ICML
International Conference
On Machine Learning

Functional Cache Grafting

Robust and rapid code-policy synthesis for embodied agents via function-level KV caching.

Saehun Chun · Wonje Choi · Sera Choi · Sanghyun Ahn · Honguk Woo*

Department of Computer Science and Engineering, Sungkyunkwan University

SUNGKYUNKWAN UNIVERSITY
1398
성균관대학교

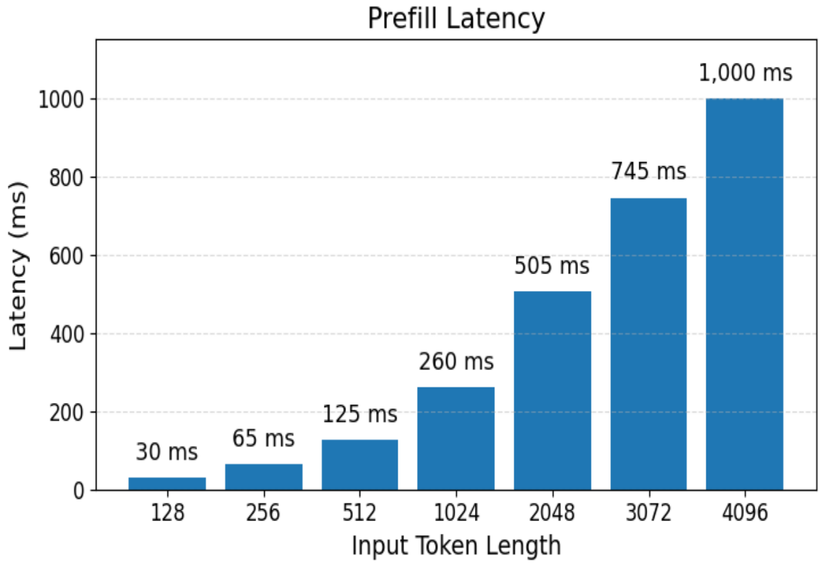
Problem — Code-as-Policies is slow & brittle

Latency

- Repetitive prefill over long CaP prompts
- Full regeneration after execution errors

Robustness

- Inconsistent code from fully generative decoding
- API mismatches, missing safety guards, unstable control logic



Task Specification

```
# Task
# Put office supplies in the drawer.
# ...
```

Ground-truth

```
def task():
    handle = "drawer_top"
    safety_check(handle)
    pick(handle)
    pull(handle)
    pick("pen")
    place("pen", "drawer_inside")
```

√ Satisfy drawer opening requirement

CodeLLM Generation

```
def task():
    handle = "drawer_high"
    pick(handle)
    pull(handle)
    pick("pen")
    place("pen", "drawer")
```

X Overlook drawer opening requirement

Idea — Reuse validated functions, not full prompts

Cache **function-level KV states** of past validated policies and **graft** them into new ones.

Key idea: avoid full regeneration by reusing and patching validated function caches.

1

Function-level KV Cache

Store each validated function in Function-Interface and Function-Code tiers, both indexed by function ID.

2

Cache-Stitching

Reuse cached interface KVs to reduce prefill and synthesize more robust code policies.

3

Cache-Patching

On runtime exceptions, generate only the corrected span while preserving the prefix and suffix.

Motivating Scenario — Gas-Leakage Handling in Open-Domain Robot Tasks

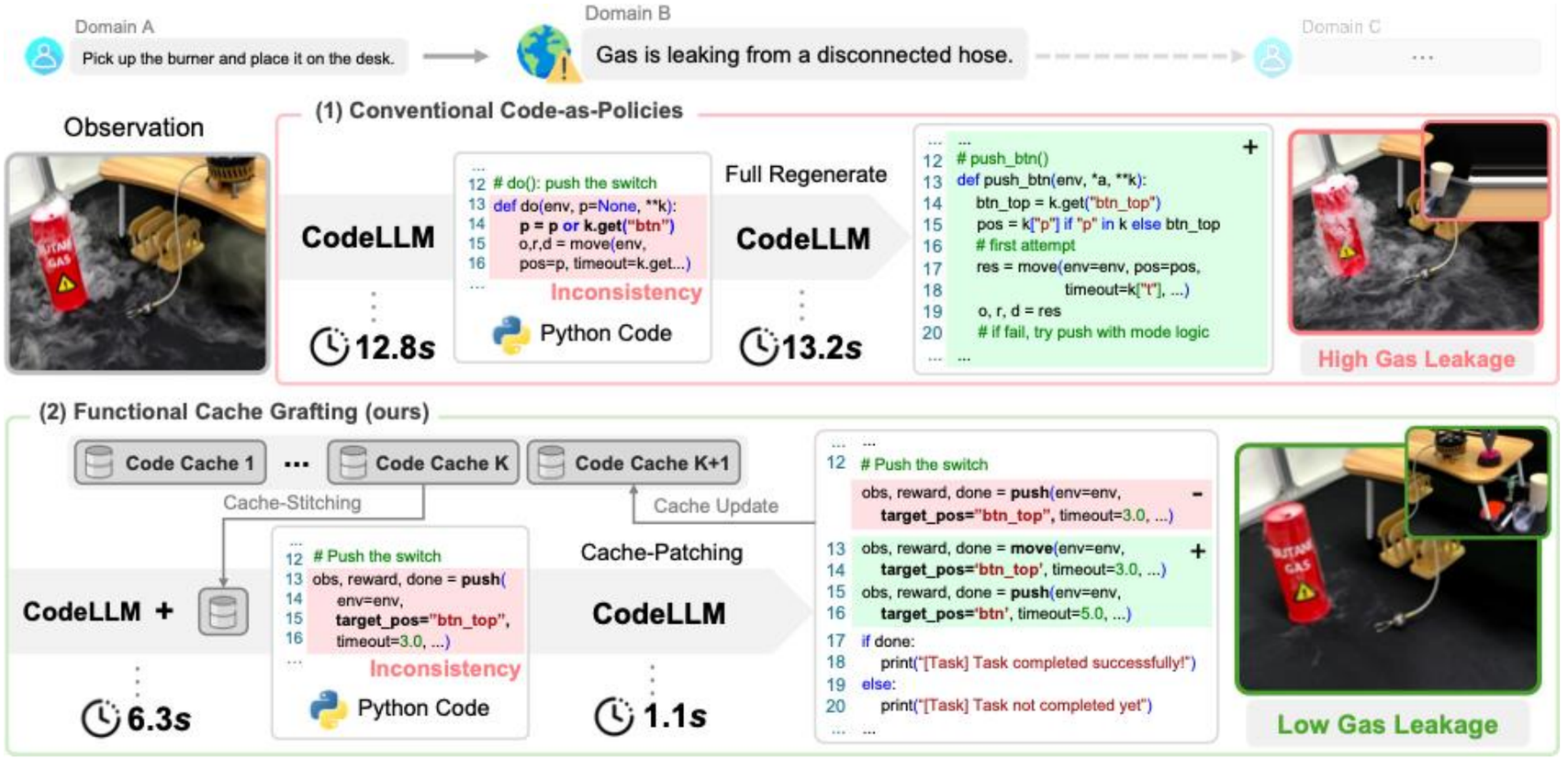
In dynamic robot tasks, delayed or inconsistent code generation can turn small execution errors into cascading failures.

Conventional CaP:

full regeneration delays response and may produce inconsistent control logic

FCGraft:

reuses cached functions and patches only faulty spans for rapid recovery



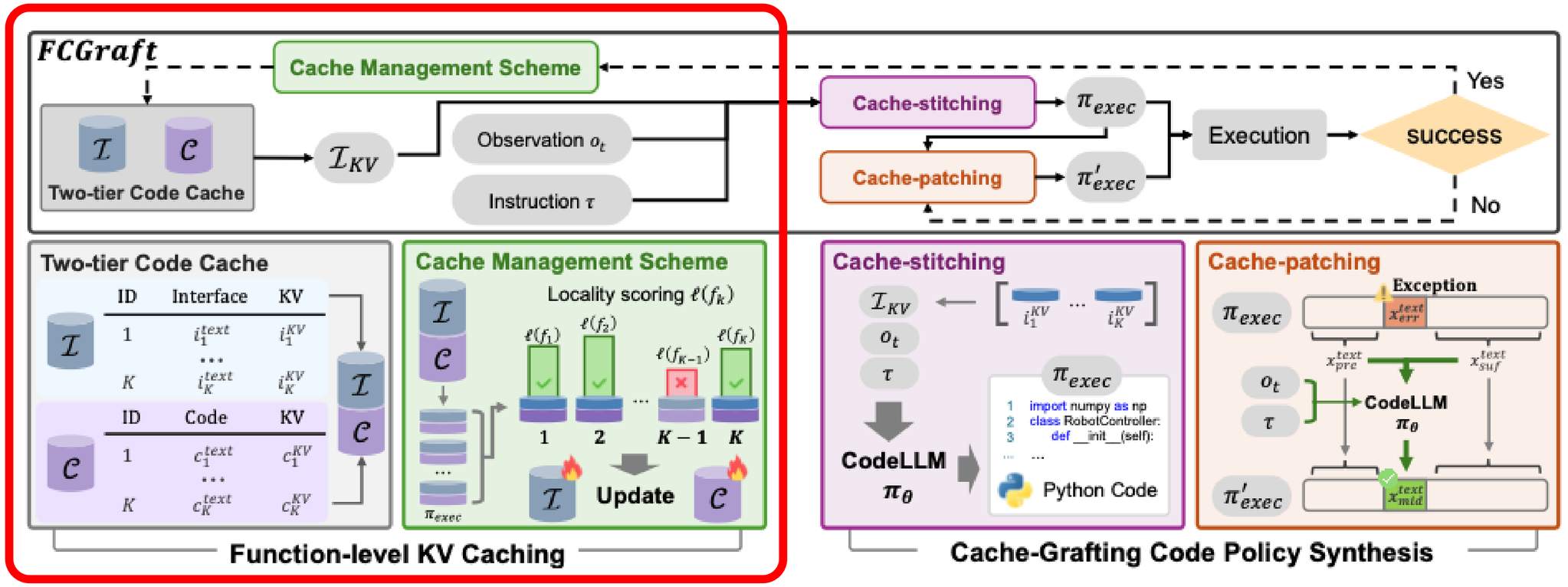
FCGraft — Two-Tier Function-level KV Cache

Function-level KV Caching: a two-tier cache of modular function-level KVs for fast policy generation and recovery.

Function-Interface tier (I): stores function **signatures + KV** → supports writing function-call code.

Function-Code tier (C): stores **code body + KV** → supports both execution and in-place **patching**.

The hierarchy improves **GPU memory utilization** and the **reusability** of execution-validated functions.



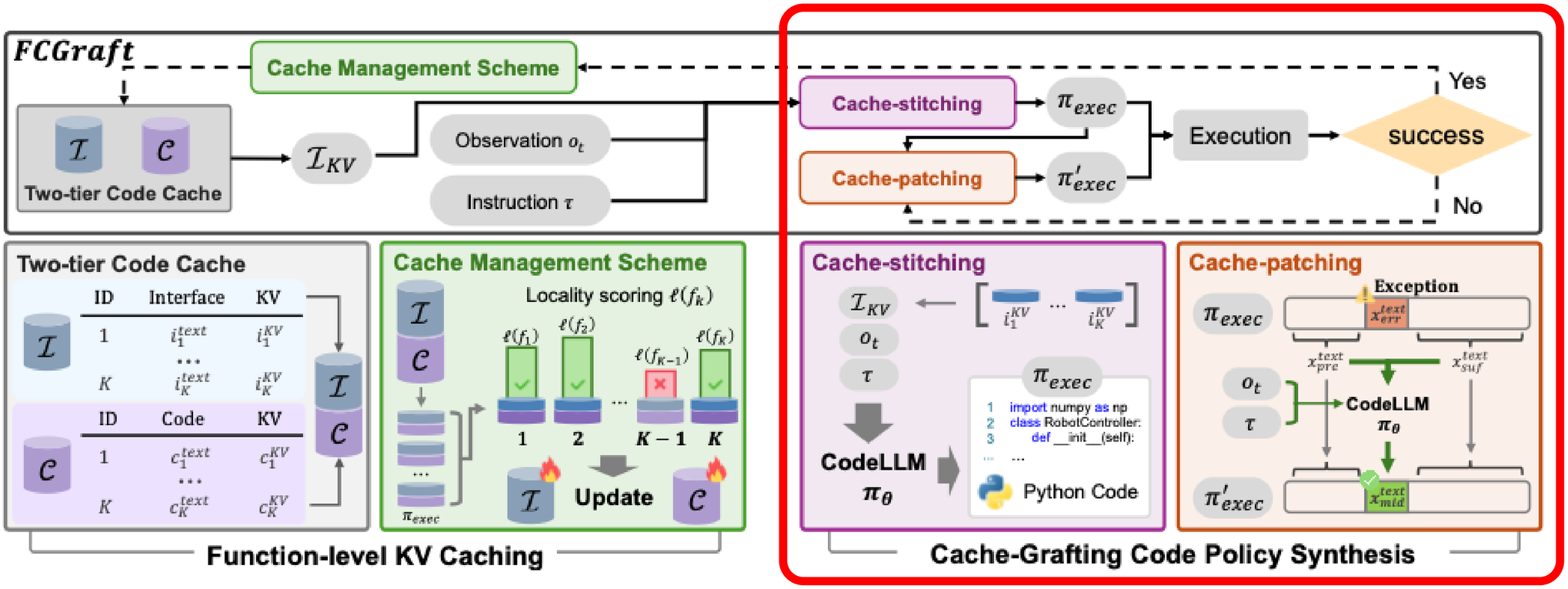
FCGraft — Cache-Grafting Code Policy Synthesis

Cache-Grafting: reuse function-level caches for cheap generation and patch runtime errors without full rewriting.

Cache-stitching: retrieve function-unit KV caches and **compose** them → efficient code generation, skipping prefill.

Cache-patching: pre-mark exception-prone spans → **partially edit** the KV cache around just the failing region.

Selective re-processing of only the **necessary composition** and **error region**.



Evaluation Setup & Results

Environments

- Simulation(RLBench, ALFRED, TEACh) and Real-world validation(Franka Emika Research 3)
- Metric : Success Rate(SR), Policy Synthsis Latency(PSL), …

Results

- Rank 1 overall

- **61.58%** task success rate

- **5.82s** policy synthesis latency

ALFRED Result

Method	SR ↑	Latency (s) ↓	Rank ↓
CaP	46.48	16.21	6
HELPER	53.83	16.82	4
LRLL	56.28	16.02	2
RAGCache	39.29	13.48	5
PromptCache	40.37	12.82	3
FCGraft (ours)	**61.58**	**5.82**	**1**

Future Work

- Shared code caches for multi-agent collaboration