Analytics Vidhya

Real-world Applications of the Reflection Pattern

Created by:
Eleni Verteouri
Gen AI Tech Lead @ UBS

Created & Narrated by:
Dipanjan Sarkar
Head of Community & Principal AI Scientist @ Analytics Vidhya
Google Developer Expert - ML & Cloud Champion Innovator
Published Author

Limitations of Static and Non-Iterative Systems

Inflexibility:

- Outputs do not adapt to new data, feedback, or changing requirements.
- **Example:** A static summary tool unable to tailor content for different audiences.

Missed Opportunities for Improvement:

- Quality remains limited to initial assumptions or inputs.
- **Example:** Non-iterative AI chatbots that fail to learn from user feedback.

Limitations of Static and Non-Iterative Systems

Error Propagation:

- Errors in initial outputs persist without correction.
- **Example:** Static document generation with formatting or grammatical mistakes.

Reduced Accuracy Over Time:

- Systems degrade in relevance or utility as contexts evolve.
- **Example:** An AI tool generating reports with outdated templates and methods.

Reflection Pattern for Research Paper Summaries

The reflection pattern helps **simplify** academic or educational content for students or researchers.

How does it work?

Generation Phase:
AI produces an initial summary of a research paper.

Critique Phase:
The system evaluates the summary for clarity and completeness.

Iteration Phase:
Refines the summary to ensure it highlights points effectively.

Example: *Summarizing research papers for high school students, adopting complexity to their learning level.*

Analytics Vidhya

Reflection Pattern for Text Generation

The reflection pattern helps **produce** formal or creative content such as articles, essays, or instructional material.

How does it work?

Generation Phase:
AI drafts the text based on user prompts.

Critique Phase:
Evaluates the draft for coherence, grammar, and relevance to the topic.

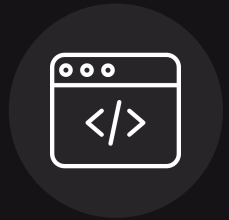
Iteration Phase:
Refines the draft to better match user expectations or style guidelines.

Example: *AI generating blog posts or explanatory content for non-technical audiences.*

Reflection Pattern for Code Debugging & Improvement

The reflection pattern helps **identify and resolve** errors in non-critical code or workflows.

How does it work?



Generation Phase:
AI proposes a solution to a coding problem.

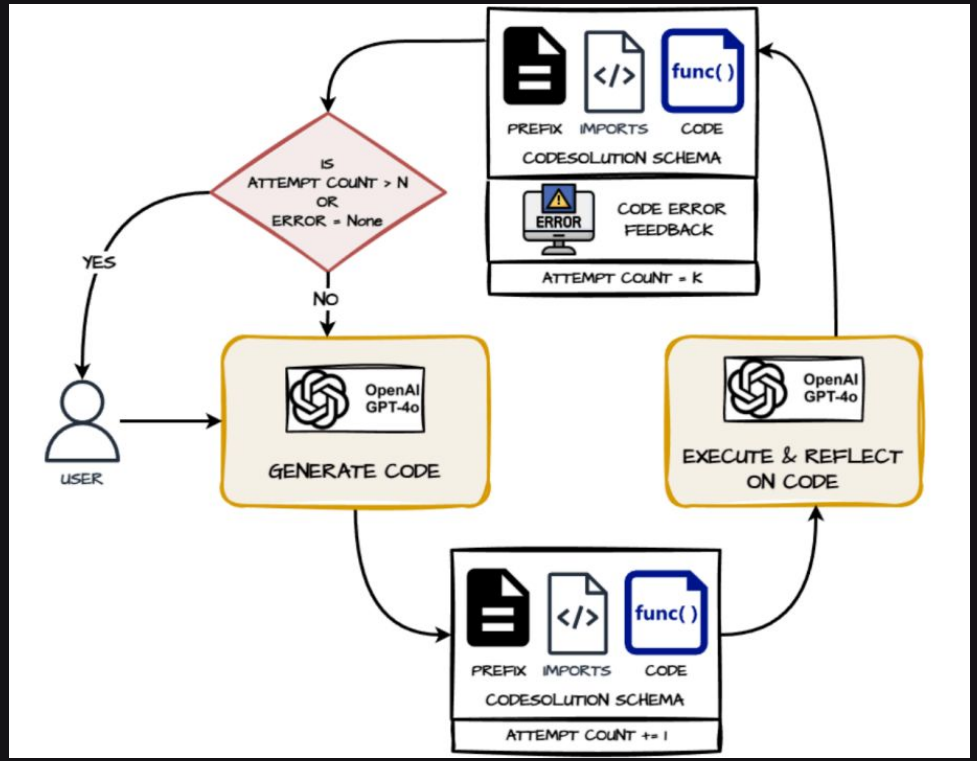
Critique Phase:
Analyze the proposed solution for correctness and efficiency.

Iteration Phase:
Refines the solution, incorporating feedback.

Example: *Debugging a Python script for organizing data files.*

Analytics Vidhya

Reflection Pattern for Code Debugging & Improvement

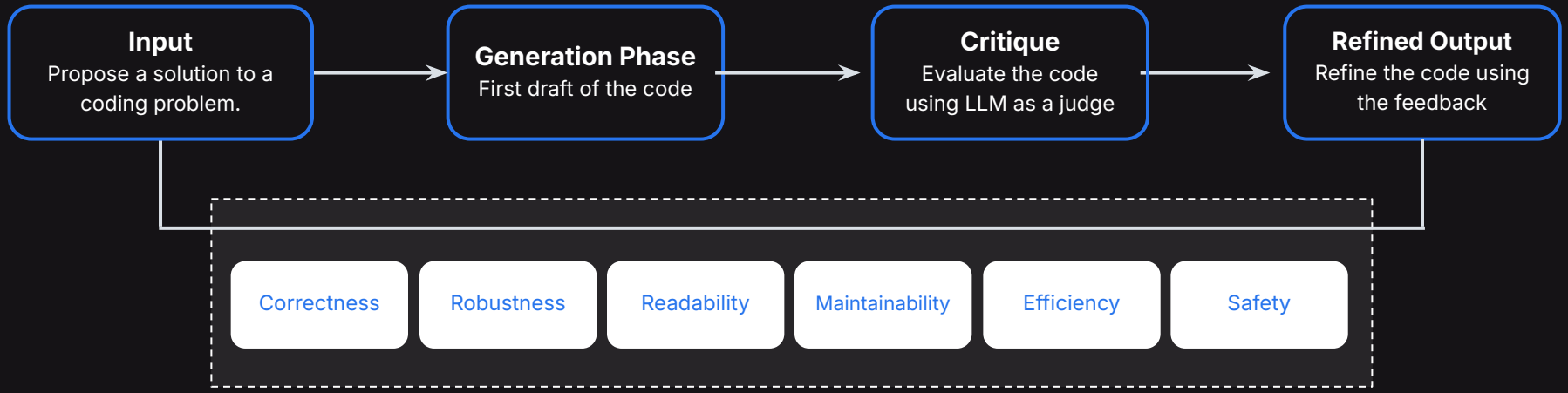


- **Analyze** user query to identify the problem.
- **Generate** code with solution description and required imports.
- **Validate** by checking imports and running the code.
- **Record** feedback on output and errors.
- **Correct** the code based on feedback.
- **Repeat** until successful or for N iterations.

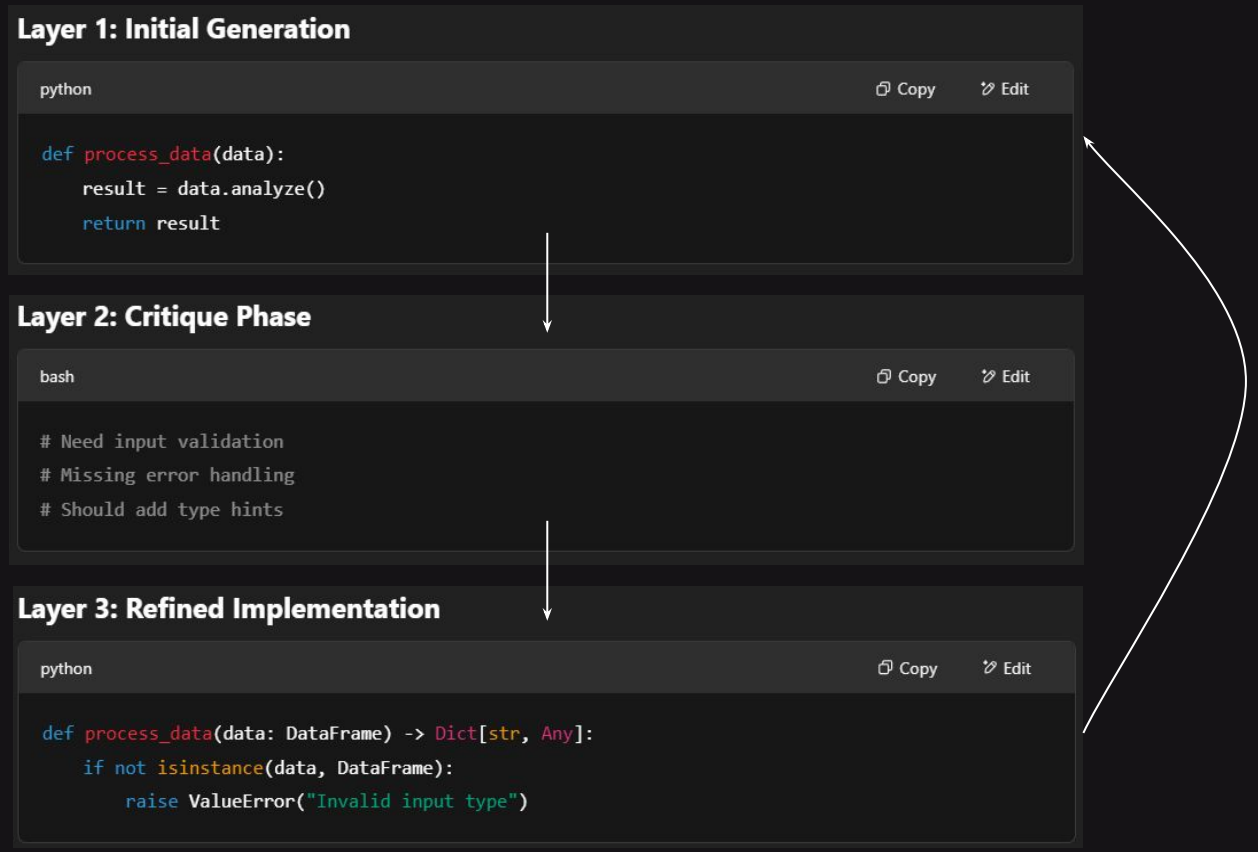
Analytics Vidhya

Reflection Pattern for Code Debugging & Improvement

The reflection pattern has taken over the creative world by storm. Take a look at the a **Code Generation Improvement Workflow** of how it achieves this.



Reflection Pattern for Code Debugging & Improvement



Analytics Vidhya

Reflection Pattern for Code Debugging & Improvement

1. Generation Phase

Goal: Propose an initial solution to a coding problem.

python Copy Edit

```python
def organize_files(path):
    files = os.listdir(path)
    return files
```

- **What it does:** Lists files in a given directory.
- **What's missing:** No input checks, error handling, or sorting.

Reflection Pattern for Code Debugging & Improvement

2. Critique Phase (LLM-as-Judge): Use an LLM to evaluate the code against key quality dimensions.

LLM Evaluation Criteria:

Dimension	Evaluation Focus
✅ Correctness	Does the function fulfill its intended purpose correctly?
⚠️ Robustness	Are edge cases (e.g., invalid path, missing folder) handled?
📄 Readability	Is the code clean and understandable (e.g., names, layout)?
🛠️ Maintainability	Are there clear type annotations and documentation?
📊 Efficiency	Are unnecessary operations avoided?
🔒 Safety	Are error cases and input types validated to prevent runtime failures?

Reflection Pattern for Code Debugging & Improvement

This automated critique can be prompted using a chain like:

python Copy Edit

```
critique_prompt = f"""
Review the following function and provide feedback on the following:

- Correctness
- Error handling
- Input validation
- Readability
- Completeness

Code:
{code_snippet}

Respond with concrete suggestions.
"""
```

Reflection Pattern for Code Debugging & Improvement

LLM Feedback Output Example:

```
- Missing type annotations for function arguments and return type.
- No input validation; 'path' can be None or non-string.
- Missing exception handling for FileNotFoundError and PermissionError.
- Returned file list is unsorted.
- No docstring explaining function behavior.
```

Reflection Pattern for Code Debugging & Improvement

3. Iteration Phase: Refine the code using feedback from the LLM Judge.

python Copy Edit

```python
from typing import List
import os

def organize_files(path: str) -> List[str]:
    """
    Organizes and returns a sorted list of files from the given directory path.
    Raises a ValueError if the path is invalid or not a string.
    """
    if not isinstance(path, str):
        raise ValueError("Expected path to be a string.")

    try:
        files = os.listdir(path)
    except FileNotFoundError:
        raise ValueError("Directory not found.")
    except PermissionError:
        raise ValueError("Permission denied to access the directory.")

    return sorted(files)
```

Analytics Vidhya

Challenges in Applying the Reflection Pattern

Parameters	Challenge	Impact	Solution
Effective Critiques	Poor or rigid feedback leads to suboptimal outputs	• Inconsistent evaluation • Missing critical aspects	• Powerful LLM-as-a-Judge • Clear Critique Criteria
Over-Optimization	Excessive refinement Diminishing returns	• Performance cost • Loss of generality	• Define clear refinement scope based on critiques • Define stopping conditions
User Input Balance	Balancing automation with user feedback	• Missed user insights • Feedback overload	• Hybrid approach • Clear user guidelines

Thanks!

Analytics Vidhya