Generative Caching for Structurally Similar Prompts and Responses

Sarthak Chakraborty[1], **Suman Nath**[2], **Xuchao Zhang**[2], **Chetan Bansal**[2], **Indranil Gupta**[1]

[1] University of Illinois Urbana-Champaign, [2] Microsoft Research

NEURAL INFORMATION PROCESSING SYSTEMS

Paper

Our Lab

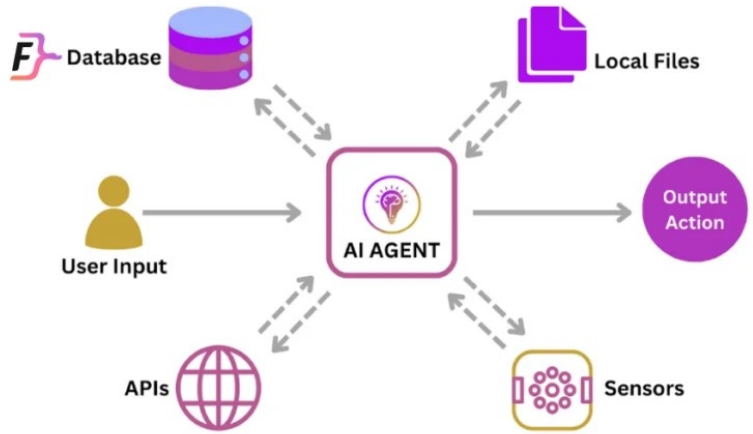
AI Agents for Repetitive Tasks/Workflows

- Autonomous Systems for **Decision Making** and Task Completion

- AI workflows are an organized sequence of tasks
  - Predetermined goal for each step
  - An LLM performs each step

- AI agents often solve **repetitive, workflow-driven tasks**
  - **SRE agents:** Alerts are recurring. Many alerts have a similar diagnosis guide
  - **Web-Shopping agents:** Search user instructions and add item to cart
  - **Web agents** for flight booking

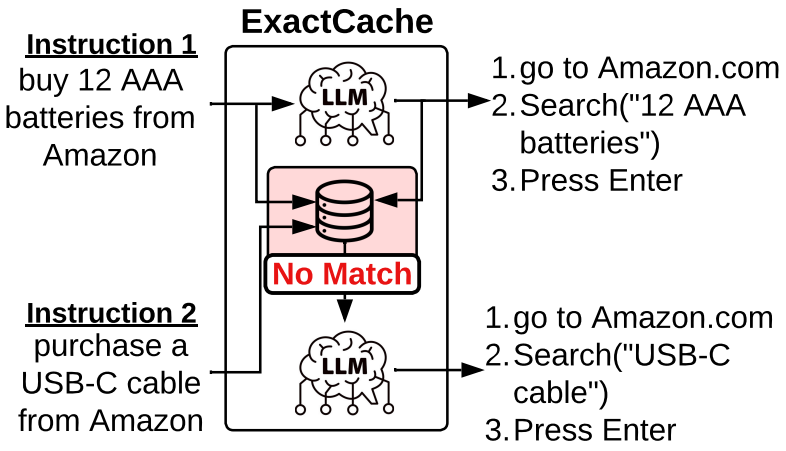
- Instead of repeatedly asking LLM at each step, **why not cache the action and reuse the cached action**?



AI Agent Architecture

Caching for AI Workflows

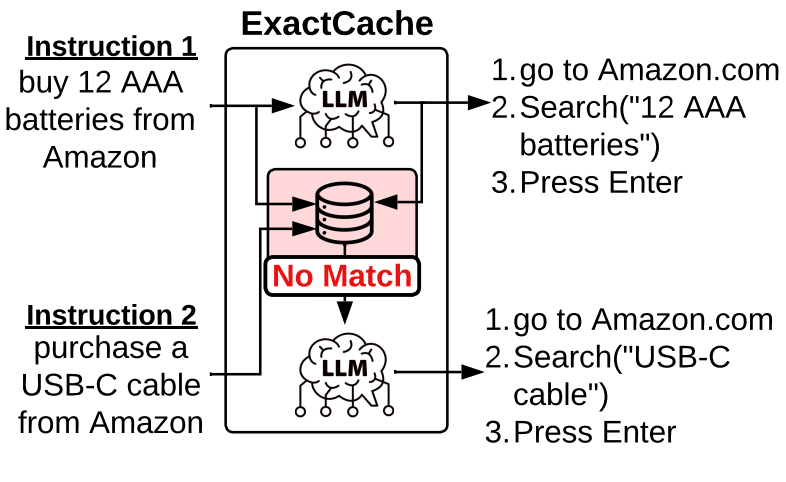
Exact Prompt Matching



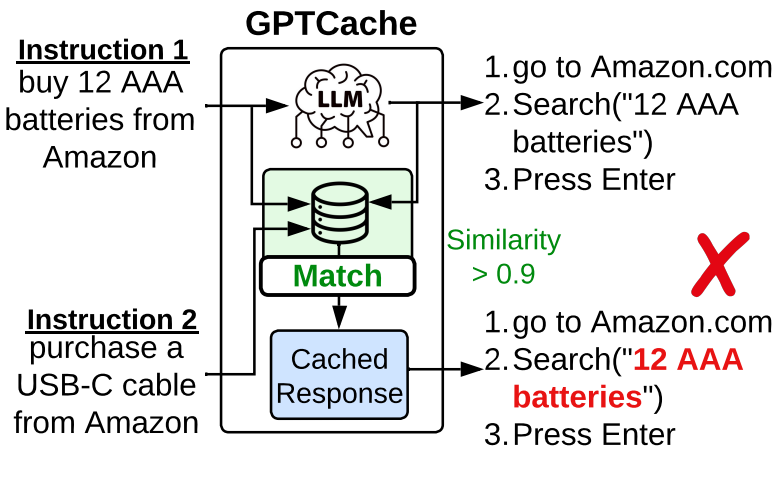
Exact Prompt Matching fails since prompts are different

Caching for AI Workflows

Exact Prompt Matching



Semantic Caching

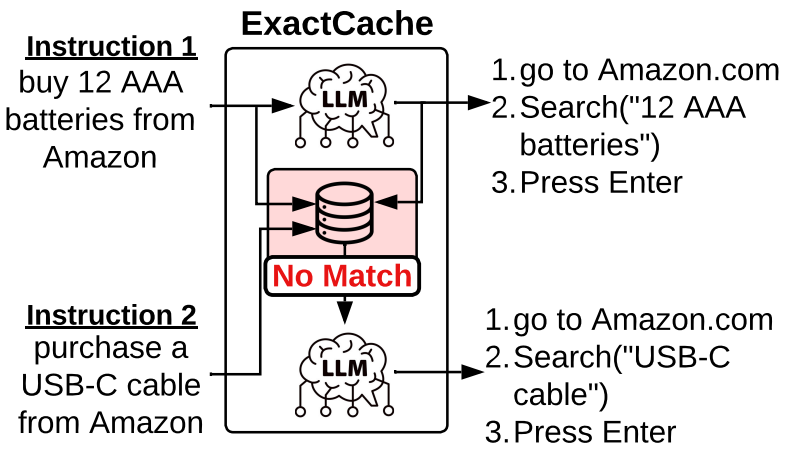


Exact Prompt Matching fails since prompts are different

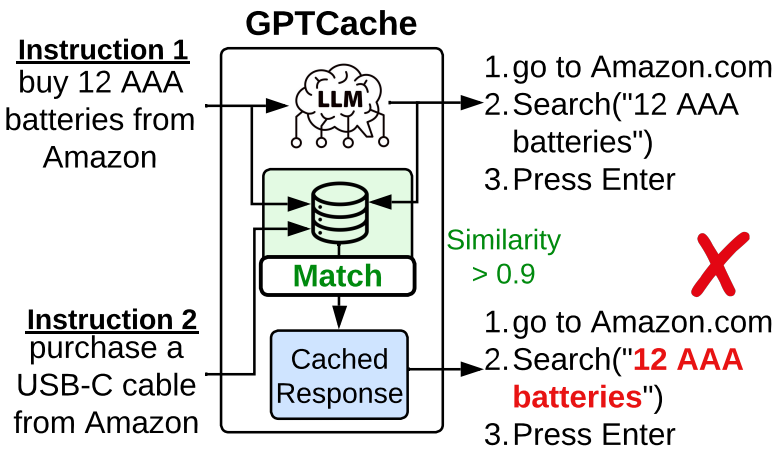
Semantic Caching reuses responses across prompt

Caching for AI Workflows

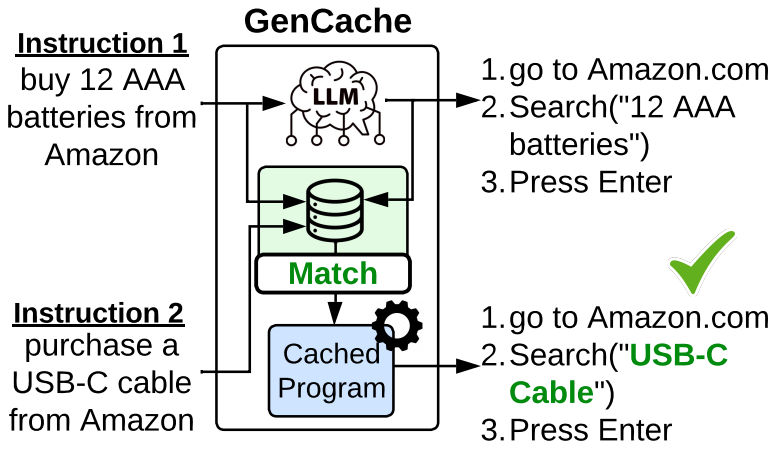
Exact Prompt Matching



Semantic Caching



Our Method (Generative Caching)



Exact Prompt Matching fails since prompts are different

Semantic Caching reuses responses across prompt

GenCache generates responses tailored to prompt

Structurally Similar Prompts

- **Structural Regularity:** Prompts follow a consistent format
  - Item to purchase always appears between the phrase “buy” and “from Amazon”
- **Controlled Variability:** Structure must remain same, but parameters (item names, alert monitors, etc.), synonyms, or optional words like “please” can vary
  - “*Please* buy {item} from Amazon” follows the structure *verb–item–website*
- **Predictable Response Patterns:** Responses generated follow a consistent format, even if they are not identical

Prompt: $\alpha(x) = a_1.\boldsymbol{x}.a_2$	**Response:** $\beta(x) = b_1.\boldsymbol{f(x)}.b_2$
Buy **12 AAA batteries** from Amazon	(1) go to Amazon.com; (2) Search(**12 AAA batteries**); (3) Press Enter
Purchase a **USB-C cable** from Amazon	(1) go to Amazon.com; (2) Search(**USB-C cable**); (3) Press Enter

Controlled Variability (synonyms)

Programmatic Way to Generate such Responses?
Cache the Program and execute locally

Overview of our Solution - GenCache

How to create Cache

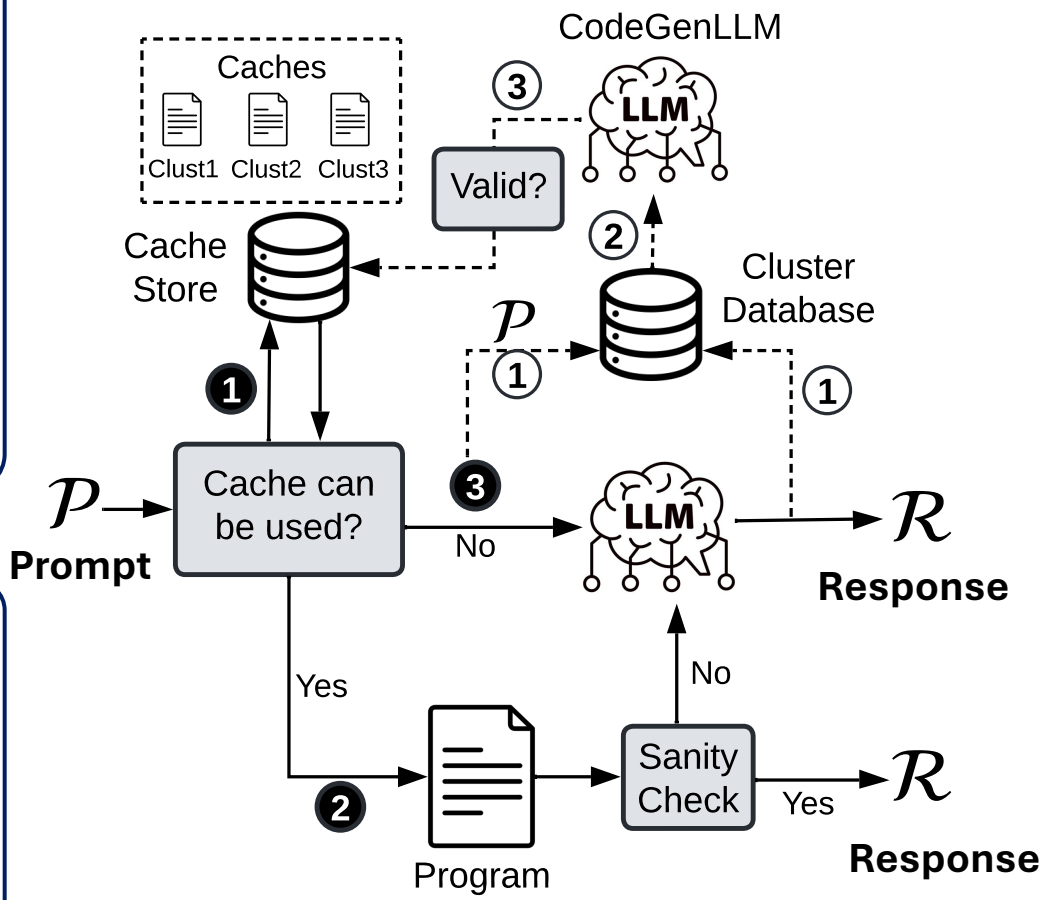
① **Cluster similar prompts and responses** and store them in a KV database

② **Infer a consistent pattern** for each cluster
- Pattern generates corresponding response given a prompt
- Write the pattern as an executable program

③ **Validate** the generated program before storing as cache

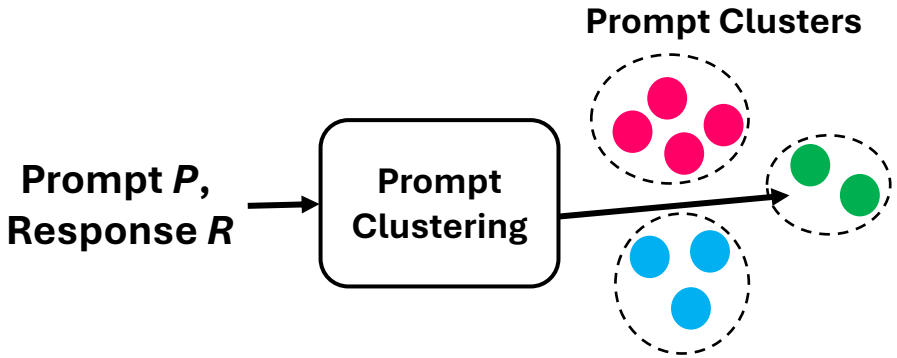
How to use Cache at Runtime

1. Check if prompt belongs to a cluster that has a cache
2. Use the cached program to generate a response
3. If cache miss, use LLM to generate response, and store the prompt-response paper in KV database



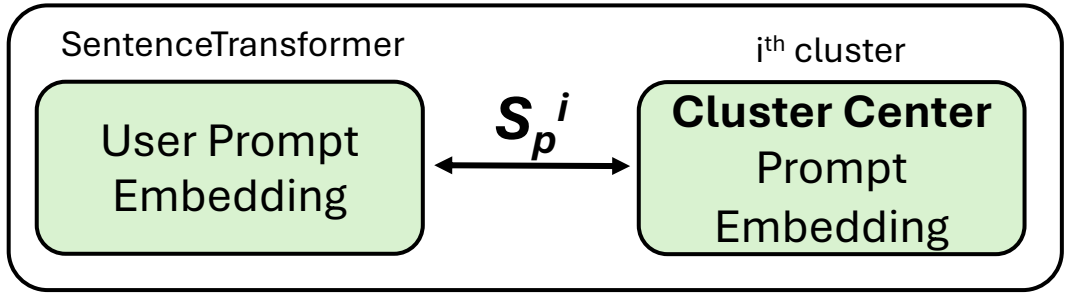
Prompt Clustering

- Structurally Similar (P, R) pairs are clustered together
- Prompt similarity computed against cluster center
- Program generated for each cluster
- Cluster ID used to choose cache for an incoming prompt

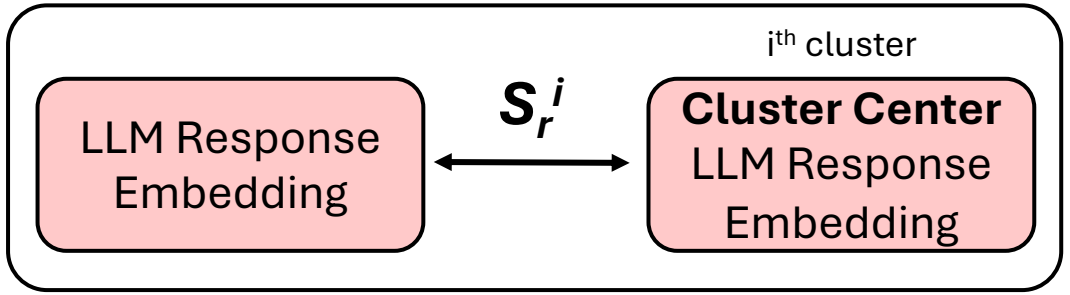


Cluster Insertion

Prompt Similarity



Response Similarity



Nearest Cluster = $argmax([S_p^i > T_p] + [S_r^i > T_r])$

Program Generation

In-context exemplars

CodeGenLLM Prompt

Prompt: You are an expert who can **analyze a few example** inputs and their outputs and find the common pattern that can generate the output from the inputs.

Example Inputs and their Corresponding Outputs

1. Input Prompt:

   Response: {...}

2. ....

The goal is to **find a pattern and generate a Python code**. The code should be able to take an arbitrary input (all following the same template) and generqate the output.

Guidelines to Follow:

....

There should be no output apart from the code.

The **format of the output** should match exactly as the example outputs.

CodeGenLLM

Program

```
import re
import sys

def func(prompt):
    try:
        user_instr = re.search(...)
    ------------------------
    ------------------------
    ------------------------
        print(f"The tenant name is {tenant_name}")
    except:
        print("None: Invalid prompt")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("None: Invalid args")
    else:
        prompt = sys.argv[1]
        tenant_name = func(prompt)
```

Evaluations

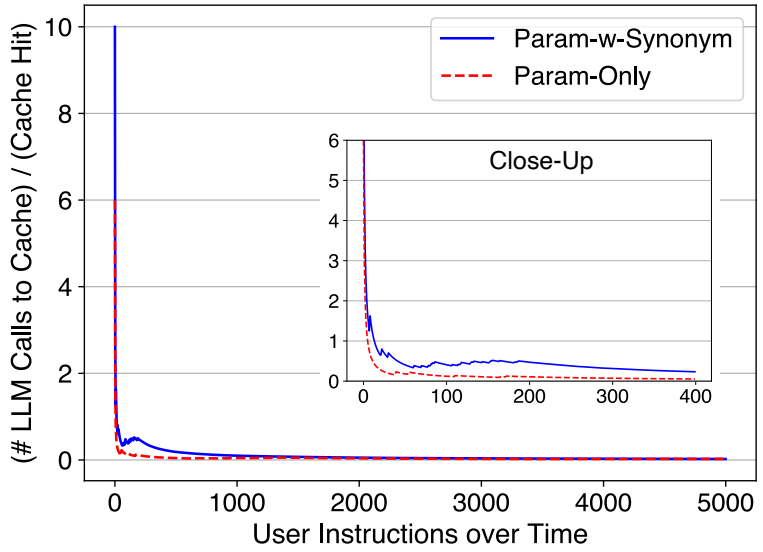
Hit Rate Measurement

Method	Param-Only			Param-w-Synonym		
	Hit %	+ve Hit	-ve Hit	Hit %	+ve Hit	-ve Hit
ExactCache	0 (± 0.0)	N/A	N/A	0 (± 0.0)	N/A	N/A
GPTCache [14]	90.92 (± 0.02)	0 (± 0.0)	100 (± 0.0)	88.71 (± 0.01)	0 (± 0.0)	100 (± 0.0)
GenCache	**97.81 (± 0.96)**	98.03 (± 0.05)	1.97 (± 0.05)	**83.66 (± 6.37)**	92.16 (± 0.12)	7.84 (± 0.12)

- Match **semantic correctness of the responses** and whether the item is searchable in an e-commerce website (+ve vs –ve Hit)

Cache Generation Cost

- Incurs an **initial cost** for using CodeGenLLM and ValidLLM
- Once cached programs are reused, **cache hits increase sharply**



Summary

- We propose **GenCache**, a novel caching technique for **structurally similar prompts**
- GenCache uses LLM to **identify a consistent common pattern** to generate responses from similar prompts, and caches the **pattern as an executable program**
- On cache hit, the program is executed to generate **variation-aware responses**

Paper

Our Lab

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Microsoft

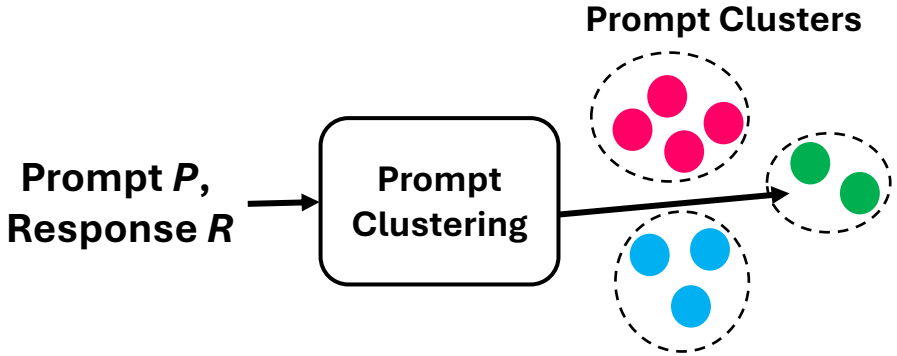
Generative Caching for Structurally Similar Prompts and Responses

BACKUP SLIDES

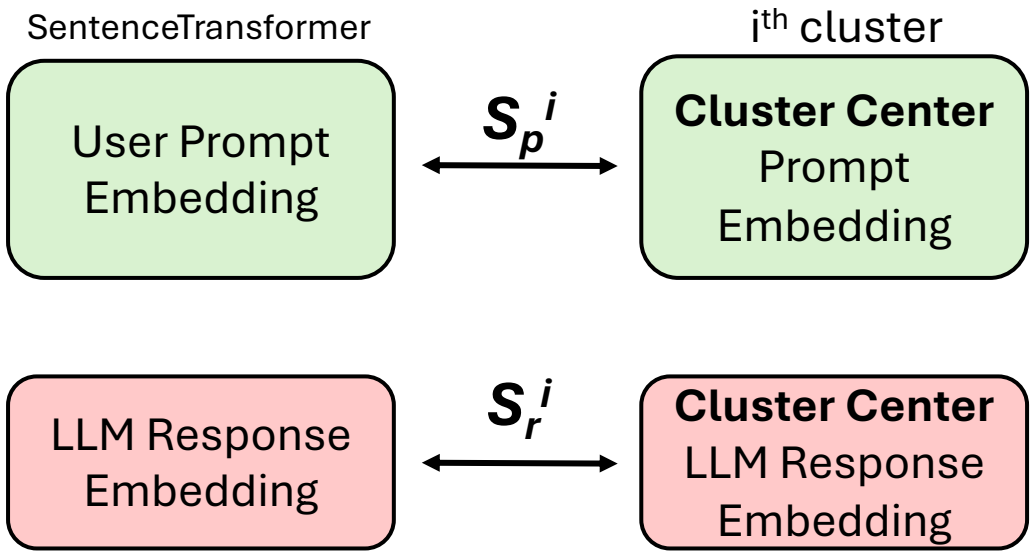
NEURAL INFORMATION PROCESSING SYSTEMS

Prompt Clustering

- Structurally Similar (P, R) pairs are clustered together
- Prompt similarity computed against cluster center
- Program generated for each cluster
- Cluster ID used to choose cache for an incoming prompt

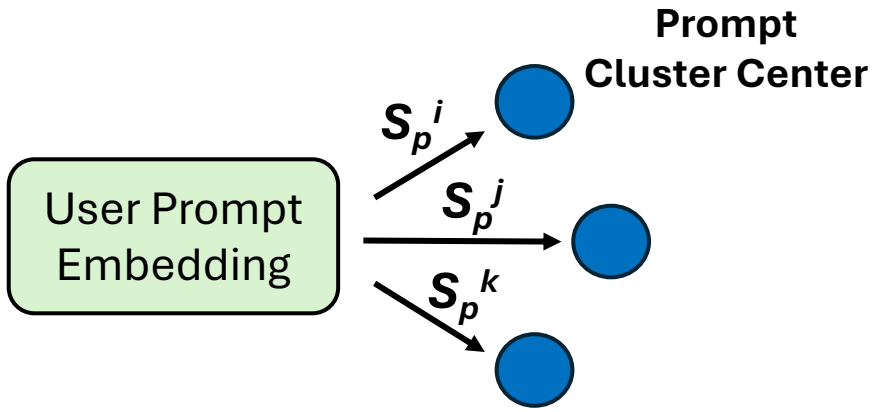


Cluster Insertion



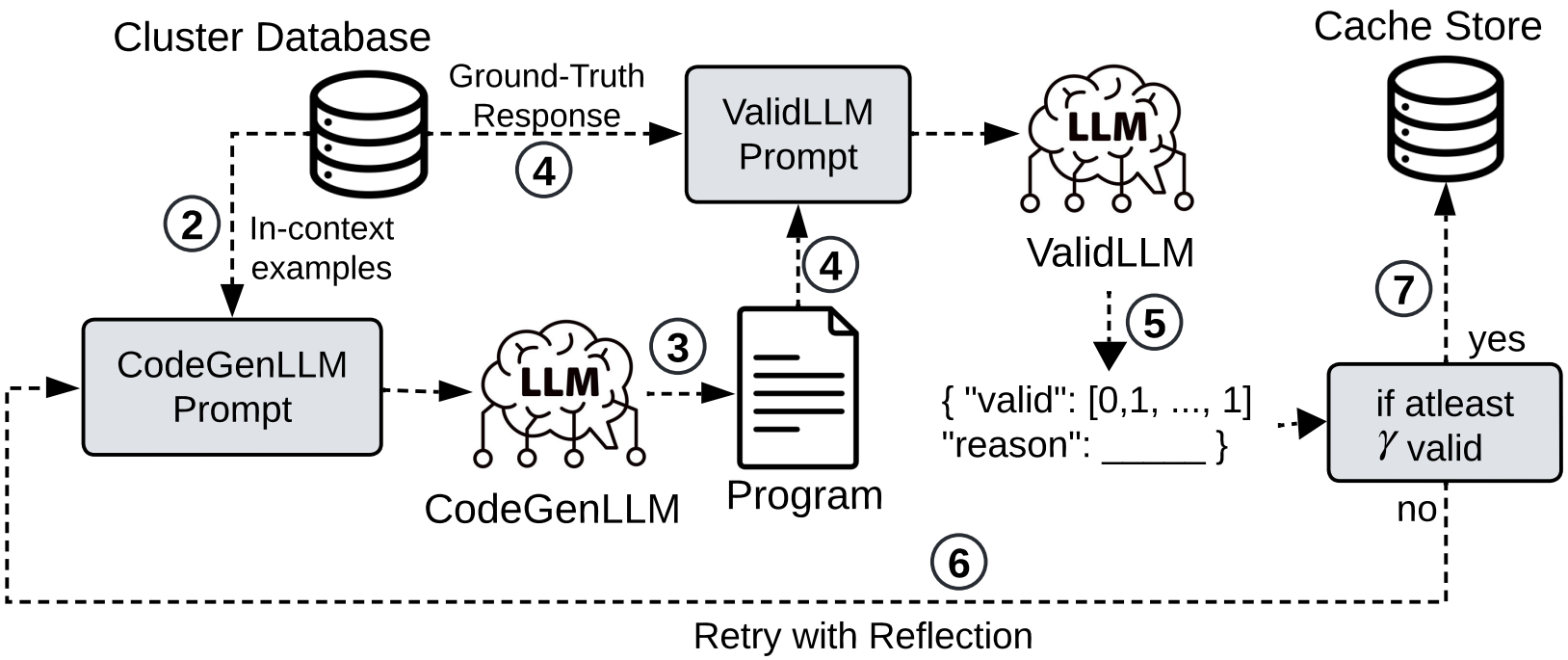
$$argmax([S_p^i > T_p] + [S_r^i > T_r])$$

Cluster Selection



$$argmax([S_p^i > T_p])$$

Program Validation



④ **Match in-context exemplar responses** with program-generated responses

⑤ Boolean value verifies the output format and keywords

⑦ If a **threshold** number of exemplars **match**, store it as cache

GenCache's Impact on Agentic Workflow

- SRE Agent identifies correct TSG, generate coarse and fine-grained plan, and diagnose (avg. 16 calls per incident) – Total 4725 calls to GenCache API
- GenCache was enabled along with Exact Prompt matching
- Total Cache hits: 54.7%, out of which 31.1% were due to exact prompt matching
- Reduced avg. diagnosis time by ~25%
- Extra 536 LLM calls for cache creation
  - 10% created reusable caches
- Failures
  - 1% due to incorrect incident → TSG mapping