Graph Structure for Data-Efficient Agentic Learning

Zixing Song

Three Graph Lenses

Classify a graph by where its operator acts in the agent loop

EXPERIENCE GRAPH

NODE tool / fact / state

EDGE compatibility / relation

USE generate → verify → grade

WHY more valid supervision

DECISION GRAPH

NODE state / routine / memory

EDGE action / reachability

USE search → retrieve → reuse

WHY fewer external trials

LEARNING GRAPH

NODE state / LM module

EDGE transition / latent transform

USE merge → credit → backprop

WHY denser signal per label

Three Graph Lifetimes

DIMENSION	EXPERIENCE	DECISION	LEARNING
Typical node	tool / fact / state	state / routine	module
Typical edge	compatibility / relation	action / reachability	latent transition
Operator	generate / verify	search / retrieve	credit / backprop
Lifetime	pipeline	query → tasks	rollout

01 / EXPERIENCE GRAPHS

Turn structure into supervision

Trajectory2Task: API DAG

NODE

API / tool endpoint

EDGE

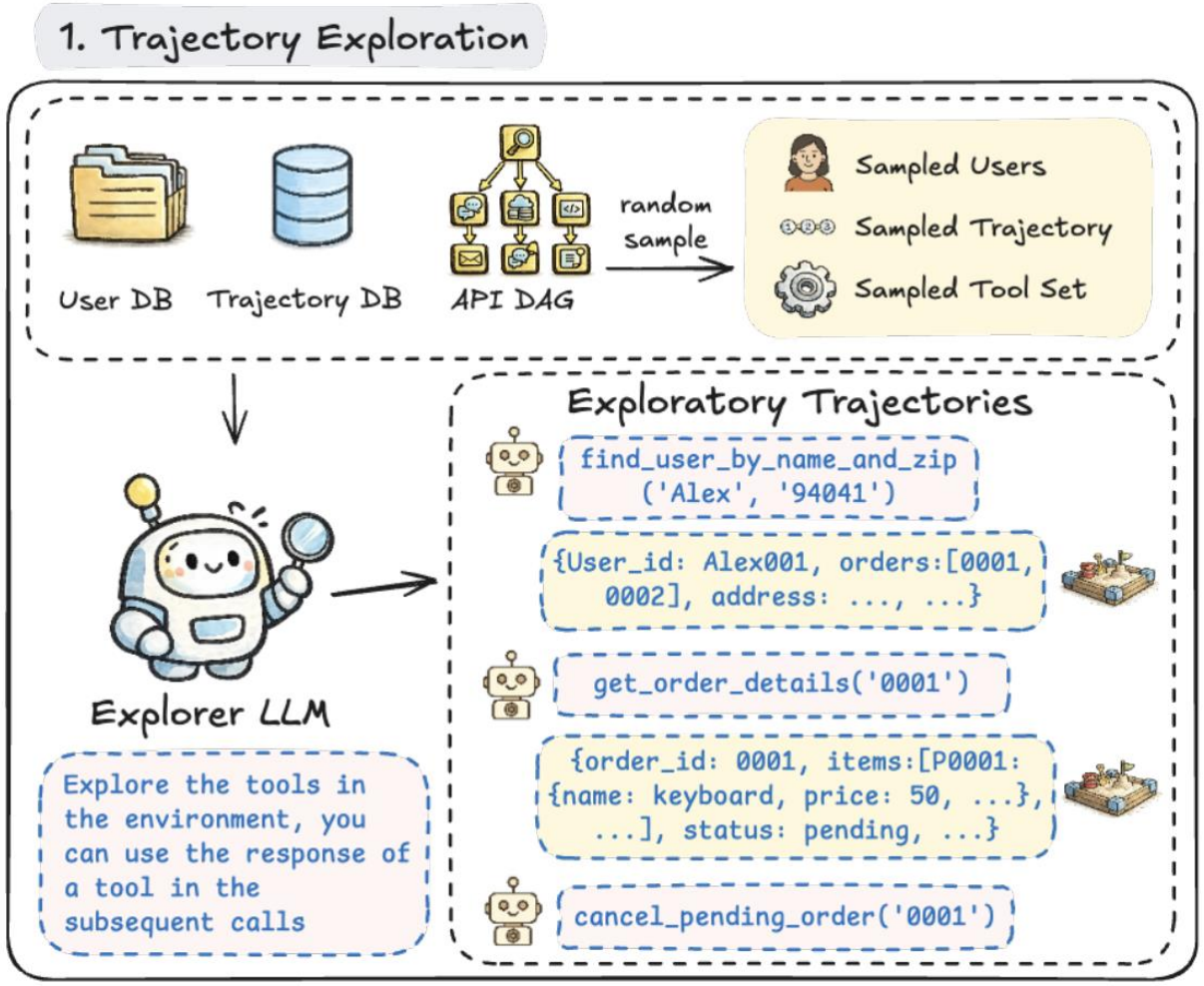
output of one API can feed another

USE

walk the DAG, then execute the composition

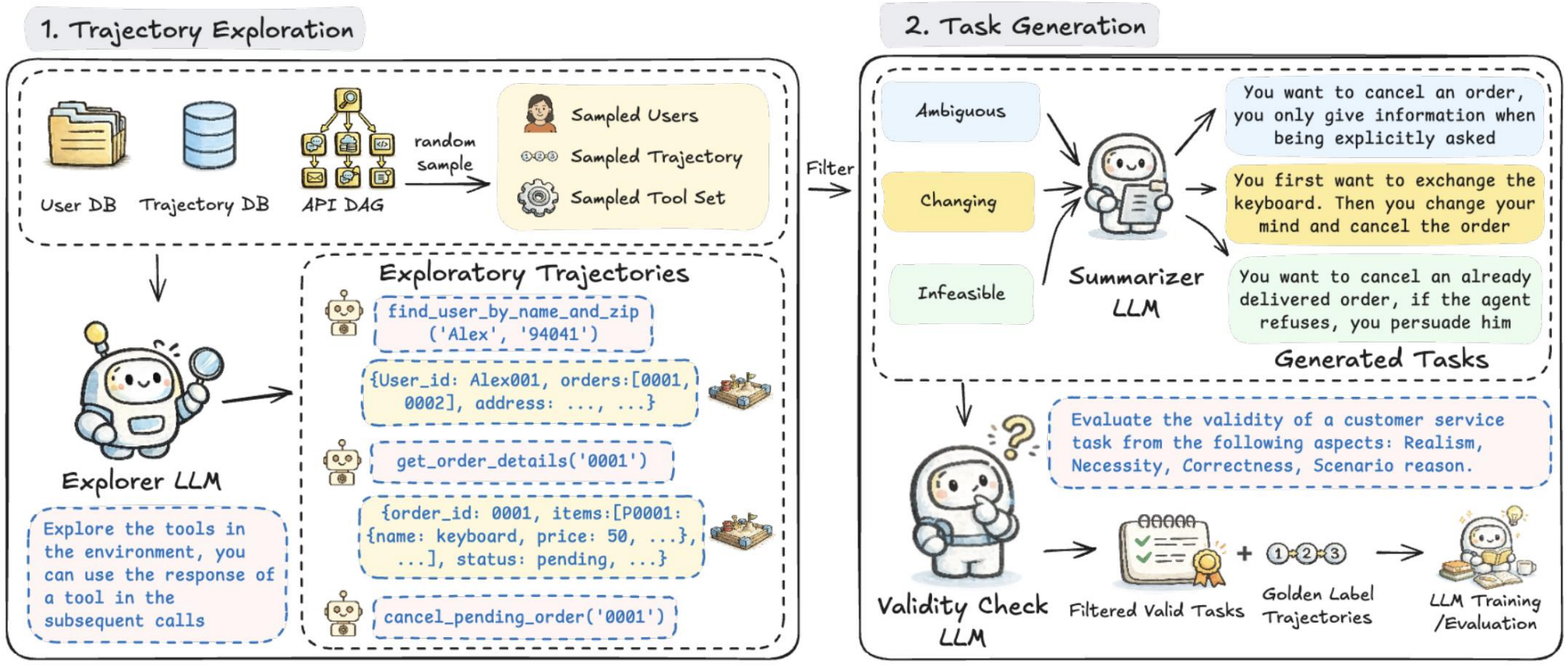
DATA EFFICIENCY

Sample more trajectories



DAG Structure Creates Verifiable Provenance

Generate the task after execution, so every label is anchored to an observed state transition.



Kowledge Graph (KG) Paths: Reuse Structure

NODE

Wikidata entity

EDGE

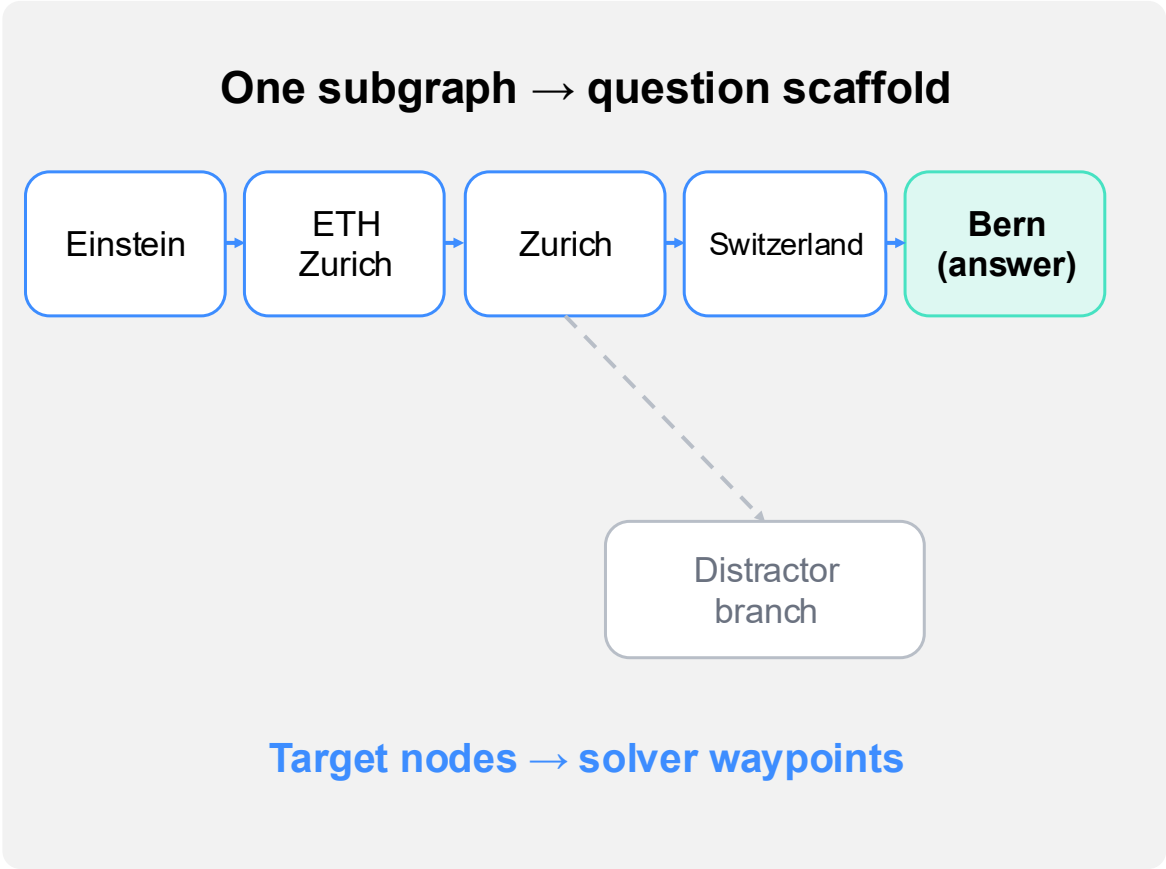
typed factual relation; target + distractor branches

USE

scaffold the question; retain target nodes as waypoints

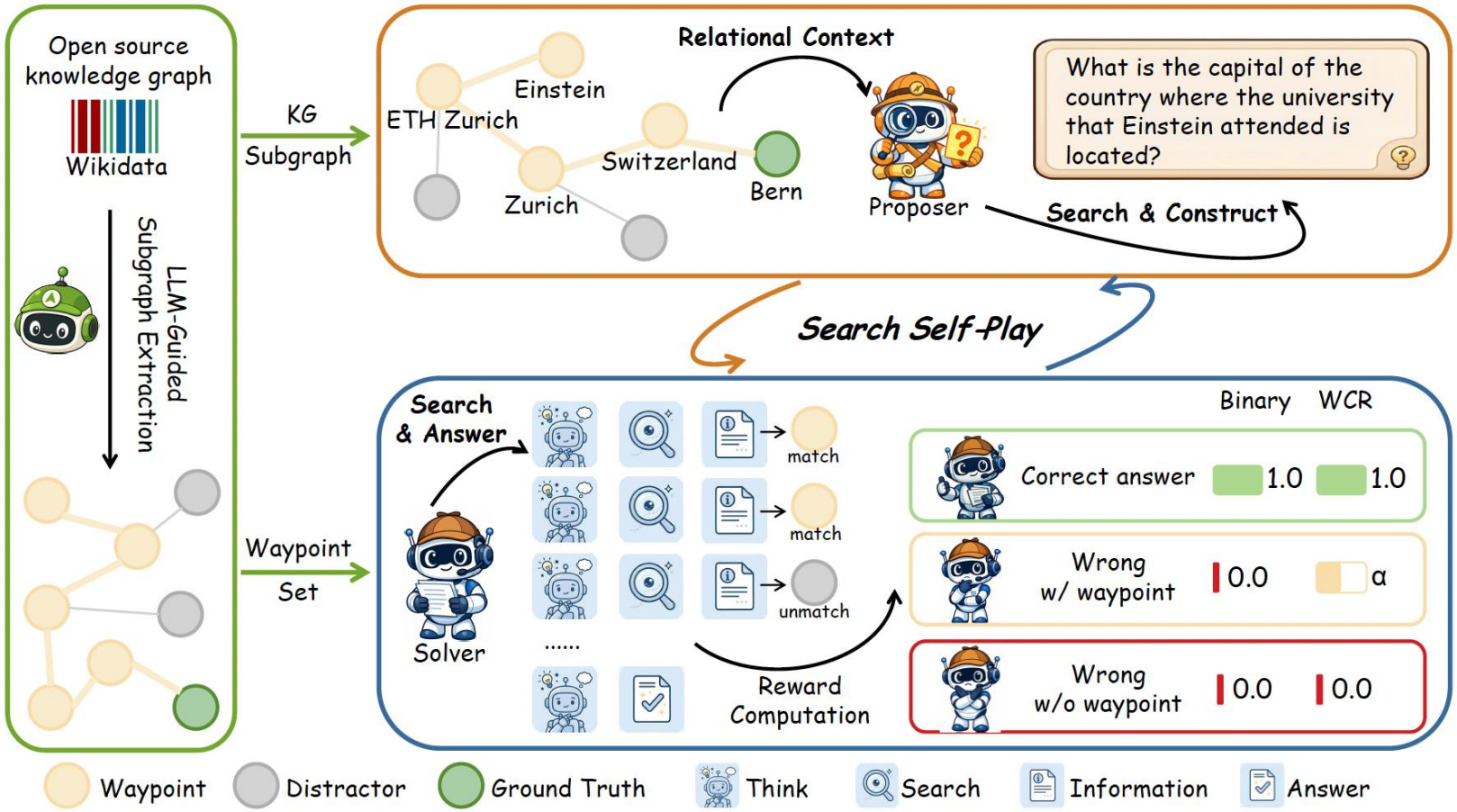
DATA EFFICIENCY

Process signal without rationale labels



Waypoint Coverage Reward Recovers Signal from Failure

Correct = 1; wrong but on-path gets partial credit; invalid = 0.



KG-RAG: Reachable Paths

NODE

GUI screen / actionable UI state

EDGE

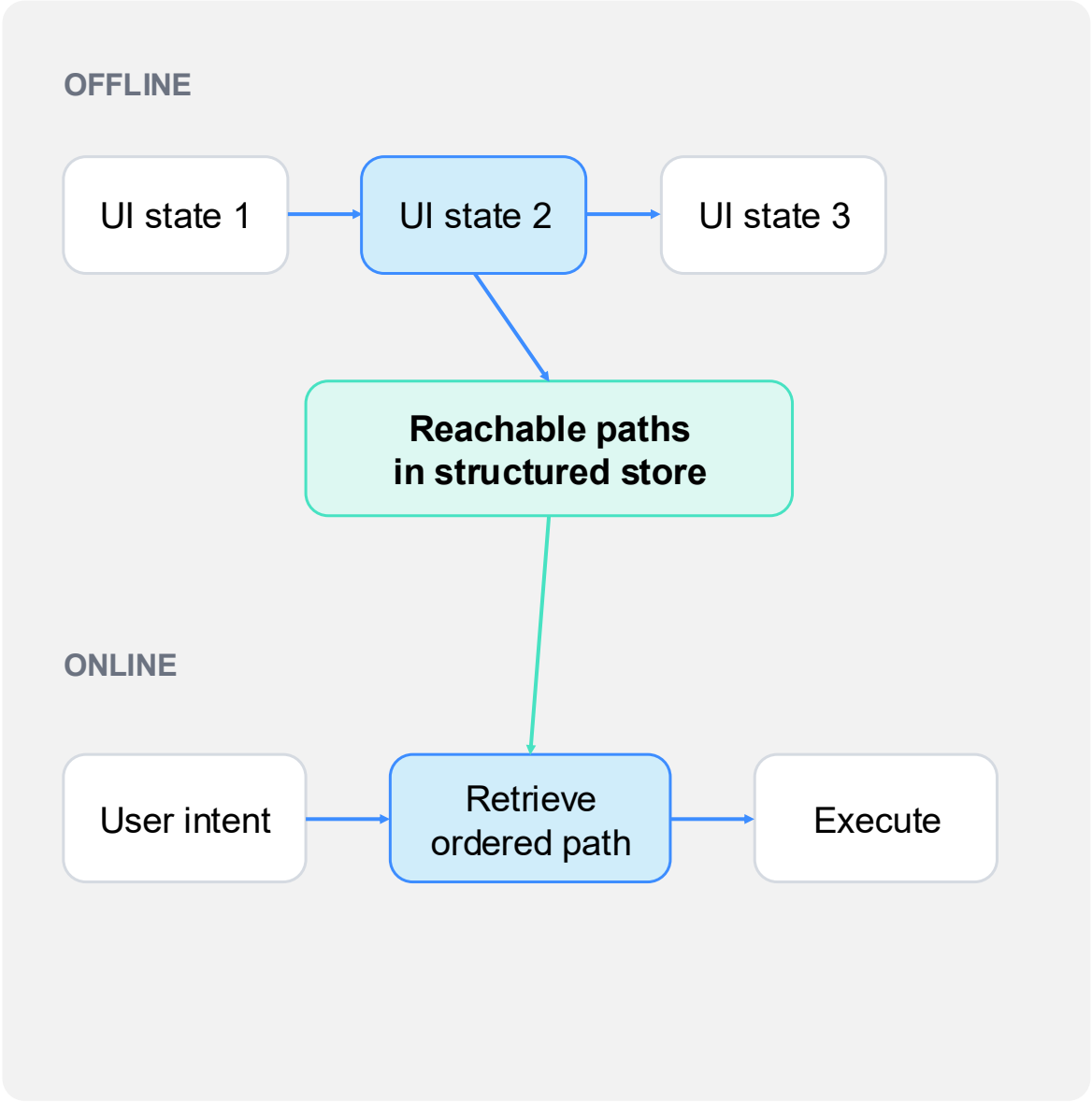
action-labeled transition to the next state

USE

build graph offline; retrieve an ordered path online

DATA EFFICIENCY

Reduce online trial-and-error



KG-RAG

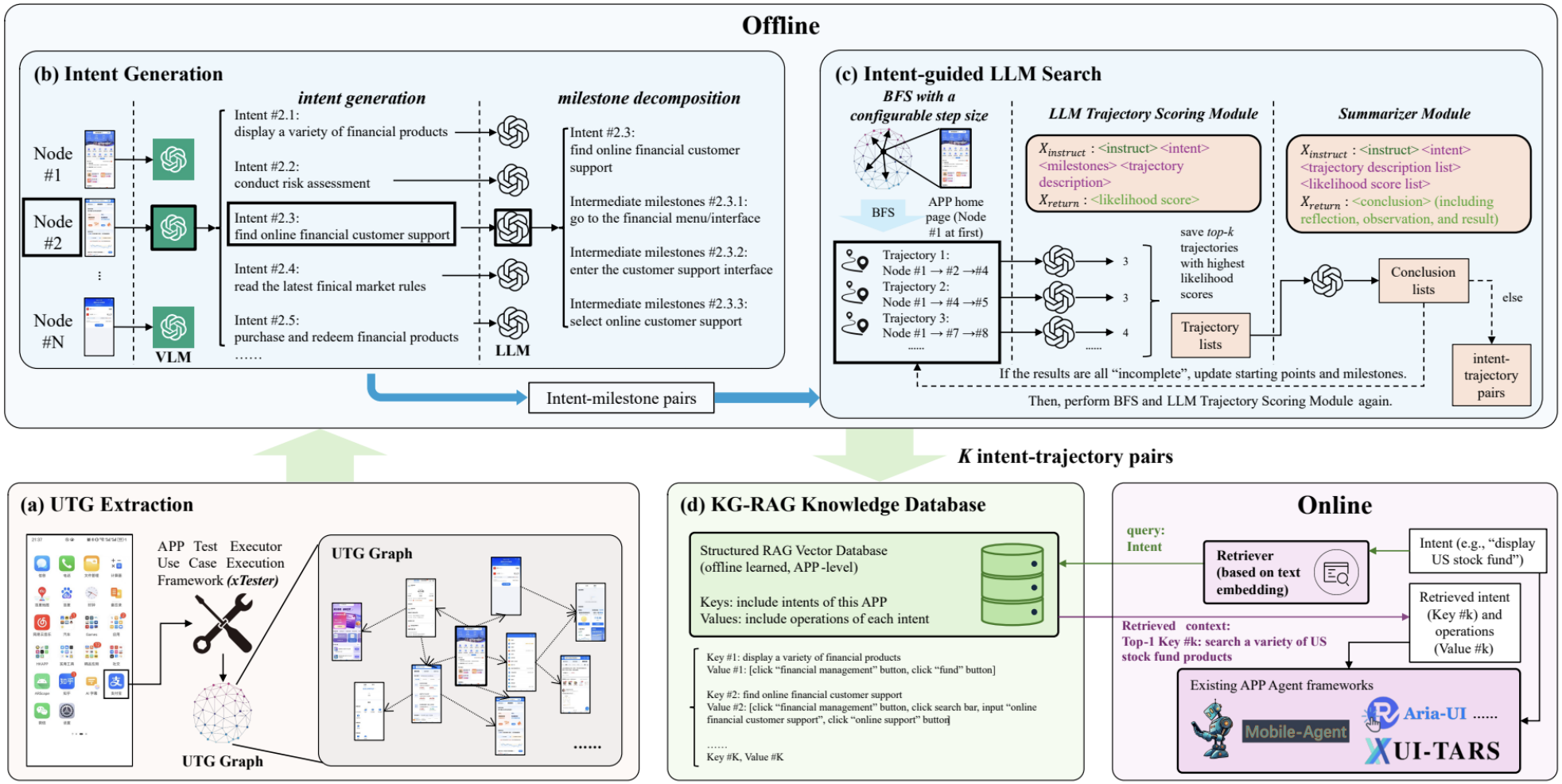


Figure 2: Overview of KG-RAG architecture: **(a) UTG extraction** capturing app UI navigation structures; **(b) Intent generation** suggesting plausible user intents and decomposing them into intermediate milestones; **(c) Intent-guided LLM search** efficiently identifying candidate trajectories aligned with user intents; and **(d) KG-RAG knowledge database** supporting effective online mobile app interactions.

02 / DECISION GRAPHS

Allocate search and Reuse exploration.

LATS: Search a Decision Tree

NODE

task-local state / interaction history

EDGE

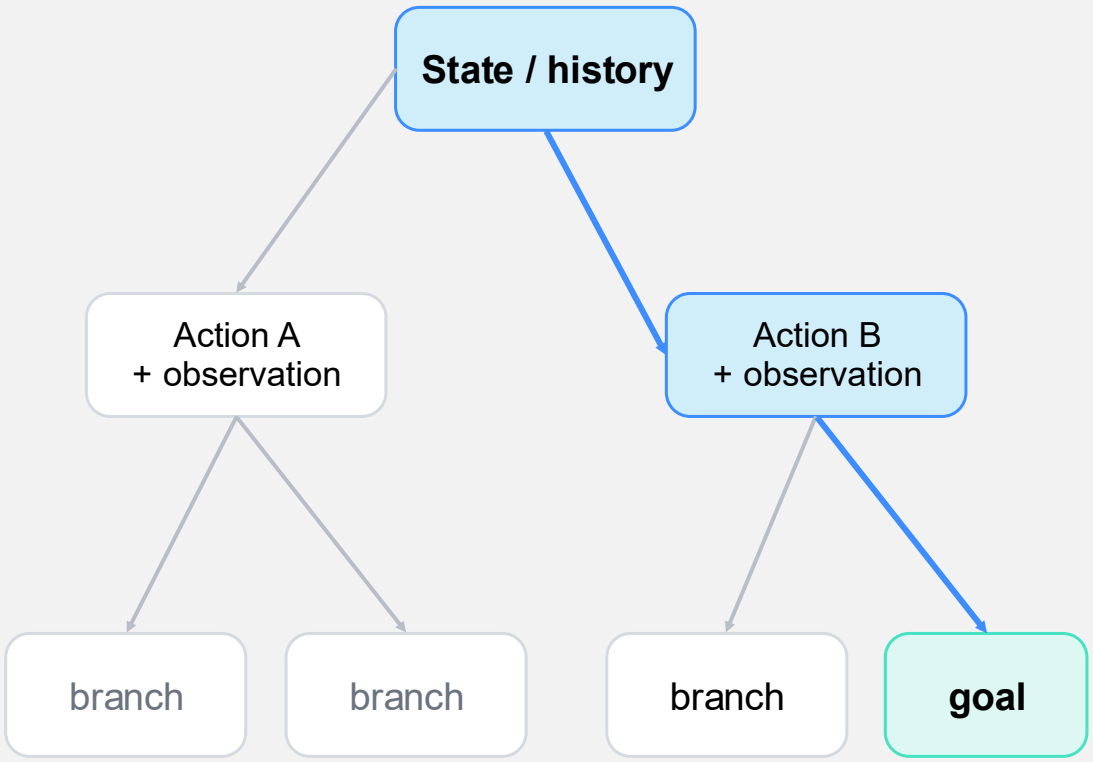
candidate action followed by observation

USE

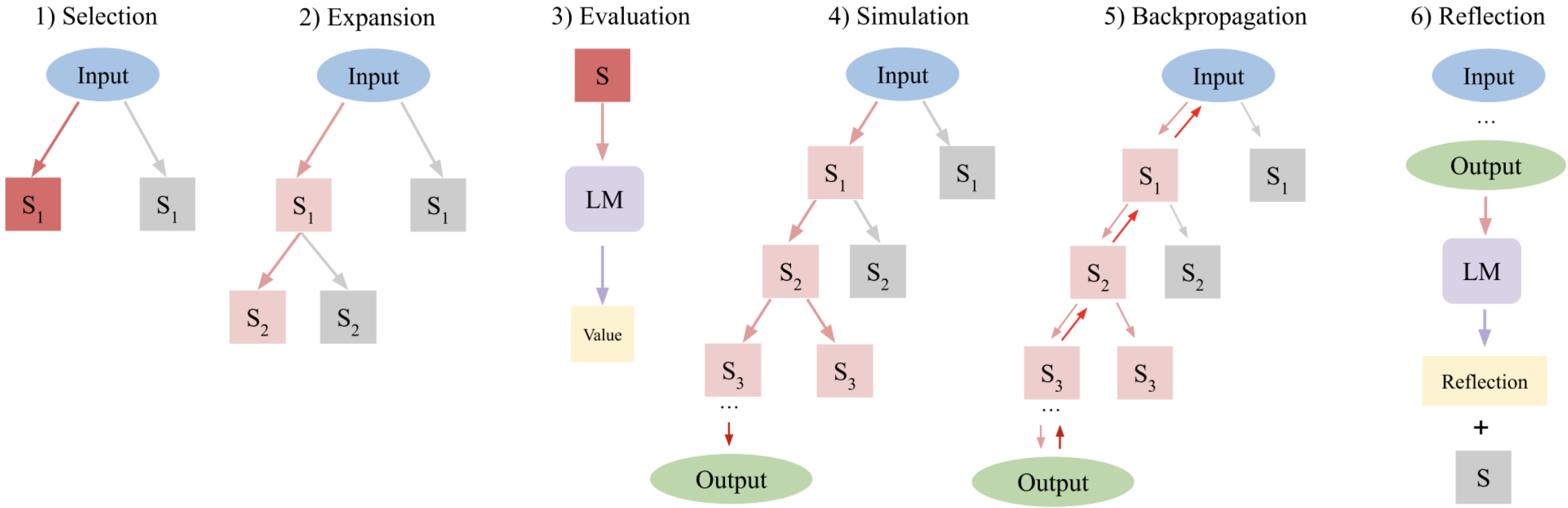
MCTS expansion, reflection, value, and backup

DATA EFFICIENCY

Better utility from a limited within-query search budget



LATS: Search a Decision Tree



LATS Improves Within-Query Search Efficiency

Algorithm 1 LATS$(s, p_\theta, p_V, p_{\text{ref}}, d, k, n, w, a, b)$

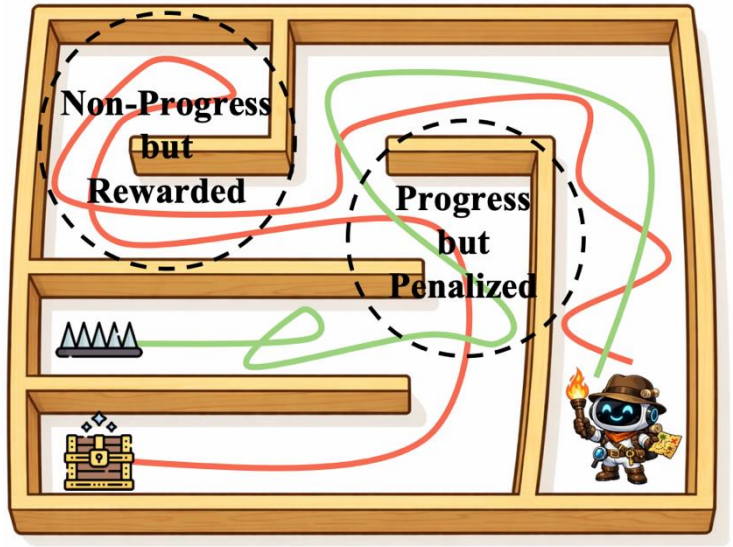
Require: Initial state s, action generator p_θ, value function p_V, reflection generator p_{ref}, number of generated actions n, depth limit L, number of roll-outs K, context c, exploration weight w, and value function weight λ
Initialize action space A, observation space O
Initialize the state-action value function $p_V : S \times A \mapsto \mathbb{R}$ and visit counter $N : S \mapsto \mathbb{N}$ to one
for $k \leftarrow 0, \ldots, K-1$ **do**
  for $t \leftarrow 0, \ldots, L-1$ **do**
    if s_t not terminal **then** ▷ Expansion & Simulation
      for $i \leftarrow 1, \ldots, n$ **do**
        Sample $a_t^{(i)} \sim p_\theta(s_t)$
        Get $o_t^{(i)}$ from environment, $s_{t+1}^{(i)} \leftarrow (c_t^{(i)}, o_t^{(i)}, a_t^{(i)})$, $c_{t+1}^{(i)} \leftarrow (o_t^{(i)}, a_t^{(i)})$
        Evaluate $V_t^{(i)} \sim \lambda * p_V(s_t^{(i)}) + (1-\lambda) * \text{SC}(s_t^{(i)})$ ▷ Evaluation
        $V(s_t) \leftarrow V_t^{(i)}$
        Add $s_t^{(i)}$ to children
      end for
    end if
    if s_t is terminal **then** ▷ Reflection
      Get r from environment
      if r not success **then**
        reflection $\leftarrow p_{\text{ref}}(c_t)$
        $c \leftarrow$ reflection
      end if
    end if
    $a_t \leftarrow \arg\max_{a \in e(s_t)} \left[V(s_t) + w\sqrt{\frac{\ln N(s_t)}{N(s_{t+1})}} \right]$ ▷ Selection
    Get corresponding o_t from memory, $s_{t+1} \leftarrow (c_t, o_t, a_t)$, $c_{t+1} \leftarrow (o_t, a_t)$
    $N(s_{t+1}) \leftarrow N(s_{t+1}) + 1$
    if a_t is an output action **then break**
  end for
  $T \leftarrow$ the actual number of steps
  for $t \leftarrow T-1, \ldots, 0$ **do** ▷ Backpropagation
    $V(s_t) \leftarrow \frac{V(s_t)(N(s_t)-1)+r}{N(s_t)}$
  end for
end for

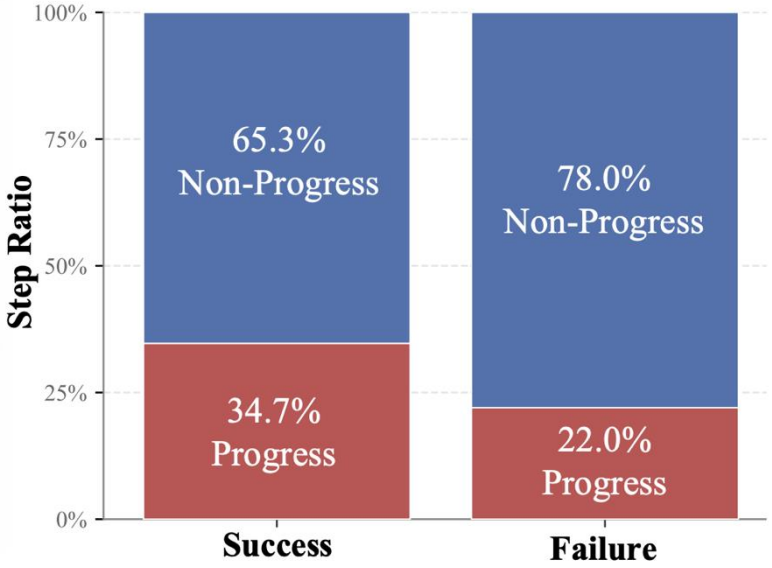
03 / LEARNING GRAPHS

Make sparse feedback local and reusable

Trajectory-Level Credit Wastes Useful Transitions

One trajectory scalar gives every step the same sign, even when step quality differs.





22% of failed steps progress

- useful prefixes stay hidden
- coarse negative advantage
- recoverable interaction signal

65.3% of successful steps stall

- detours receive positive credit
- coarse positive advantage
- reward is spent on non-progress

Cheng et al., Beyond Trajectory-Level Attribution: Graph-Based Credit Assignment for Agentic Reinforcement Learning. ICML'26

GraphGPO: Credit on Edges

NODE

canonical environment state merged across rollouts

EDGE

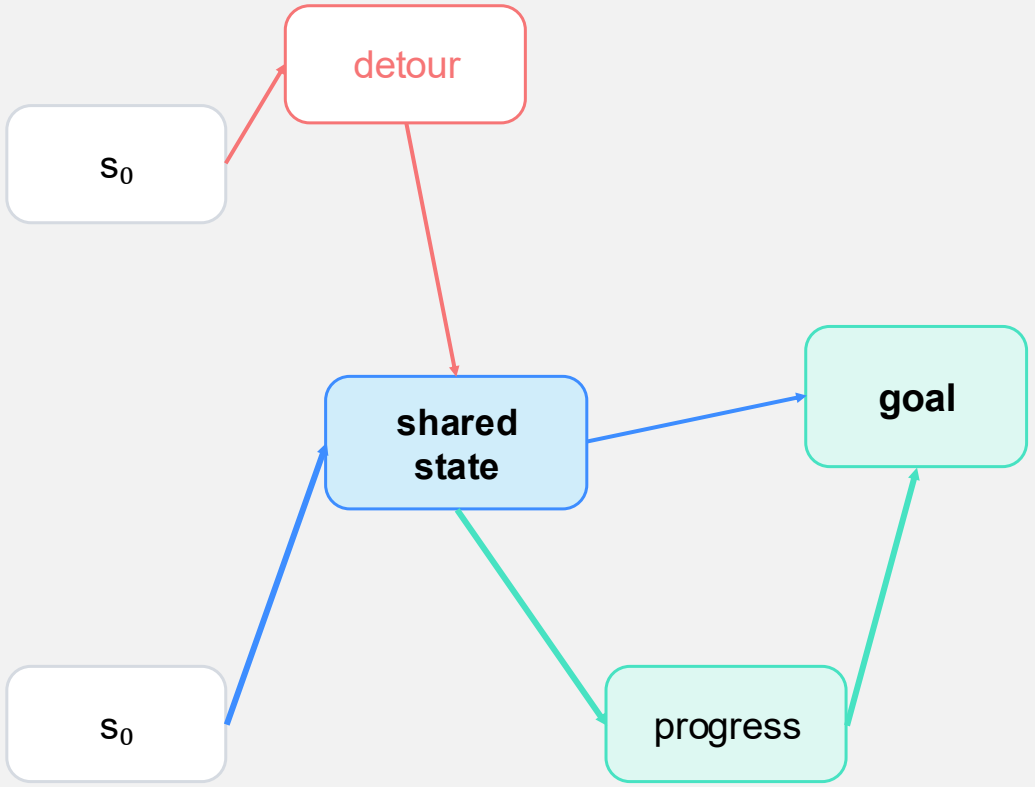
observed action transition (s, a, s')

USE

distance-to-success → edge reward → policy advantage

DATA EFFICIENCY

No process labels or critic



shorter distance to success ⇒ higher edge credit

GraphGPO Learns More from the Same Rollouts

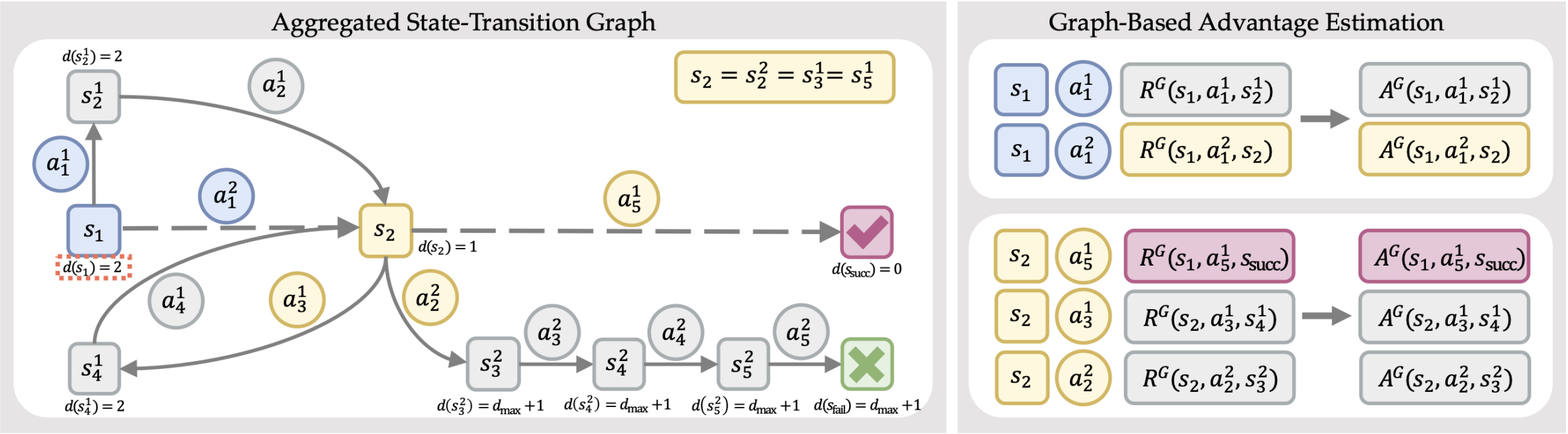


Figure 3. Overview of GraphGPO. For simplicity, we assume all transition costs are unitary, i.e., $c(s, \boldsymbol{a}) = 1$. **Left:** The aggregated state-transition graph constructed based on states from rollout trajectories, where identical states are merged (e.g., $s_1 = s_1^2 = s_2^1 = s_1^4$). **Right:** Graph-based advantage estimation, where the credit assignment relies on the the distance $d(\cdot)$ of the next state. Taking the initial state s_1 as an example, the shortest path to the goal state s^s is $\{(s_1, \boldsymbol{a}_1^2, s_2), (s_2, \boldsymbol{a}_5^1, s_{\text{succ}})\}$, yielding $d(s_1) = 2$. Moreover, the maximum shortest-step distance among all states that can reach the goal is $d_{\max} = 2$.

Summary

The Three Graphs Intervene at Different Points

Classify a graph by where its operator acts

EXPERIENCE

tool / fact / state nodes

compatibility / relation edges

generate + verify → labels

DECISION

decision / memory nodes

action / reachability edges

search + retrieve → fewer trials

LEARNING

state / module nodes

transition / latent edges

assign + backprop → dense signal

Choose the Appropriate Graph under Different Resource-scarce Settings

Weak synthetic data

API/entity nodes + compatibility/relation edges. Generate and verify;

Costly current trial

State nodes + action transitions. Search and constrain;

Repeated task family

Case/workflow nodes + similarity/precedence. Retrieve and reuse;

Sparse outcomes

State/module nodes + transition/latent edges. Assign or backprop;