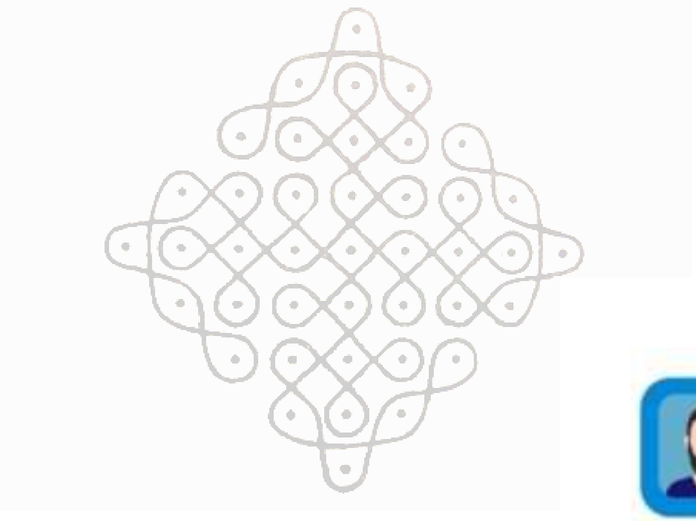
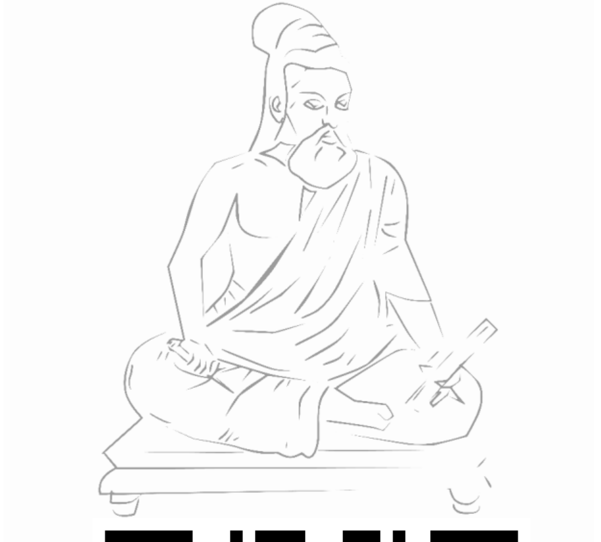




Azure Developer Community

Tamil Nadu

What we do?

Meetups

Network of Beginners, Professionals & Experts

Azure Cert Study Jam

Tech-talks / workshops
In Colleges

Who am I?

Kishore Kumar L

- Senior AI/ML Engineer
- Working deeply with GenAI + agent systems
- Community Organizer - Elastic User Group Chennai

Have you worked with agents in production?

The Hook and The Conflict

The Randomness Trap

- The Robot Arm: Your Python Code (Rigid, Precise, Unforgiving).
- The Jelly: The LLM (Fluid, Wobbly, Approximate).
- The Trap: You cannot grip Jelly with a Robot Arm. The harder you squeeze (strict parsing), the more it slips.

The "Happy Path" Fallacy

- Demo = P(intent) × P(tool) × P(args) = 1 (fake)
- Production = statistical entropy
- Need agents robust to partial success

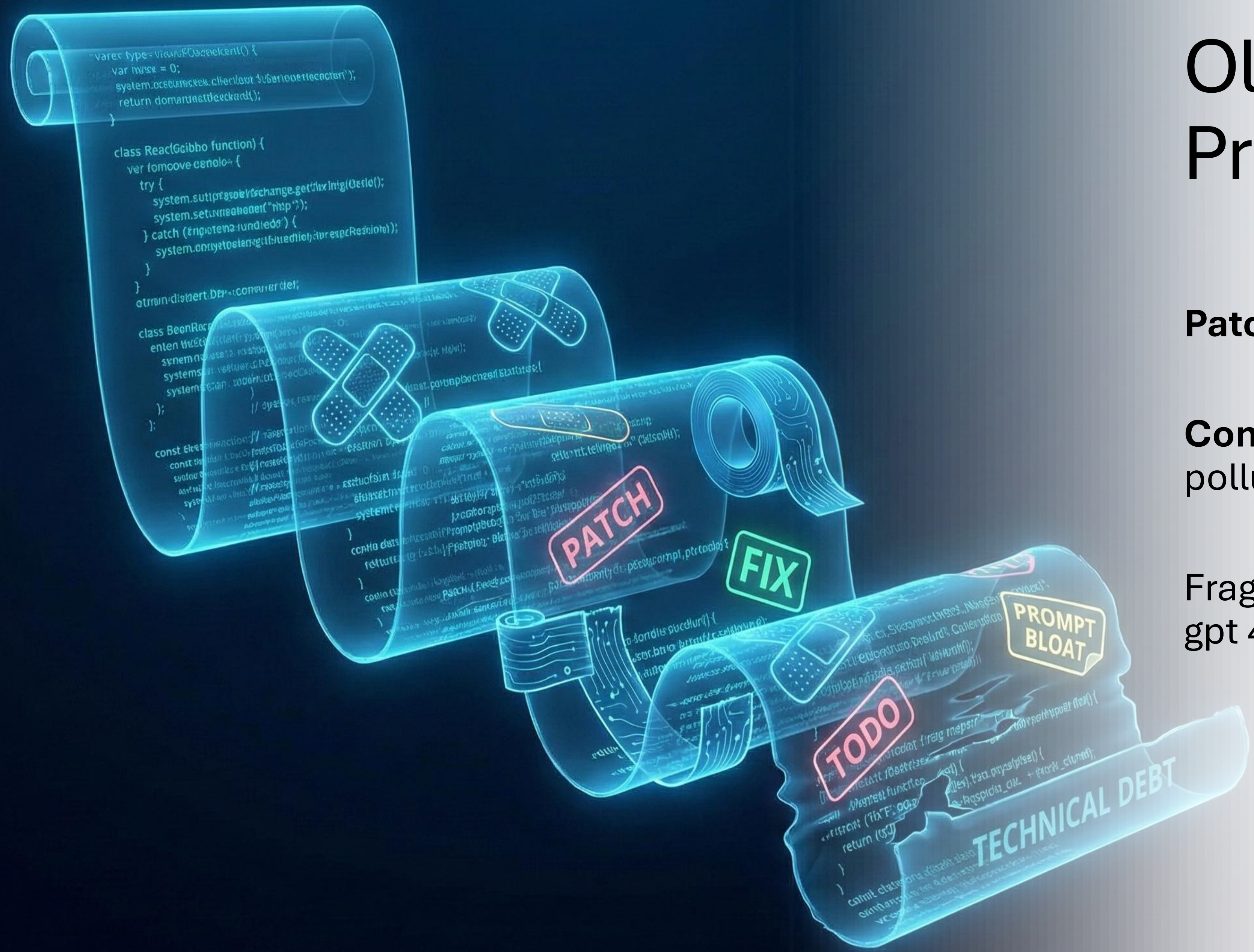
Issues Faced in Production

Semantic Loops: The "Spinning Wheel of Death"

Context Drift: Forgetting the goal

Tool Misuse: Hallucinated capabilities

Cost: Burning tokens on failure



Old Ways: Defensive Prompting

Patch prompts to handle edge cases

Consequence: prompt bloat & context pollution

Fragile across model versions (gpt 4 Vs gpt 4o)

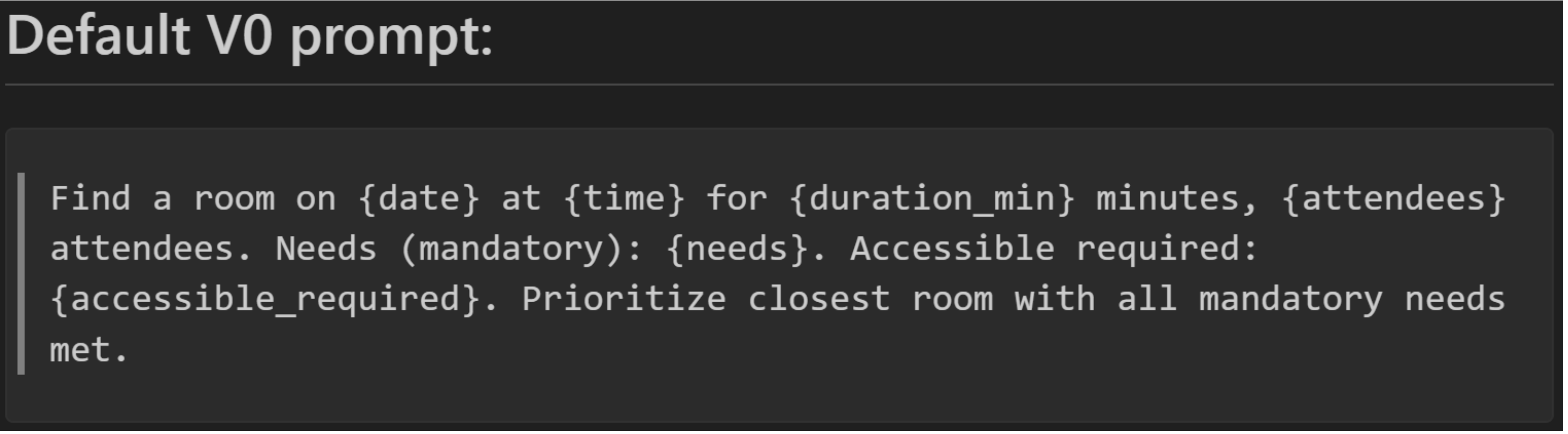
The Paradigm Shift

What is Reinforcement Learning?

Reinforcement Learning: A Compiler for Behavior

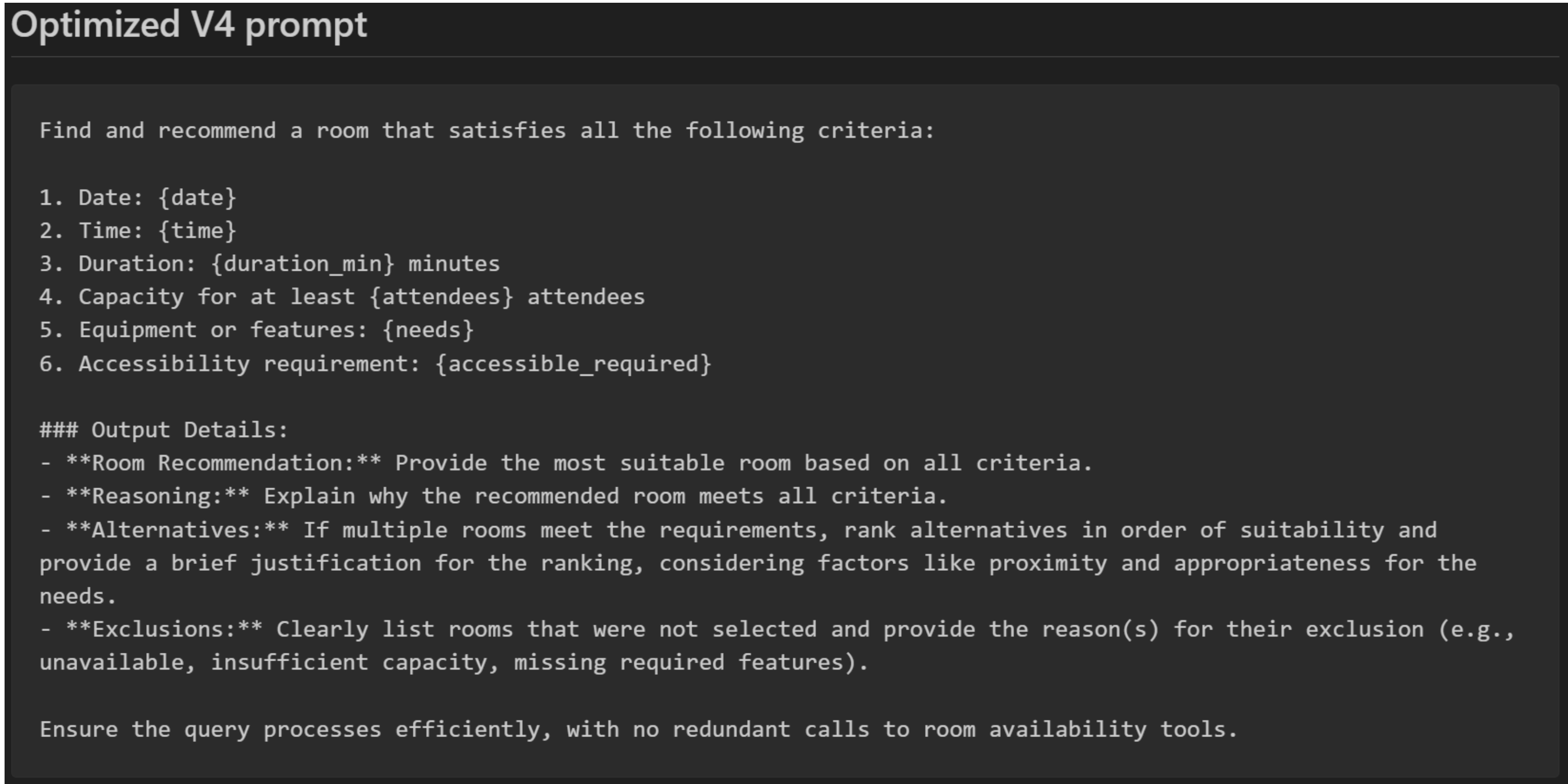
- **Action:** The Dog sits (The Agent tries a task).
- **Reward:** You give a treat (Score = 1.0) or ignore it (Score = 0.0).
- **Policy:** The dog learns: "Sitting = Treats".

Reward Optimization



Default V0 prompt:

> Find a room on {date} at {time} for {duration_min} minutes, {attendees} attendees. Needs (mandatory): {needs}. Accessible required: {accessible_required}. Prioritize closest room with all mandatory needs met.



Optimized V4 prompt

```
Find and recommend a room that satisfies all the following criteria:

1. Date: {date}
2. Time: {time}
3. Duration: {duration_min} minutes
4. Capacity for at least {attendees} attendees
5. Equipment or features: {needs}
6. Accessibility requirement: {accessible_required}

### Output Details:
- **Room Recommendation:** Provide the most suitable room based on all criteria.
- **Reasoning:** Explain why the recommended room meets all criteria.
- **Alternatives:** If multiple rooms meet the requirements, rank alternatives in order of suitability and
provide a brief justification for the ranking, considering factors like proximity and appropriateness for the
needs.
- **Exclusions:** Clearly list rooms that were not selected and provide the reason(s) for their exclusion (e.g.,
unavailable, insufficient capacity, missing required features).

Ensure the query processes efficiently, with no redundant calls to room availability tools.
```

AGENT LIGHTNING

What is Agent Lightning?

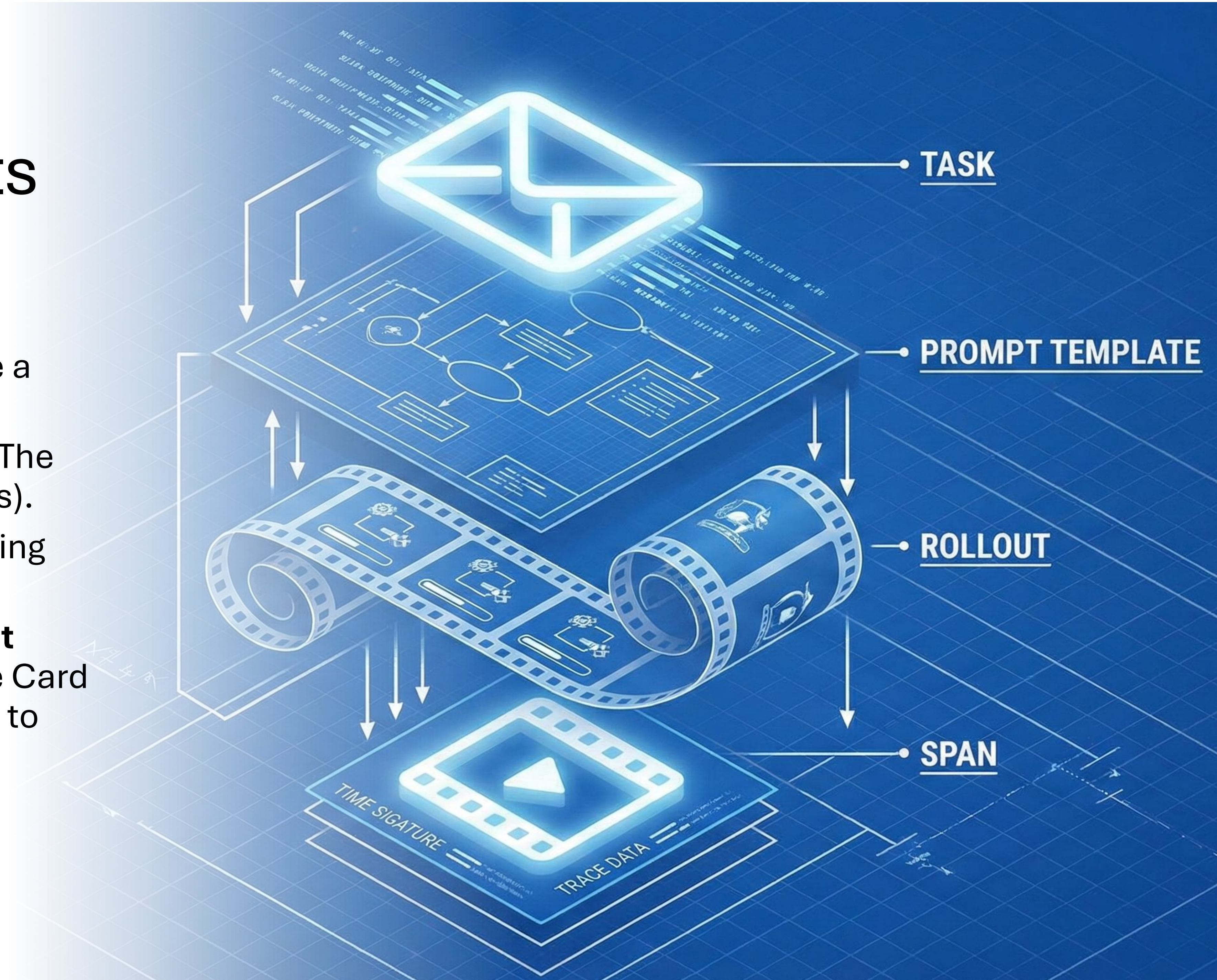
Training-Agent Disaggregation (Decouples Execution from Optimization)

Separates the **Student** (The Agent) from the **Teacher** (The Optimization Algorithm).

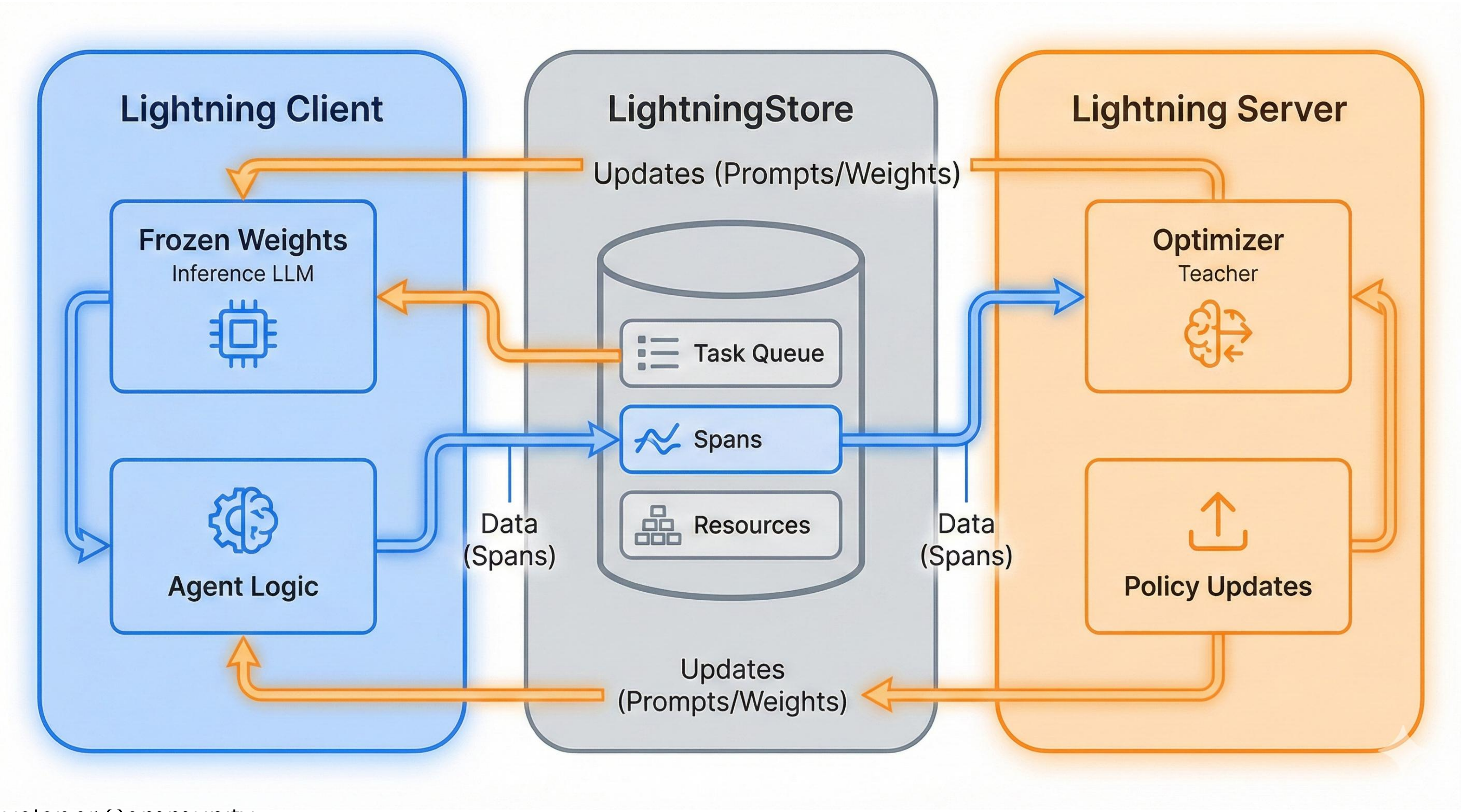
Apply Reinforcement Learning to *any* agent architecture, regardless of the underlying LLM or tools.

Core Concepts

- **Task:** The Order ("Bake a Cake").
- **Rollout:** The Attempt (The actual cooking process).
- **Span:** The Step (Cracking eggs, mixing flour).
- **The Resource (Prompt Template):** The Recipe Card (The resource we want to optimize).



Training-Agent Disaggregation



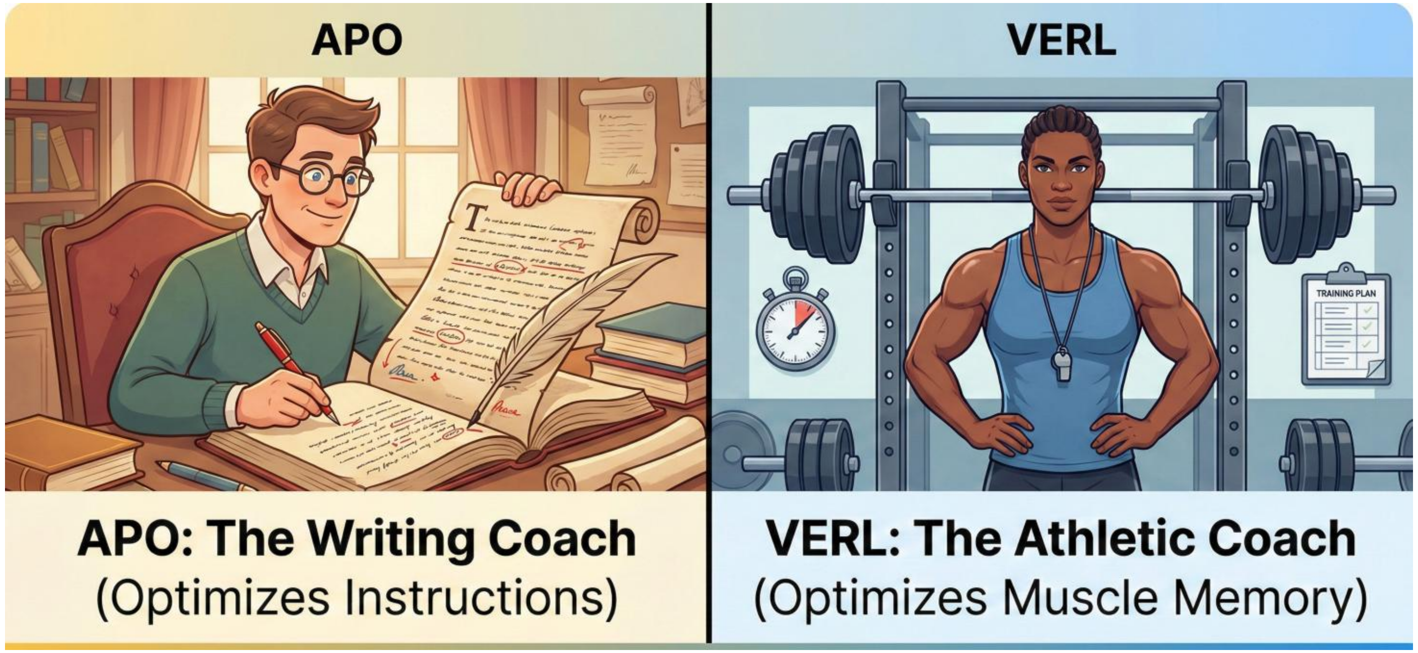
What are Algorithms?

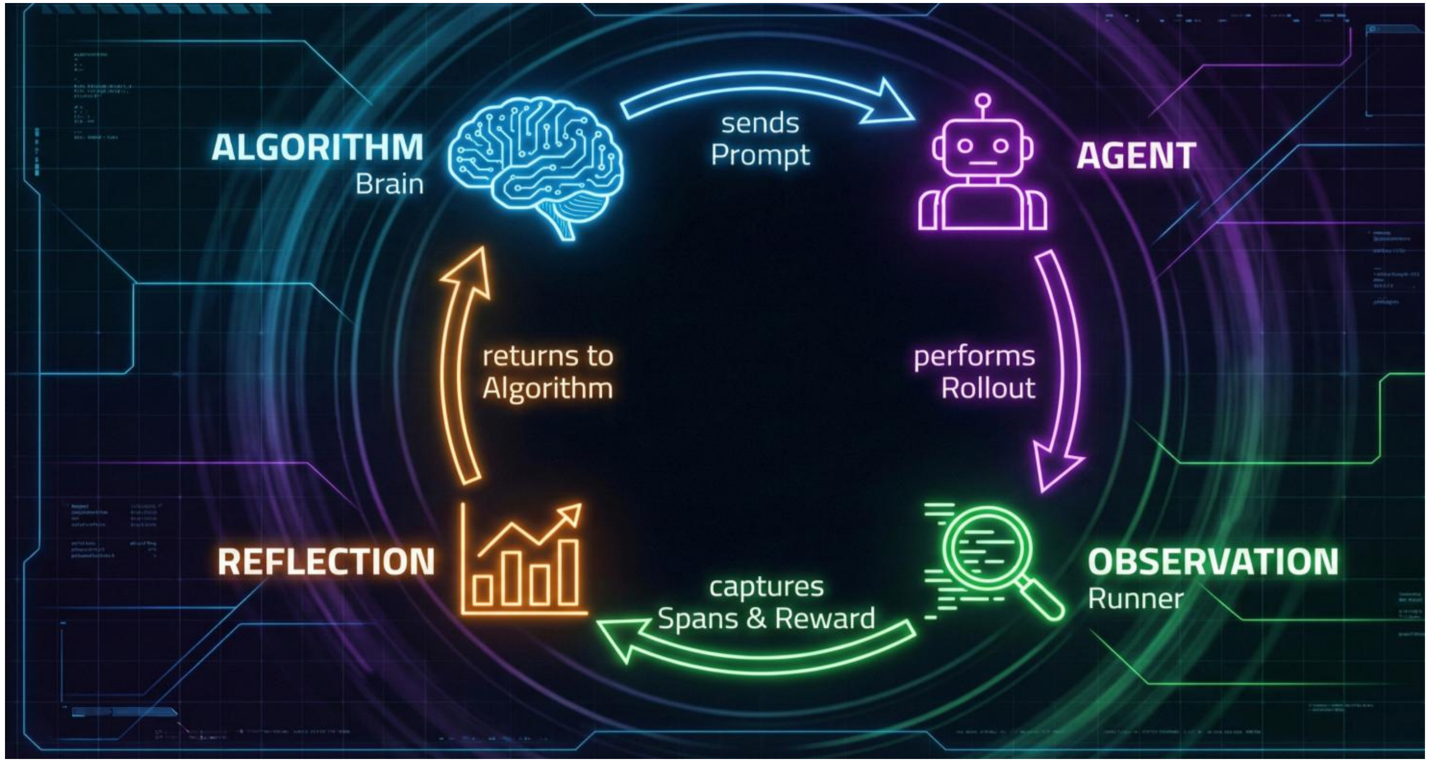
APO:

- Iterative instruction refinement without retraining weights.
- Uses textual gradients and beam search.
- Rapid prototyping and task-specific steering.

VERL:

- High-performance Reinforcement Learning at scale.
- Designed for distributed training of massive models.
- Post-training models for complex reasoning or alignment.





The Agent Algorithm Loop

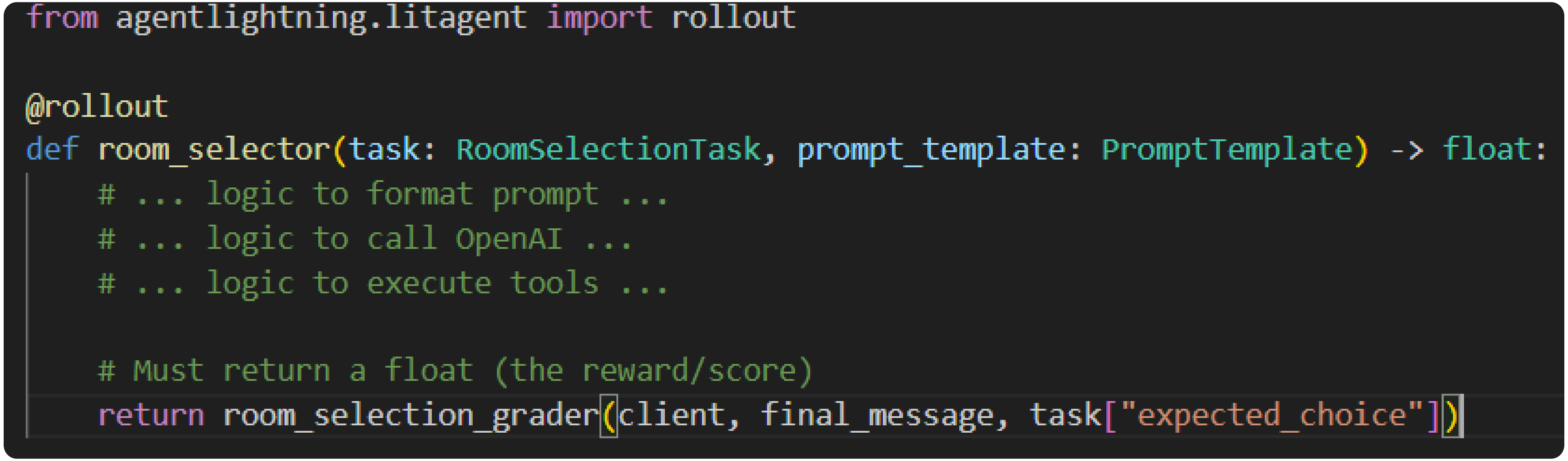
- **Algorithm (Brain):** Sends the current Prompt.
- **Agent:** Executes the "Rollout".
- **Observation:** Captures "Spans" (Trace data) & "Reward" (Score).
- **Reflection:** The Algorithm critiques the traces and updates the Prompt.

Showcase: The Room Selector

"Task: Find the Best Room."

- **Input:** "Find a room for 4 people at 10 AM with a whiteboard".
- **Action:** The agent queries a database tool.
- **Reward:** We score it 0.0 to 1.0. If it picks a room without a whiteboard? 0.0. A perfect match? 1.0.
- **Self-Healing:** Through RL, the agent learns *on its own* to check constraints before answering, without us writing a 50-page defensive prompt.

Demo



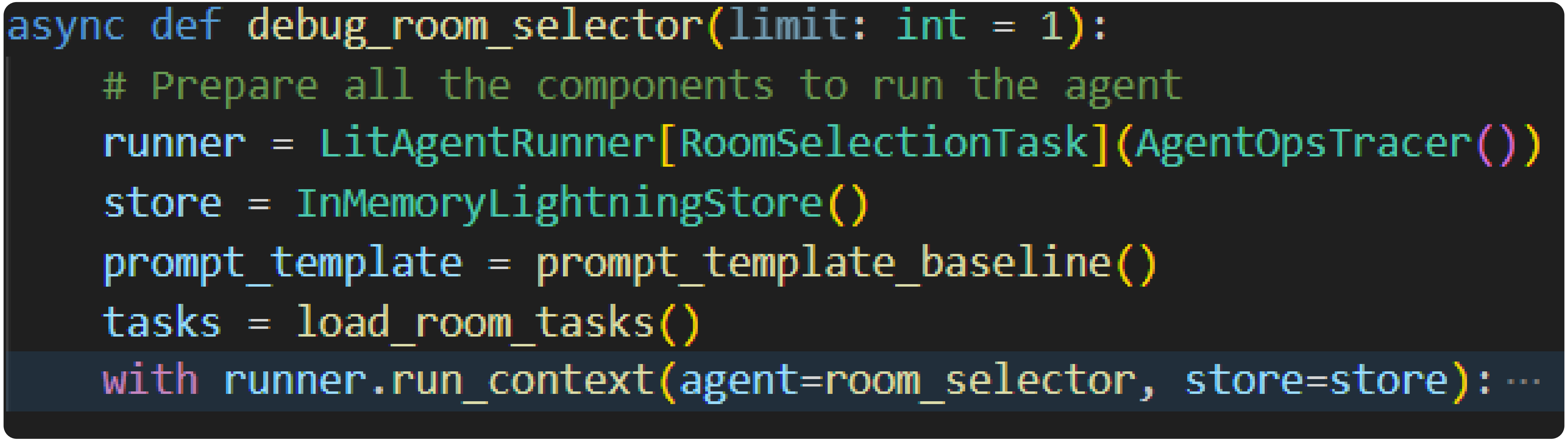
```python
from agentlightning.litagent import rollout

@rollout
def room_selector(task: RoomSelectionTask, prompt_template: PromptTemplate) -> float:
    # ... logic to format prompt ...
    # ... logic to call OpenAI ...
    # ... logic to execute tools ...

    # Must return a float (the reward/score)
    return room_selection_grader(client, final_message, task["expected_choice"])
```

What Is An Agent? (Code)

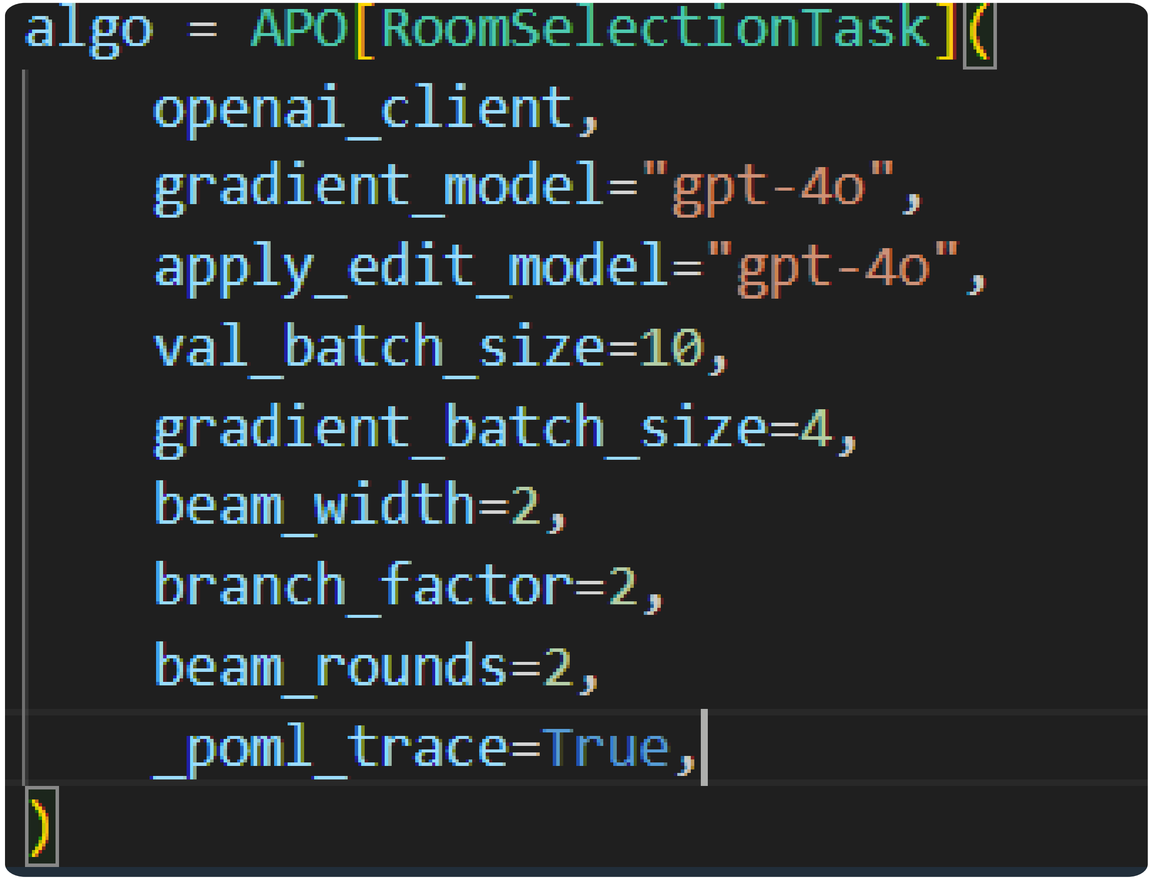
- Python function with @rollout decorator
- Encapsulates task logic, LLM calls, tool calls
- Returns reward + spans



```
async def debug_room_selector(limit: int = 1):
    # Prepare all the components to run the agent
    runner = LitAgentRunner[RoomSelectionTask](AgentOpsTracer())
    store = InMemoryLightningStore()
    prompt_template = prompt_template_baseline()
    tasks = load_room_tasks()
    with runner.run_context(agent=room_selector, store=store):···
```

What is a Runner and tracer?

- **Runner:** The engine. It manages the lifecycle of a single attempt. (executes rollouts, manages context)
- **Tracer:** The observer. It records every step ("Span") so the Algorithm can "see" what went wrong.
- **Spans** → dataset for learning

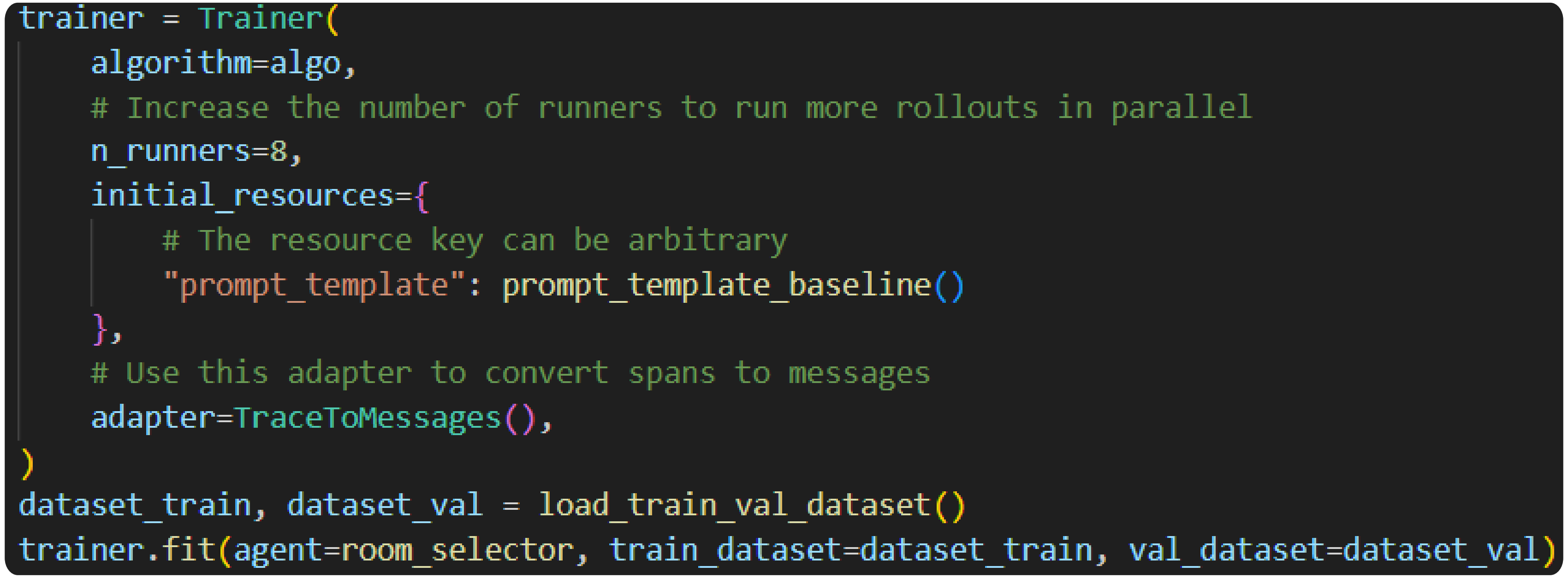


```
algo = APO[RoomSelectionTask](
    openai_client,
    gradient_model="gpt-4o",
    apply_edit_model="gpt-4o",
    val_batch_size=10,
    gradient_batch_size=4,
    beam_width=2,
    branch_factor=2,
    beam_rounds=2,
    _poml_trace=True,
)
```

What is an Algorithm?

- The Algorithm is the "brain" responsible for learning.

  - APO: LLM critiques & rewrites prompt templates
  - VERL: RL updates to policy/weights
  - Both ingest spans + rewards



```
trainer = Trainer(
    algorithm=algo,
    # Increase the number of runners to run more rollouts in parallel
    n_runners=8,
    initial_resources={
        # The resource key can be arbitrary
        "prompt_template": prompt_template_baseline()
    },
    # Use this adapter to convert spans to messages
    adapter=TraceToMessages(),
)
dataset_train, dataset_val = load_train_val_dataset()
trainer.fit(agent=room_selector, train_dataset=dataset_train, val_dataset=dataset_val)
```

What is a Trainer?

- Orchestrates algorithm ↔ runner ↔ dataset
- Maintains prompt versions, metrics, checkpoints
- Schedules training & evaluation rollouts

Summary of flow

Trainer starts.

Trainer asks **Algorithm** for the current Prompt.

Trainer gives Prompt and Task to **Runner**.

Runner starts **Tracer** and executes **Agent**.

Agent talks to LLM, uses tools, and returns Reward.

Tracer records the conversation history (Spans).

Algorithm reads Spans + Reward, critiques the errors, and writes a *new* Prompt.

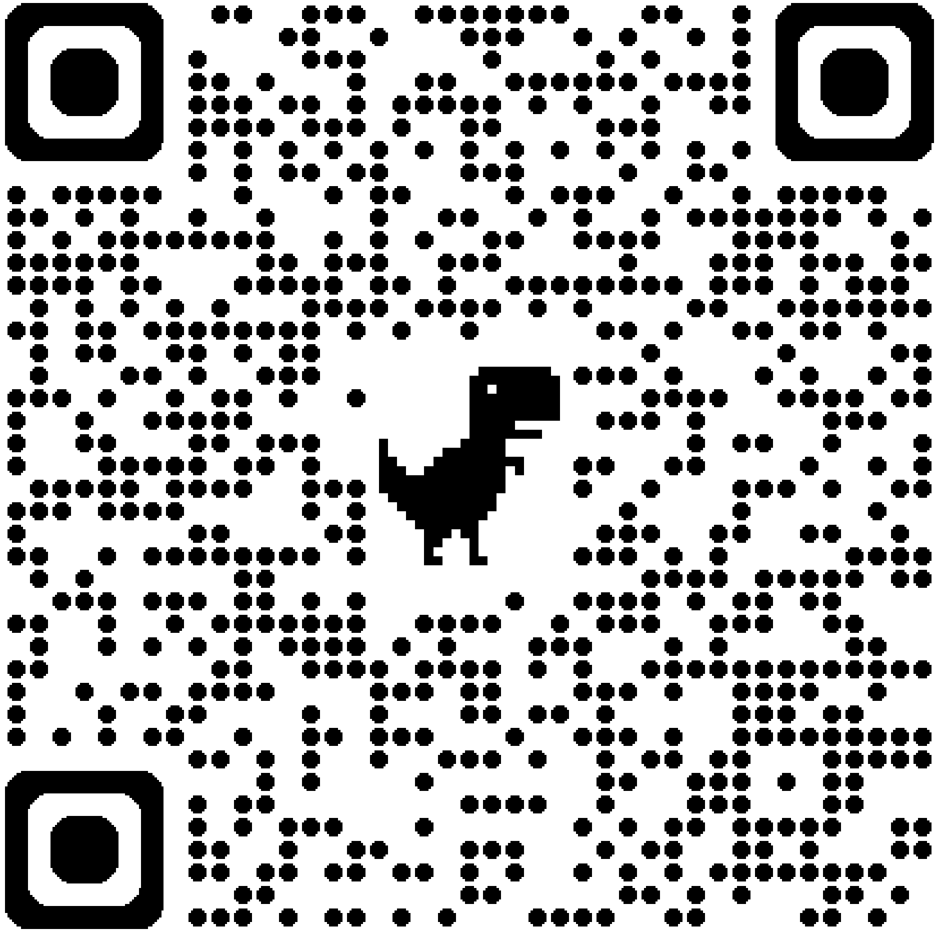
Repeat.

Practical Checklist

- Use APO first; escalate to VERL if needed
- Monitor token & cost metrics
- Test with a smaller question set

Q&A

Scan the QR code



Open for Q&A

Thank you!!

#Azure Developer Community