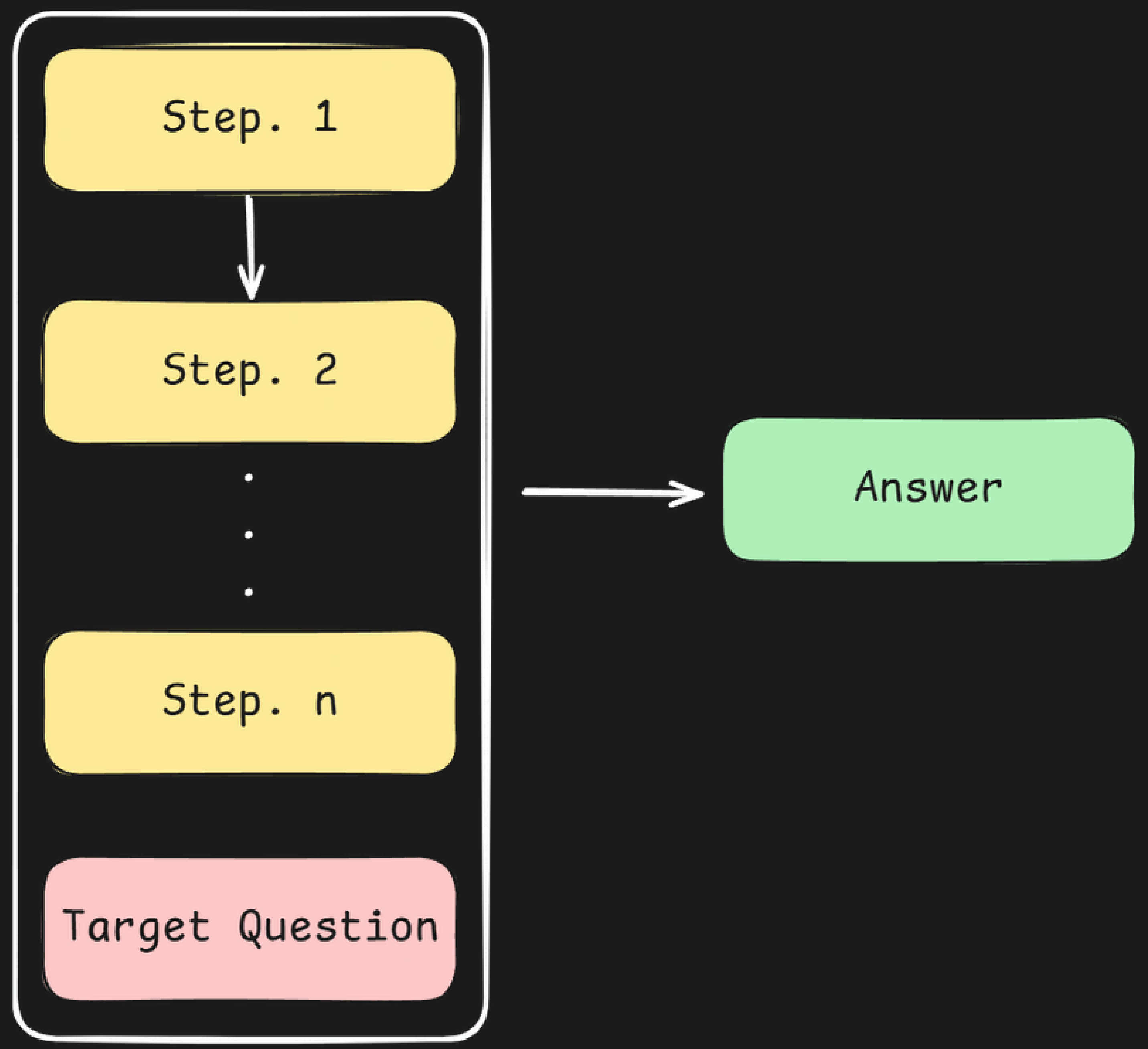
Observation

- **High Recall, Poor Precision** - Flags many safe functions by mistake.
- **Overgeneralization** - Matches risky-looking patterns without checking if they're actually unsafe.
- **Lack of Reasoning** - Matches patterns without applying deeper logic or understanding the code's intent.

```
+----------------+----------+
| Metric         |    Value |
+================+==========+
| True Positive  |       28 |
+----------------+----------+
| True Negative  |        0 |
+----------------+----------+
| False Positive |       29 |
+----------------+----------+
| False Negative |        1 |
+----------------+----------+
accuracy: 0.4827586206896552
precision: 0.49122807017543857
recall: 0.9655172413793104
f1: 0.6511627906976745
```

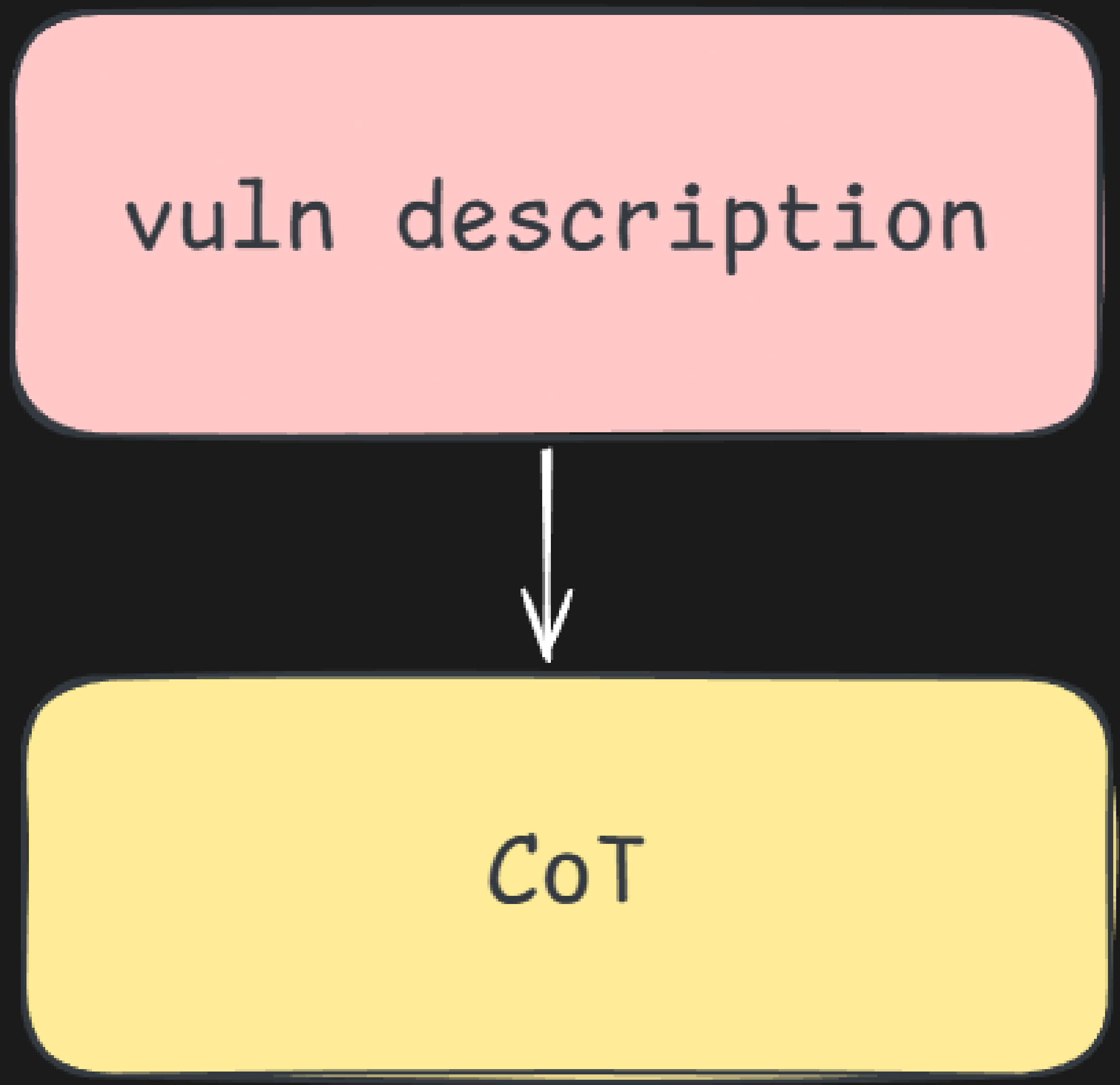
Prompt Engineering Technique - CoT

- Chain-of-Thought
- Breaks down reasoning into explicit, step-by-step explanations.
- Mimics human(auditor) thinking by laying out how each part leads to the next.
- Helps identify where reasoning might go wrong, making the model's logic more transparent and interpretable.



Prompt Ver. 2 - CoT

1. **Identification** - Find out function involving swap-related actions.
2. **Extraction** - Spot parameters related to slippage protection in these functions.
3. **Validation** - Assess use of slippage parameters. (unused? hardcoded?)
4. **Confirmation** - Verify actual vulnerability.



Observation

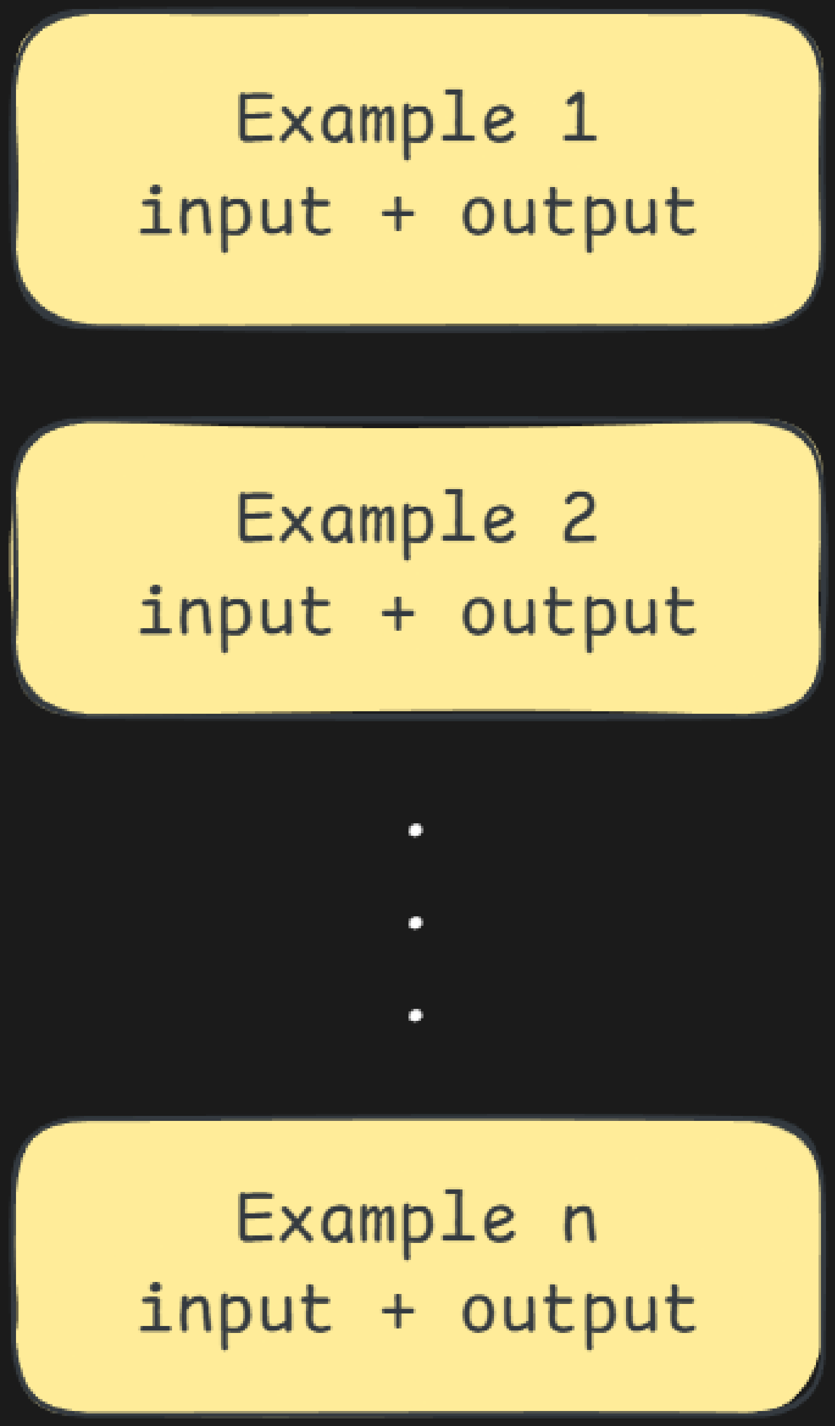
- Lower false positives, but not enough.
- More Explainable Output - Step-by-step thinking improves clarity and makes results easier to audit.
- Steps Not Followed - Steps are sometimes skipped, merged, or misinterpreted.

Metric	Value
True Positive	27
True Negative	9
False Positive	20
False Negative	2

```
accuracy: 0.6206896551724138
precision: 0.574468085106383
recall: 0.9310344827586207
f1: 0.7105263157894737
```

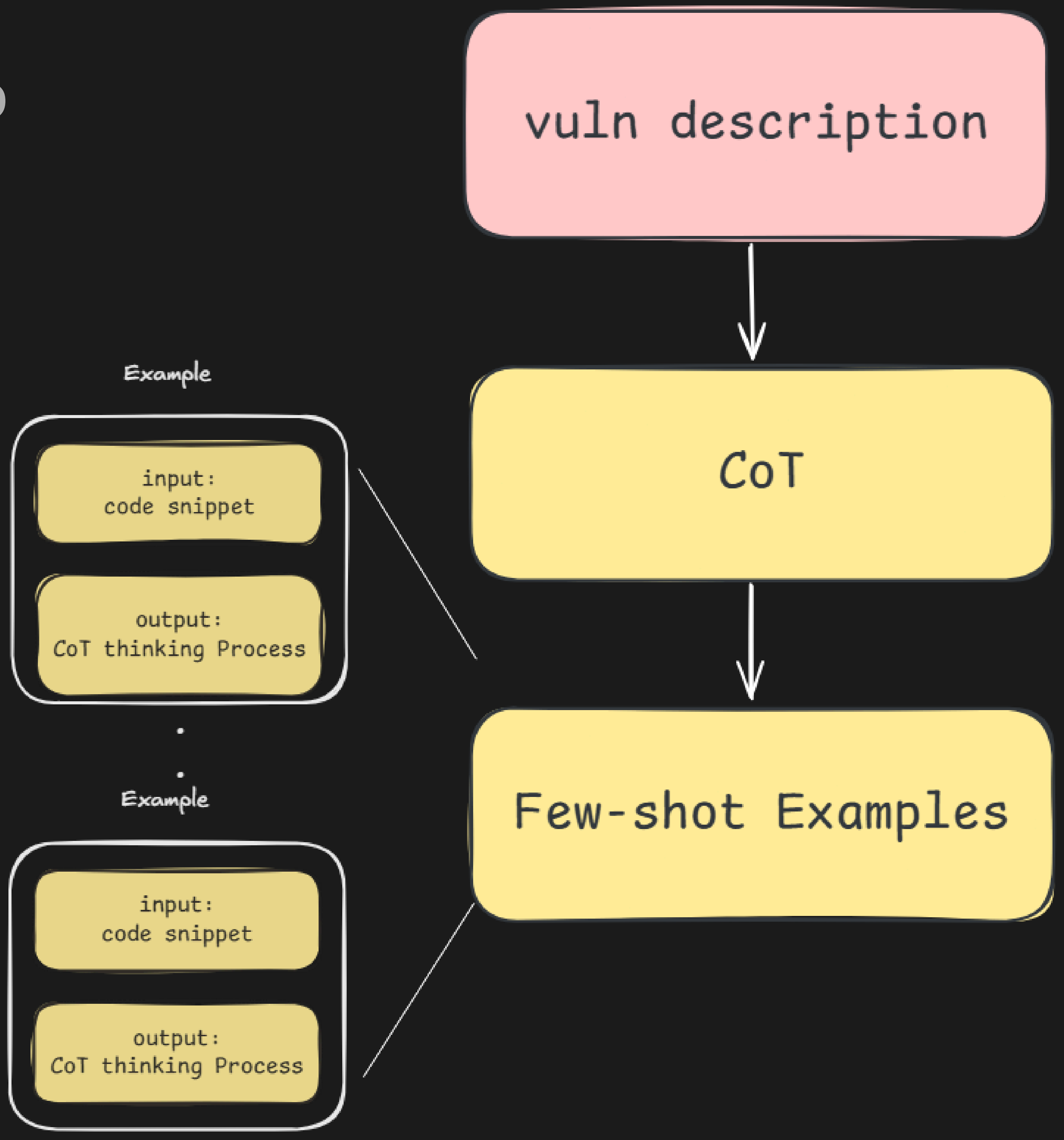
Prompt Engineering Technique - Few-shot

- Giving the model a few **examples** to help it learn how to perform a task.
- Each example shows **input + desired output**, helping the model generalize to new cases.
- Typical number of examples: 2 ~ 5.



Prompt Ver. 3 - CoT + Few-Shot

- Besides CoT, provide several examples to illustrate how it works in practice.
- Example input - Code snippet.
- Example output - CoT thinking process.
- 3 positive + 2 negative examples.



Observation

- Significantly Lower False Positives - With a slight rise in false negatives. A worthwhile tradeoff in most audit scenarios.
- Strong Alignment with Desired Behavior - Examples enhance the CoT, guiding LLM to reason consistently and stay on track.

Metric	Value
True Positive	24
True Negative	22
False Positive	7
False Negative	5

```
accuracy: 0.7931034482758621
precision: 0.7741935483870968
recall: 0.8275862068965517
f1: 0.8
```

Other Vulnerabilities Explored

- Reentrancy
- Liquidation - No Incentive to Liquidate Small Positions
- Liquidation - DoS
- Liquidation - Accounting Error

Conclusion

Future Works

DEFIHACKLABS
Let's make web3 more secure!

- Establish Clear Evaluation Standards - Develop more rigorous, fine-grained methods to evaluate prompt performance.
- Expand Dataset Scale - ~4,400 cases labeled with data mutation; more contributors needed for labeling.
- Refine Prompt Engineering - Test diverse prompt techniques and sharing results with the community.
- Explore More Vulnerabilities - Leverage Bastet across a wider range of attack vectors, and promote knowledge sharing with the community.

Takeaway

DEFIHACKLABS
Let's make web3 more secure!

- Static analysis tools are limited by rule-based approaches.
- LLMs offer flexible, cross-language, and semantically aware analysis that better handles complex code.
- Prompt engineering guides LLMs to perform context-aware and logic-aware auditing.
- Prompt engineering requires iterative experimentation and refinement.
- Rapid AI advancement makes LLM-based auditing increasingly powerful and easier to apply.

Join Us - Shape the DeFi Security with Bastet

- Bastet is a dataset of DeFi smart contract vulnerabilities, paired with an AI-driven detection process.
- We're building tools to enhance vulnerability detection and optimize security lifecycle management.
- Whether you're from blockchain or AI, your contribution can shape the future of DeFi security. We're building a safer Web3 and need your expertise to get there.

DEFIHACKLABS

Let's make web3 more secure!