INFERENCE-TIME DISTILLATION: COST-EFFICIENT AGENTS WITHOUT FINE-TUNING OR MANUAL PROMPT ENGINEERING

Vishnu Sarukkai[*], Asanshay Gupta[*], James Hong[†], Michaël Gharbi[†], Kayvon Fatahalian[*]

Deploying LLM agents at scale typically requires choosing between quality and cost. Existing cost-reduction approaches fail to preserve *agility*: the ability to iterate rapidly without human time bottlenecks. Prompt engineering is brittle and slows iteration, while fine-tuning requires multi-day training and commitment to fixed designs; both are impractical for iterative workflows and time-sensitive batch jobs. We demonstrate that established inference-time techniques—dynamic in-context learning and self-consistency cascades—can be leveraged to shift the cost-accuracy Pareto frontier while preserving agility. Practitioners run the teacher on a small task subset to collect demonstrations, then immediately deploy a cheaper student on the remainder. At each step, the system retrieves relevant teacher demonstrations as in-context examples. When multiple student samples agree, we proceed; when they diverge, we fall back to the teacher. This requires no prompt engineering or training. On ALFWorld, we match teacher accuracy at 2.5× lower cost ($0.059 → $0.024 per episode). On AppWorld, we achieve 3.5× cost reduction while recovering 79% of teacher accuracy. Our empirical analyses provide guidance on key design choices: teacher database size, demonstration set size, retrieval strategy, and cascade thresholds. These analyses highlight inference-time levers for navigating cost-performance tradeoffs without sacrificing human development speed.

1 INTRODUCTION

The world has an abundance of ideas for agentic solutions to enable new software features or automate tasks. However, practitioners face a fundamental tension: developing agents requires *agility*—the ability to iterate rapidly without human time bottlenecks—but deploying them at scale requires cost efficiency. Traditional approaches force an unacceptable tradeoff. Prompt engineering can improve model performance, but requires laborious trial-and-error that is brittle, unreliable, and slows iteration cycles (Khattab et al., 2023). Fine-tuning reliably improves the cost-performance frontier, but has high infrastructure overhead and can require days of training, making it impractical when jobs need immediate execution.

We define *agility* as minimizing human development time: the ability to iterate on agent designs and deploy at scale without manual prompt engineering, multi-day training cycles, or trial-and-error tuning. This is critical in two scenarios: (1) *iterative development workflows*, where organizations refine agent architectures based on user feedback while serving production traffic; and (2) *time-sensitive batch jobs*, such as processing large-scale datasets, where running a teacher model on a small subset to bootstrap a cheaper student for the remaining tasks must happen immediately, without waiting for fine-tuning cycles or iterative prompt refinement.

We demonstrate that the established inference-time mechanisms of dynamic in-context learning (Liu et al., 2021; Zhou et al., 2024; Sarukkai et al., 2025) and self-consistency cascades (Wang et al., 2022b; Yue et al., 2024) can be combined to achieve distillation-like cost reductions while preserving agility. The key is *how* these techniques are applied. When running a large batch of tasks, practitioners run the teacher model on a small representative subset (hundreds of episodes) to collect demonstrations, then immediately deploy a cheaper student on the remainder. At each step, the system retrieves relevant teacher demonstrations and inserts them into the student's context. When retrieved examples enable confident student predictions (detected via self-consistency across multiple samples), the student proceeds; otherwise, the system falls back to the teacher. This requires no prompt engineering, model training, or manual tuning.

[*]Stanford University
[†]Reve

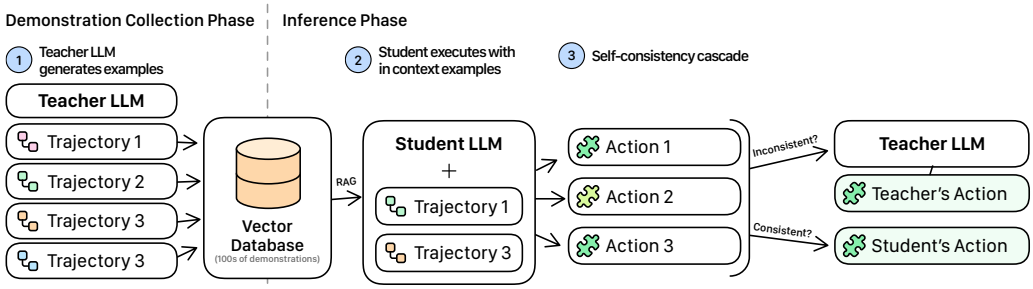

Figure 1: **Our approach combines dynamic in-context learning with self-consistency cascades.** Teacher demonstrations are collected once and indexed. At deployment, relevant examples are retrieved at each step to guide a cheaper student model. Self-consistency across multiple student samples determines when to proceed vs. defer to the teacher.

Across ALFWorld (Shridhar et al., 2020) and AppWorld (Trivedi et al., 2024) benchmarks, our approach shifts the cost-accuracy Pareto frontier. On ALFWorld, we achieve 2.5× cost reduction while matching teacher-level accuracy ($0.059 → $0.024 per episode). On AppWorld, a more complex benchmark requiring multi-step API workflows, we achieve 3.5× cost reduction while recovering 79% of teacher accuracy. Crucially, our empirical analyses provide practitioners with guidance on key design choices: teacher database size, demonstration set size, demonstration content, retrieval strategy, and cascade thresholds. These levers enable practitioners to reliably navigate cost-performance tradeoffs without sacrificing human development speed.

2 Problem Statement

We address cost-effective agent deployment in settings requiring rapid iteration and immediate execution. We focus on multi-step interactive tasks where an agent must take multiple actions, each time receiving feedback from its environment to accomplish a goal (e.g., API call sequences, structured data workflows).

Formally, consider an interactive environment $\mathcal{E}$ in which an agent receives a task specification (e, g) (consisting of an environment instance e and task goal g), observes states o_t, and executes actions a_t over time steps $t = 1, \ldots, T$. Each episode produces a trajectory $\tau = \{(o_t, a_t)\}_{t=1}^{T}$, terminating in success or failure.

We assume access to a high-capacity *teacher* M_{t} with per-token cost c_{t} and a lower-cost *student* M_{s} with cost $c_{\text{s}} \ll c_{\text{t}}$. The teacher is generally more capable. We can collect teacher demonstrations offline from tasks $\mathcal{T}_{\text{demo}}$ and have inference access to both models.

Constraints. We operate under constraints ruling out traditional cost-reduction approaches: (1) no model training—the policy π uses only inference-time mechanisms (retrieval, prompting, routing), and (2) immediate deployment—the system must be ready after demonstration collection, without training cycles. These constraints arise in iterative development workflows (where quick redeployment as agent designs evolve is essential) and time-sensitive batch jobs.

Objective. We seek a policy maximizing success on test tasks $\mathcal{T}_{\text{test}}$ (with $|\mathcal{T}_{\text{test}}| \gg |\mathcal{T}_{\text{demo}}|$) while minimizing cost:

$$\pi^* = \arg \max_{\pi \in \Pi_{\text{frozen}}} \mathbb{E}_{(e,g)} \left[S(\pi, e, g) - \lambda \cdot C(\pi, e, g) \right]$$

where Π_{frozen} denotes policies using only frozen models, $\mathrm{S}(\pi, e, g) \in \{0, 1\}$ indicates success, $\mathrm{C}(\pi, e, g)$ is total inference cost, and $\lambda > 0$ controls the tradeoff.

The challenge: the student alone has low cost but poor accuracy; the teacher has high accuracy but prohibitive cost at scale. Traditional distillation via fine-tuning violates our constraints (requires training, takes days, breaks when architectures change). We must improve student capability using only inference-time mechanisms.

Algorithm 1 ReAct-Style Agent Loop

function AGENT($g, \mathcal{D}, \mathcal{E}, T$)
  $C_p \leftarrow \texttt{Retrieve}(\mathcal{D}, [g])$ ▷ Retrieve for planning
  $p \leftarrow \texttt{LLM}_{\text{plan}}(g, C_p)$ ▷ Generate high-level plan
  $o_1 \leftarrow \mathcal{E}.\texttt{reset}(g)$
  for $t = 1$ to T **do**
    $C_t \leftarrow \texttt{Retrieve}(\mathcal{D}, [g, p, o_t])$ ▷ Step-level retrieval
    $r_t, a_t \leftarrow \texttt{LLM}_{\text{react}}(g, p, o_t, C_t)$ ▷ Reason and act
    o_{t+1}, done $\leftarrow \mathcal{E}.\texttt{step}(a_t)$
    if done **then return** trajectory τ
  return trajectory τ

3 METHODS

Our approach collects teacher demonstrations once, then uses dynamic in-context learning and self-consistency cascades to route between student and teacher models at each agent decision step. Figure 1 illustrates the overall logic of our algorithm.

Demonstration collection phase: We execute the teacher model M_{t} on tasks from $\mathcal{T}_{\text{demo}}$ to collect a database $\mathcal{D}$ of complete trajectories, including observations, reasoning traces, and actions. This database is indexed using dense embeddings to enable fast similarity-based retrieval. Demonstration data collection is a one-time upfront cost; since we assume operating at scale with $|\mathcal{T}_{\text{test}}| \gg |\mathcal{T}_{\text{demo}}|$, this upfront cost is negligible when amortized over many test tasks. For readers interested in the exact crossover point where inference savings exceed demonstration costs, we provide a detailed analysis in Appendix C.3.4.

Inference phase: For each new task $(e, g) \in \mathcal{T}_{\text{test}}$, the student model M_{s} executes a standard ReAct-style agent loop (Section 3.1). At each decision step t, the system retrieves the most relevant teacher examples from $\mathcal{D}$ based on the current goal, plan, and observation, and inserts them into the student's prompt (Section 3.3). The student then samples multiple candidate actions under the same retrieved context. If all samples agree, the student's action is executed; if they diverge, the system defers to the teacher for that step (Section 3.4). This loop continues until the episode terminates, with total cost determined by the number of student queries (cheap) and teacher deferrals (expensive).

3.1 BACKGROUND: REACT-STYLE AGENT ARCHITECTURE

The design of our agents follows the ReAct framework (Yao et al., 2023), which interleaves planning, reasoning, and acting in a structured loop (Algorithm 1). At the start of an episode, the agent generates a high-level plan p for the goal g. At each timestep t, it observes the current state o_t, produces intermediate reasoning r_t, and selects an action a_t to execute in the environment $\mathcal{E}$. Following recent best practices (Kagaya et al., 2024; Zhou et al., 2024; Sarukkai et al., 2025), we perform *dynamic retrieval*—retrieving distinct exemplars for each step based on the current context rather than reusing a static prompt throughout the episode. We adopt this architecture as our foundation, building on recent findings that dynamic per-step retrieval improves multi-step agent performance.

The action a_t at each step is obtained by querying a language model conditioned on the goal g, plan p, current observation o_t, and retrieved context C_t from the teacher database. Our approach builds on this by using teacher-generated trajectories as the retrieval source, enabling cost reduction through a cheaper student model while maintaining the flexibility of this established architecture.

3.2 TEACHER DEMONSTRATION DATABASE

We first construct a database $\mathcal{D} = \{\tau_i^{(\text{t})}\}_{i=1}^{|\mathcal{T}_{\text{demo}}|}$ of teacher trajectories using a high-performing model M_{t} (e.g., `Claude Sonnet 4.5`). For each demonstration task $(e_i, g_i) \in \mathcal{T}_{\text{demo}}$, we run the agent using the teacher LLM in the environment e_i to produce a complete trajectory:

$$\tau_i^{(\text{t})} = \{g_i, p_i, (o_t^{(i)}, r_t^{(i)}, a_t^{(i)})_{t=1}^{T_i}\},$$

where p_i is the teacher's plan, and each step includes the observation o_t, reasoning trace r_t, and action a_t.

To enable retrieval, we compute dense embeddings using `MiniLM-L6-v2` (Wang et al., 2020). For each trajectory, we separately embed the goal Embed(g_i), plan Embed(p_i), and at each step t, the reasoning Embed$(r_t^{(i)})$. These embeddings are indexed for fast similarity-based lookup during test-time deployment. Unlike prior self-improving agents that expand $\mathcal{D}$ online (Sarukkai et al., 2025), our setup keeps $\mathcal{D}$ fixed after initial collection to cleanly separate one-time teacher cost from ongoing student deployment cost.

3.3 'DISTILLATION' VIA DYNAMIC IN-CONTEXT LEARNING

At test time, a smaller frozen model M_{s} (e.g., GPT-4.1-mini) interacts with new tasks $(e, g) \in \mathcal{T}_{\text{test}}$ using in-context exemplars retrieved from $\mathcal{D}$. The student reuses teacher reasoning traces as dynamic contextual guidance for each step, transferring behavior without weight updates.

Multi-key retrieval. Inspired by the retrieval system in Sarukkai et al. (2025), our retrieval operates on two levels:

- **Trajectory-level (planning):** Before execution, retrieve the k most similar teacher goals based on cosine similarity. Extract the high-level plans $\{p_j\}_{j=1}^{k}$ from these trajectories and provide them as exemplars when prompting the student to generate its own plan p.
- **Step-level (acting):** At each step t, retrieve the top-k most similar teacher steps using averaged cosine similarity across goal, plan, and reasoning. Return a window of neighboring steps $\{(o_j^*, r_j^*, a_j^*)\}$ from each retrieved trajectory—providing only the most relevant window of the trajectory.

This multi-key approach ensures retrieved examples match the current context across multiple relevant dimensions rather than relying on a single composite embedding.

These exemplars are inserted directly into the prompt of M_{s}:

$$(r_t, a_t) \sim M_{\text{s}}(\cdot \mid g, p, o_t, \text{Retrieve}(\mathcal{D}; g, p, r_t)) .$$

Because retrieval occurs dynamically at each step, the student continually adapts to new contexts using only teacher demonstrations.

Connection to reasoning distillation. Since retrieved trajectories contain both actions *and* reasoning traces r_t from the teacher, our approach can be viewed as an in-context variant of reasoning distillation (Hsieh et al., 2023), which shows that distilling intermediate reasoning (not just final actions) improves student generalization. By including r_t in the retrieved context, we enable the student to imitate not only *what* the teacher did, but *why*—providing richer signal for behavioral transfer.

Because distillation occurs via in-context learning rather than weight updates, it requires far fewer demonstration examples to effectively transfer knowledge. In our experiments we show this approach is effective using only a few hundred examples. This has important practical implications: (1) we can safely assume $|\mathcal{T}_{\text{demo}}| \ll |\mathcal{T}_{\text{test}}|$, making up-front demonstration costs negligible when amortized over large test sets, and (2) the cost of storing and retrieving from a database of a few hundred demonstrations is negligible. This synergy between sample-efficient in-context learning and cost-effective deployment at scale is a key advantage of our approach.

3.4 SELF-CONSISTENCY BASED DEFERRAL

While dynamic retrieval improves the student's ability to perform more tasks, there may remain situations where retrieved exemplars fail to effectively guide the student LLM. To detect when the student is likely to fail despite having access to retrieved teacher examples, we employ a lightweight introspection mechanism based on *self-consistency* (Wang et al., 2022b). The key insight is that when in-context learning succeeds, the student's understanding of the correct action should be stable across multiple samples. Conversely, when the retrieved examples provide insufficient guidance, there may be uncertainty in the student's output, manifesting as disagreement among samples.

Why self-consistency rather than output uncertainty? LLM cascade methods (Chen et al., 2023b; Schuster et al., 2022) sometimes use model confidence scores (e.g., token probabilities) to trigger deferral. In chain-of-thought settings, where the LLM first generates reasoning tokens before action tokens, computing the uncertainty in the generated answer either requires marginalizing out the reasoning path (Wang et al., 2022b), or using a sampling-based approach to model the answer

Algorithm 2 Dynamic In-Context Learning with Cascade

function STEP($o_t, g, p, \mathcal{D}, M_\text{s}, M_\text{t}, N$)
  $C_t \leftarrow$ Retrieve($\mathcal{D}, [g, p, o_t]$) ▷ Retrieve examples
  $\{a_t^{(i)}\}_{i=1}^N \leftarrow$ Sample($M_\text{s}, g, p, o_t, C_t$) ▷ Sample student
  consistent $\leftarrow$ Consistent($\{a_t^{(i)}\}_{i=1}^N$) ▷ Check agreement
  if consistent **then**
    $a_t \leftarrow a_t^{(1)}$ ▷ Use student action
  else
    $a_t \leftarrow M_\text{t}(g, p, o_t)$ ▷ Defer to teacher
  return a_t

distribution (Yue et al., 2024), which naturally allows for the comparison of answers of different lengths. Paired with an LLM verifier, we find the sampling-based approach to be a simple and effective way to implement cascades in agentic settings with large action spaces.

Implementation. At each step t, the student samples $N = 3$ candidate actions conditioned on retrieved exemplars C_t:

$$\{a_t^{(i)}\}_{i=1}^N \sim M_\text{s}(\cdot \mid g, p, o_t, C_t).$$

By default, agreement requires all sampled actions to be identical:

$$\text{Consistent}(\{a_t^{(i)}\}_{i=1}^N) = \mathbb{1}\left[\forall\, i, j,\ a_t^{(i)} = a_t^{(j)}\right].$$

If consistent, we execute the student's action; otherwise, we defer to the teacher:

$$a_t = \begin{cases} a_t^{(1)} & \text{if Consistent} = 1, \\ M_\text{t}(g, p, o_t) & \text{otherwise.} \end{cases}$$

For tasks with multiple valid formulations (e.g., API orchestration where different code snippets implement equivalent behavior), we use a *soft-equivalence* variant: an auxiliary verifier assesses whether candidate actions are semantically equivalent despite syntactic differences.

Together, these mechanisms trigger teacher queries only when the student's in-context reasoning fails to produce consistent outputs.

Cost accounting. A critical consideration is that sampling N student outputs incurs N times the student output token cost at each step. However, since $c_\text{s} \ll c_\text{t}$ (typically $10\times$ cheaper), and costs are dominated by input tokens in the agentic setting, this overhead remains small compared to a single teacher query.

4 EXPERIMENTS

We structure our evaluation around three goals: (1) evaluate cost-accuracy trade-offs relative to baselines; (2) isolate contributions of in-context learning vs. adaptive routing; and (3) provide practitioners with guidance on key implementation choices.

4.1 BENCHMARK SELECTION

Since agents are commonly used to automate rote workflows—tasks that benefit from rapid prototyping and deployment—we chose benchmarks where task instances share structural patterns enabling cross-task learning, while exhibiting sufficient variety to require genuine adaptation. Please see Appendix D for further details on benchmarks.

AppWorld (Trivedi et al., 2024) features workflow automation coding tasks requiring API composition (Gmail, Contacts, Calendar, etc.), user context interpretation, and multi-step state management. Tasks are evaluated via automated unit tests. We use 147 tasks from `train` and `val` as $\mathcal{T}_\text{demo}$, evaluating on 168 tasks in `test-normal`.

ALFWorld (Shridhar et al., 2020) provides a controlled diagnostic on well-understood planning problems with discrete action spaces. We collect 500 teacher demonstrations from `train` as $\mathcal{T}_\text{demo}$ and evaluate on 134 tasks in `eval-out-of-distribution`.

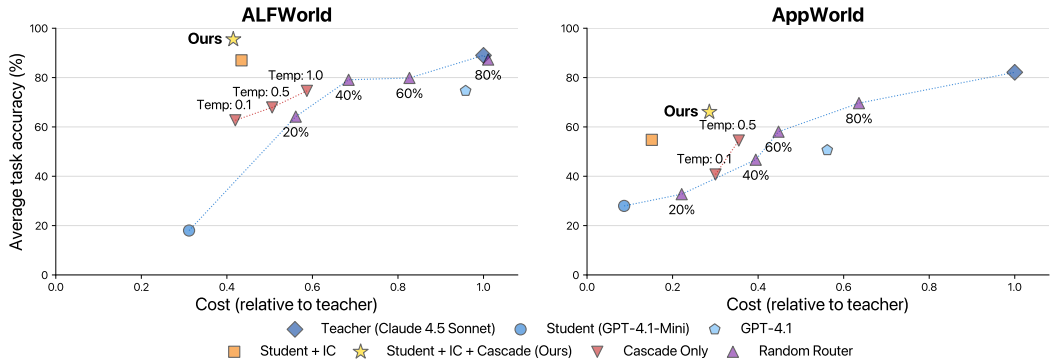

Figure 2: **Combining in-context learning with cascades optimizes cost-accuracy tradeoffs.** Our IC + Cascade method breaks the Pareto frontier, performing better than the teacher on ALFWorld and significantly above others at similar cost on both benchmarks.

We seek to reduce inference costs at scale, where $|\mathcal{T}_{\text{test}}| \gg |\mathcal{T}_{\text{demo}}|$. This mirrors deployment scenarios like batch processing where demonstration costs amortize over many instances. We only count inference costs on test sets, not demonstration collection costs, approximating execution at scale. Amortization analysis is in Appendix C.3.4.

4.2 Teacher-student setup

We use `Claude Sonnet 4.5` as teacher and `GPT-4.1-mini` as student. On AppWorld, `GPT-4.1-mini` serves as an LLM verifier for action equivalence. We validate generalization with `Llama-3.3-70B` as student to confirm our approach is not API-specific.

Core algorithmic configurations:

- **Teacher**: Upper bound on accuracy; ceiling cost.
- **Student (ZS)**: Zero-shot student; lower bound on cost, baseline accuracy.
- **Student (IC)**: Student with retrieved teacher exemplars (tests dynamic in-context learning alone).
- **Student (IC+Cascade)**: Full system (our primary method).

Alternative routing strategies:

- **Random Mix**: Query teacher for fixed fraction of steps (0.2, 0.4, 0.6, 0.8). Controls for whether routing value comes from adaptive gating vs. simply mixing teacher capacity.
- **Student (Cascade only)**: Self-consistency gating without exemplars, varying temperature (0.1, 0.5, 1.0). Isolates adaptive routing value without in-context improvement.

Ablations:

- **Retrieval strategies**: Compare retrieval approaches (number of examples k, retrieval keys, similarity metrics).
- **Demonstration set size**: Vary $|\mathcal{T}_{\text{demo}}|$ to characterize how many demonstrations are needed.
- **In-context mechanism analysis**: Investigate relationship between retrieved example content and student behavior replication.

4.3 Metrics

We measure task success rate as defined by each benchmark. For costs, we track all LLM calls: (i) student queries, (ii) teacher queries (when cascading), and (iii) LLM verifier calls (AppWorld only), recording input/output tokens and computing cost using current API pricing. We report costs normalized by Teacher baseline; see Appendix C for details.

5 RESULTS

5.1 COMBINING DYNAMIC IN-CONTEXT LEARNING WITH CASCADES IMPROVES COST-ACCURACY TRADEOFFS

Figure 2 demonstrates that dynamic in-context learning from teacher examples and self-consistency cascades are complementary techniques that jointly optimize the cost-accuracy frontier.

Dynamic in-context learning (alone) bridges the teacher-student gap. Figure 2 shows that Student (IC) establishes a new Pareto-optimal point between the zero-shot student and teacher baselines, outperforming Student (Cascade only) configurations at equivalent cost levels. On ALFWorld, Student (IC) achieves 0.87 accuracy—97% of teacher accuracy (0.89) at 43% of the cost ($0.026 vs. $0.059), compared to 0.18 zero-shot. On AppWorld, dynamic in-context learning improves accuracy from 0.28 to 0.55 while remaining at 15% of teacher cost ($0.089 vs. $0.589).

Combined system dominates alternatives. Our full algorithm, marked by the yellow stars on Figure 2, establishes a new Pareto frontier on both benchmarks. On ALFWorld, we achieve 2.5× cost reduction at iso-quality compared to the teacher baseline, reducing inference costs from $0.059 to $0.024 per episode. At current API pricing, the upfront demonstration cost of $29.50 amortizes after just 843 episodes, with cumulative savings exceeding $34,900 for batch processing scenarios with 1M tasks (see Appendix C.3.4 for full analysis). Notably, our combined system exceeds teacher accuracy (96% vs. 89%), likely because retrieved examples provide the student with both demonstration of the teacher's reasoning and implicit information about environment dynamics across multiple trajectories.

On AppWorld, our system achieves 3.5× cost reduction while recovering 79% of teacher accuracy, demonstrating effective cost-performance trade-offs on this more challenging benchmark. Alternatively, when compared to the Pareto frontier obtained by randomly routing between student and teacher at different ratios, our system delivers a 0.28 boost in task success rate at iso-cost, while we achieve 2× cost reduction at iso-quality.

Method	GPT-4.1-mini	Llama-3.3-70B
ALFWorld		
Teacher	1.00 / 0.89	1.00 / 0.89
Student (ZS)	0.31 / 0.18	0.11 / 0.50
Student (IC)	0.43 / 0.87	0.14 / 0.87
Student (IC+Cascade)	0.41 / 0.96	0.21 / 0.93
AppWorld		
Teacher	1.00 / 0.83	1.00 / 0.83
Student (ZS)	0.09 / 0.28	0.05 / 0.11
Student (IC)	0.15 / 0.55	0.10 / 0.32
Student (IC+Cascade)	0.29 / 0.66	0.19 / 0.44

Table 1: **Across multiple LLMs and benchmarks, Student (IC+Cascade) optimizes cost-accuracy tradeoffs.** Overall comparison of teacher–student configurations on ALFWorld and AppWorld. For both student LLMs, we use Sonnet 4.5 as the teacher. All numbers are formatted (Relative cost / Accuracy). Cost is reported as fraction of teacher cost; Accuracy is episode success rate.

Difficulty	Teacher	Student (IC+Casc.)	Student (ZS)	Tasks
1	0.96	0.91	0.51	57
2	0.85	0.70	0.29	48
3	0.71	0.43	0.06	63

Table 2: **Student (IC+Cascade) is competitive with Teacher on easier AppWorld tasks, but struggles on harder tasks.** Task accuracy by difficulty level on AppWorld. Difficulty levels were manually assigned by benchmark authors based on planning and reasoning complexity (Trivedi et al., 2024).

Our findings generalize to open-weight LLMs To test generality, we evaluate both closed- and open-weight models in structurally distinct domains. In Table 1, trends remain consistent across sparse-reward embodied reasoning (ALFWorld) and multimodal app-use reasoning (AppWorld). On

ALFWorld, Llama-3.3-70B reproduces the in-context gain (0.87 vs. 0.50 zero-shot) and exhibits comparable scaling under self-consistency deferral (boost to 0.93). On AppWorld, Llama-3.3-70B is less effective than GPT-4.1-mini–but experiences similar benefits from in-context distillation and self-consistency deferral. These results indicate that exemplar reuse and confidence-based deferral generalize beyond proprietary APIs.

5.2 Contextualization: Comparing Against More Complex Alternatives

Support for agile development workflows discourages reliance of fine-tuning and specialized system engineering. However to contextualize the cost-accuracy benefits of our approach we compare against these more complex alternatives.

Our boosted GPT-4.1-mini offers better cost-accuracy tradeoffs than zero-shot GPT-4.1. In Figure 2, Zero-shot GPT-4.1 (marked with a green pentagon) achieves 51% accuracy at 56% relative cost on AppWorld—worse than our Student (IC) (55% at 15% cost) and far worse than Student (IC + Cascade) (66% at 29% cost). In-context distillation paired with intelligent deferral dominates fixed model selection. Similar trends hold for ALFWorld. Using a smaller LM (GPT-4.1-mini) with in-context examples from a frontier LM (Sonnet 4.5) can strictly dominate the use of a medium-cost LLM (GPT-4.1).

The accuracy of our simple approach is competitive with bespoke, state-of-the-art compound agentic systems. Our full method (Student IC + Cascade) achieves 65.5% accuracy on AppWorld—approaching IBM's CuGA (Marreed et al., 2025) (73.2% on the leaderboard), a specialized compound system incorporating documentation retrieval, safety guardrails, multi-stage knowledge augmentation, and leverages task-specific modules and complex orchestration infrastructure. Our approach requires only teacher demonstrations, vector indexing, and self-consistency checking. In Section 5.6, we show that if practitioners have access to task difficulty metadata, a simple difficulty-aware routing variant of our method reaches 77.6% accuracy—surpassing CuGA.

Our approach is a training-free alternative to fine-tuning. On ALFWorld, fine-tuning `GPT-4.1-mini` achieved 94% accuracy (matching our 96%). On AppWorld, we were unable to successfully fine-tune within our development timeline: API services failed when training on our teacher demonstrations, and open-weight models (`Llama-3.3-70B`) trained but produced 0% task success despite converging on training loss. We see this outcome as evidence of the complexity of fine-tuning models for complex multi-step agents. It requires careful objective design and extensive debugging—challenges that are well-documented in prior work Barnett et al. (2024). Our method provides a practical alternative for practitioners who seek rapid deployment without this overhead.

5.3 Retrieving more in-context examples can boost task accuracy in exchange for higher costs

To guide practitioners in configuring in-context distillation systems, we vary the number of retrieved teacher exemplars (k) and measure the resulting cost-accuracy tradeoff on ALFWorld and AppWorld (Fig. 3).

On ALFWorld, agent accuracy improves as more examples are added: accuracy rises from 0.75 at k=1 to 0.81 at k=2 (+7.8%), and continues climbing to 0.86 at k=4 (+13.8% from baseline). Beyond this point, improvements plateau—k=6 through k=10 yield accuracies between 0.86–0.88, representing only marginal gains (+2–3%) despite costs increasing from 0.43× to 0.612× teacher cost. The cost-benefit analysis reveals diminishing returns: moving from k=1 to k=4 improves accuracy by 10.4 percentage points while increasing relative cost by 0.12 (54% increase), whereas moving from k=4 to k=10 adds only 1.5 percentage points at a cost increase of 0.26 (76% increase). In this paper, we chose $k = 6$ on ALFWorld for our experiments.

AppWorld exhibits a similar but more modest pattern. Accuracy improves from 0.49 at k=1 to a peak of 0.57 at k=5 (+7.6 percentage points), with costs rising from 0.09× to 0.19× teacher cost. However, the trend is less smooth—accuracy fluctuates between k=3 and k=7 (0.52–0.57), suggesting task-specific sensitivity to example selection. Beyond k=5, additional examples provide no consistent benefit while costs continue to scale linearly. In this paper, we chose $k = 3$ on AppWorld for our experiments.

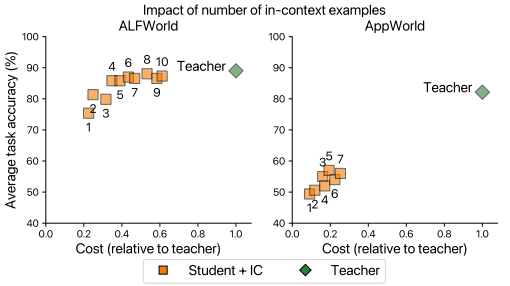

Figure 3: **Retrieving more in-context examples can boost task accuracy in exchange for higher costs**. Cost-accuracy tradeoff for varying numbers of retrieved in-context exemplars (k, labeled on each datapoint) on ALFWorld and AppWorld (Student IC, no cascade). On ALFWorld, accuracy improves rapidly from k=1 to k=4, then exhibits diminishing returns beyond k=6. On AppWorld, accuracy peaks at k=5 with more modest overall gains. In this paper, we use k=6 on ALFWorld and k=3 on AppWorld by default.

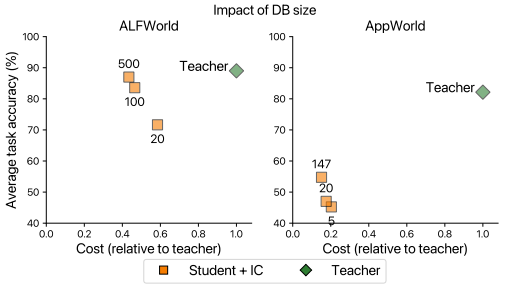

Figure 4: **Scaling teacher database size makes in-context distillation more effective.** The cost-accuracy tradeoff for varying teacher database sizes (labeled on each datapoint) on ALFWorld and AppWorld (Student+IC, no cascade). As we scale database size, more relevant examples are retrieved at each ReAct step, helping the agent solve tasks more successfully (increasing accuracy) and, as a corollary, more efficiently (reducing costs by shortening trajectories).

5.4 Scaling teacher database size makes in-context distillation more effective.

A key practical question is how many teacher demonstrations are needed to achieve strong student accuracy. We vary the size of the teacher demonstration database and measure Student (IC) accuracy without cascading (Fig. 4).

In-context distillation exhibits strong data efficiency: on ALFWorld, just 100 teacher demonstrations yield 0.836 accuracy (94% of teacher accuracy), while 500 demonstrations close the gap to 98%. On AppWorld, 147 demonstrations (the full available training set) achieve 0.548 accuracy (67% of teacher accuracy). Notably, even minimal databases (20 demonstrations on ALFWorld, 5 on AppWorld) substantially outperform zero-shot students (0.18 and 0.28 respectively), confirming that retrieval-based learning transfers effectively even from small exemplar sets.

Interestingly, larger databases also *reduce cost* (AppWorld: 20% of teacher costs at size 5 to 15% at size 147, ALFWorld: 58% of teacher costs at size 20 to 43% at size 500). This occurs because better retrieval quality enables the agent to solve more tasks/take fewer steps to solve tasks, which in turn reduces total ReAct token usage. Our results suggest scaling teacher DB size may further close the accuracy gap, though we leave this to future work on AppWorld where dataset constraints currently limit exploration.

5.5 Student failures are correlated with coverage gaps in retrieved examples

We investigate the relationship between in-context examples and student behavior in our Student+IC setup (without Cascade) for AppWorld, hypothesizing that student failures occur primarily when retrieved examples fail to demonstrate needed behavior—*coverage gaps*. We analyzed 53 tasks where the student (3-shot, no cascade) failed but the teacher succeeded. Using an LLM judge (Gemini-3.0-Flash), we found attribution in 48/53 cases (90.6%), with 33 cases (68.8%) specifically attributed to coverage gaps: retrieved examples did not demonstrate necessary API usage, disambiguation strategy, error handling, or other pivotal behavior. Remaining errors follow a long tail: contradictory examples (8.3%), behavior shown but not applied (2.1%), etc. Student performance is primarily governed by demonstration coverage—when retrieved examples lack necessary operations, students fail. This suggests demonstration collection should prioritize diversity of operations to ensure coverage. While LLM-based attribution is imperfect, we nevertheless believe this provides valuable insight. Future work could iteratively add demonstrations to fill identified coverage gaps. Prompts and qualitative examples are in Appendix E.

5.6 Difficulty-Aware Deferral: Leveraging Domain Knowledge

While self-consistency provides an effective automatic deferral signal, practitioners with domain knowledge about task difficulty can use it to inform routing decisions. We explore this using AppWorld's difficulty labels, where tasks are manually classified into three tiers based on planning complexity and API interaction requirements.

Table 2 shows that Student (IC + Cascade) accuracy degrades with difficulty but maintains substantial gains over Student (ZS) across all levels: 91% vs 51% on difficulty-1 (40pp gap), 70% vs 29% on difficulty-2 (41pp gap), and 43% vs 6% on difficulty-3 (37pp gap). This demonstrates the value of in-context learning across the difficulty spectrum, though the gap to Teacher widens on harder tasks (95% $\rightarrow$ 82% $\rightarrow$ 61% recovery).

Practitioners with difficulty metadata can achieve different cost-accuracy tradeoffs. Using Student (IC + Cascade) for difficulty-1 and difficulty-2 while routing all difficulty-3 tasks to Teacher achieves 0.78 accuracy at 71% of teacher cost—recovering 94% of teacher accuracy with 29% cost reduction. This represents a different operating point: compared to our automatic cascade (0.66 accuracy at 29% cost), difficulty-based routing trades 2.4$\times$ higher cost for 12 percentage points of accuracy. When task difficulty signals are available and maximizing accuracy is critical, combining in-context distillation with domain-specific routing can shift the cost-accuracy tradeoff, though it sacrifices the deployment simplicity of purely automatic approaches.

6 Related Work

Knowledge distillation. Distillation trains a low-cost "student" to mimic a high-cost "teacher" Buciluǎ et al. (2006); Hinton et al. (2015), optimizing student weights via gradient descent on teacher-labeled data. Recent trajectory-level distillation methods train smaller agents to imitate expert behavior (Zeng et al., 2023; Chen et al., 2023a). We also leverage teacher trajectories but perform distillation non-parametrically: trajectories serve as dynamically-retrieved in-context exemplars rather than supervised training targets. This eliminates fine-tuning costs and enables immediate deployment—the same frozen student applies to new tasks by adding demonstrations to the retrieval database.

Dynamic in-context learning and retrieval. Modern LLMs adapt dramatically based on in-context demonstrations (Yin et al., 2024; Agarwal et al., 2024). Prior retrieval-based approaches improve performance by dynamically selecting relevant examples Liu et al. (2021); Su et al. (2022); Qin et al. (2024), but focus on single-query tasks. Recent work extends this to multi-step agent execution, using self-generated experience to improve agent capability over time (Shinn et al., 2023; Zhao et al., 2024; Sarukkai et al., 2025). We build on these approaches—particularly per-step retrieval mechanisms Sarukkai et al. (2025); Zhou et al. (2024)—but apply them to cost reduction rather than capability improvement: we retrieve teacher-generated demonstrations at each step to guide a cheaper student model, enabling knowledge transfer without weight updates. Our contribution is operationalizing this for cost-efficient agentic workflows and characterizing how retrieval quality and demonstration coverage affect cost-accuracy tradeoffs.

Cascades and adaptive routing. Cascades improve cost-accuracy tradeoffs by routing easier instances to cheaper models and harder ones to expensive models Viola & Jones (2001); Wang et al. (2022a); Mamou et al. (2022); Chen et al. (2023b). Prior work uses learned routers Li et al. (2021); Gupta et al. (2024); Chen et al. (2023b); Ong et al. (2025) or model-intrinsic confidence scores Chow (1970); Jitkrittum et al. (2023); Gupta et al. (2024), requiring training or model access. We adopt self-consistency-based routing Yue et al. (2024); Wang et al. (2022b): sampling the student multiple times, where agreement signals confidence and disagreement triggers teacher deferral. This requires no learned components. Self-consistency has been used in single-model settings (Manakul et al., 2023; Wang et al., 2024); we apply it as a cascade routing signal. In our system, self-consistency evaluates the combined effectiveness of retrieval and in-context learning: relevant demonstrations yield convergent samples, while poor retrieval causes divergence.

REFERENCES

Rishabh Agarwal, Avi Singh, Lei Zhang, Bernd Bohnet, Luis Rosias, Stephanie Chan, Biao Zhang, Ankesh Anand, Zaheer Abbas, Azade Nova, et al. Many-shot in-context learning. *Advances in Neural Information Processing Systems*, 37:76930–76966, 2024.

Scott Barnett, Zac Brannelly, Stefanus Kurniawan, and Sheng Wong. Fine-tuning or fine-failing? debunking performance myths in large language models. *arXiv preprint arXiv:2406.11201*, 2024.

Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pp. 535–541, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933395. doi: 10.1145/1150402.1150464. URL https://doi.org/10.1145/1150402.1150464.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023a.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023b.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. Automanual: Generating instruction manuals by llm agents via interactive environmental learning. *arXiv preprint arXiv:2405.16247*, 2024.

C. Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16(1):41–46, 1970. doi: 10.1109/TIT.1970.1054406.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. *arXiv preprint arXiv:2403.08978*, 2024.

Neha Gupta, Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Aditya Krishna Menon, and Sanjiv Kumar. Language model cascades: Token-level uncertainty and beyond. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=KgaBScZ4VI.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *arXiv preprint arXiv:2305.02301*, 2023.

Wittawat Jitkrittum, Neha Gupta, Aditya Krishna Menon, Harikrishna Narasimhan, Ankit Singh Rawat, and Sanjiv Kumar. When does confidence-based cascade deferral suffice? In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Tomoyuki Kagaya, Thong Jing Yuan, Yuxuan Lou, Jayashree Karlekar, Sugiri Pranata, Akira Kinose, Koki Oguri, Felix Wick, and Yang You. Rap: Retrieval-augmented planning with contextual memory for multimodal llm agents. *arXiv preprint arXiv:2402.03610*, 2024.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Lei Li, Yankai Lin, Deli Chen, Shuhuai Ren, Peng Li, Jie Zhou, and Xu Sun. Cascadebert: Accelerating inference of pre-trained language models via calibrated complete models cascade, 2021. URL https://arxiv.org/abs/2012.14682.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

Jonathan Mamou, Oren Pereg, Moshe Wasserblat, and Roy Schwartz. Tangobert: Reducing inference cost by using cascaded architecture, 2022. URL https://arxiv.org/abs/2204.06271.

Potsawee Manakul, Adian Liusie, and Mark JF Gales. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. *arXiv preprint arXiv:2303.08896*, 2023. URL https://arxiv.org/abs/2303.08896.

Sami Marreed, Alon Oved, Avi Yaeli, Segev Shlomov, Ido Levy, Offer Akrabi, Aviad Sela, Asaf Adi, and Nir Mashkif. Towards enterprise-ready computer using generalist agent. *arXiv preprint arXiv:2503.01861*, 2025.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=8sSqNntaMr.

Chengwei Qin, Aston Zhang, Chen Chen, Anirudh Dagar, and Wenming Ye. In-context learning with iterative demonstration selection. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 7441–7455, 2024.

Vishnu Sarukkai, Zhiqiang Xie, and Kayvon Fatahalian. Self-generated in-context examples improve LLM agents for sequential decision-making tasks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=WdL3O58gde.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Hongjin Su, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, et al. Selective annotation makes language models better few-shot learners. *arXiv preprint arXiv:2209.01975*, 2022.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *ACL*, 2024.

P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pp. I–I, 2001. doi: 10.1109/CVPR.2001.990517.

Han Wang, Archiki Prasad, Elias Stengel-Eskin, and Mohit Bansal. Soft self-consistency improves language model agents. *arXiv preprint arXiv:2402.13212*, 2024. URL https://arxiv.org/abs/2402.13212.

Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. Minilmv2: Multi-head self-attention relation distillation for compressing pretrained transformers. *arXiv preprint arXiv:2012.15828*, 2020.

Xiaofang Wang, Dan Kondratyuk, Eric Christiansen, Kris M. Kitani, Yair Movshovitz-Attias, and Elad Eban. Wisdom of committees: An overlooked approach to faster and more accurate models. In *International Conference on Learning Representations*, 2022a. URL https://openreview.net/forum?id=MvO2t0vbs4-.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022b. URL https://arxiv.org/abs/2203.11171.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Qingyu Yin, Xuzheng He, Luoao Deng, Chak Tou Leong, Fan Wang, Yanzhao Yan, Xiaoyu Shen, and Qiang Zhang. Deeper insights without updates: The power of in-context learning over fine-tuning. *arXiv preprint arXiv:2410.04691*, 2024.

Ming Yue, Jieyu Zhao, et al. Large language model cascades with mixture of thoughts. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=6okaSfANzh.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*, 2023.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Ruiwen Zhou, Yingxuan Yang, Muning Wen, Ying Wen, Wenhao Wang, Chunling Xi, Guoqiang Xu, Yong Yu, and Weinan Zhang. Trad: Enhancing llm agents with step-wise thought retrieval and aligned decision. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 3–13, 2024.

A Per-Step Retrieval Reduces Cost Without Sacrificing Accuracy

A natural question is whether retrieving fresh examples at each step is necessary, or whether retrieving once at the start of the trajectory suffices. We compare our default **per-step retrieval**—which dynamically retrieves the top-k most relevant trajectory windows at each decision point—against a **single retrieval** baseline that retrieves k full trajectories once at the beginning and includes them in context for the entire episode.

Table 3 shows that per-step retrieval achieves equivalent accuracy while substantially reducing cost. On ALFWorld, both approaches achieve 87% accuracy, but single retrieval incurs 54% of teacher cost versus 43% for per-step retrieval—a 26% cost increase for no performance gain. On AppWorld, the pattern holds: accuracy remains essentially identical (55% vs. 54%), while the single-retrieval setup costs 60% more (24% vs. 15% of teacher cost).

Table 3: Comparison of retrieval granularity strategies. Per-step retrieval matches single-retrieval accuracy while reducing cost by dynamically selecting relevant windows rather than including full trajectories in context.

Method	Benchmark	Accuracy	Rel. Cost
Per-step retrieval	ALFWorld	0.87	0.43
Single retrieval	ALFWorld	0.87	0.54
Per-step retrieval	AppWorld	0.55	0.15
Single retrieval	AppWorld	0.54	0.24

The cost difference arises because single retrieval includes entire teacher trajectories (often 10-20 steps) in the student's context throughout execution, while per-step retrieval selects only the most relevant 3-5 step windows at each decision point. Since LLM API costs scale with input tokens, the longer context in single retrieval directly translates to higher costs.

Interestingly, the lack of accuracy difference suggests that **retrieval relevance matters more than context volume**. Including full trajectories does not help the student generalize better—it simply pays for irrelevant context. This validates our design choice to dynamically retrieve short, highly relevant windows rather than relying on static, comprehensive examples.

B PROMPTS

We use identical prompt formats across both the ALFWorld and AppWorld benchmarks for all agent operations. The templates are deliberately minimal to avoid prompt engineering overhead: they specify the task structure and insert retrieved examples in the appropriate format (goals with plans for planning prompts, state-action-reasoning triples for execution prompts), but require no per-task customization.

Plan:

```
system_prompt: f'You are an expert at generating high-level plans of
    actions to achieve a goal.\n Here is your action space: {
    action_space}.\n Here are some examples of goal,plan from
    episodes that successfully achieved similar goals: {examples}'
user_prompt: f'goal: {goal}\n plan: '
```

ReAct:

```
system_prompt: f"You are a ReAct agent that is an expert at
    reasoning and taking actions to accomplish a goal. \n Goal: '{
    self.goal}'. \n You will receive observations and respond with *
    exactly* one reasoning and one action per observation. \n Write
    them on their own lines, with these exact labels and a colon: \n
    reasoning: <your step-by-step reasoning, concise, may be multi-
    line> \n action: <a single, directly executable command or final
    answer> \n Use the labels exactly as written: 'reasoning:' and '
    action:' (no variations). \n Here is your action space: {
    action_space}.\n Here are some examples of goal, plan,
    observation, reasoning, action from episodes that successfully
    achieved similar goals: {examples}"
user_prompt: f'goal: {goal}\n plan: {plan}\n trajectory: {trajectory
    }\n action: '
```

Verifier:

```
system_prompt: f"You are an expert LLM judge that determines if a
    set of actions that could be taken by an LLM agent are all
    effectively equivalent. First, you will be provided the current
    state of the agent: the trajectory so far. You will receive a set
     of actions and respond with 'YES' if they are all equivalent and
     'NO' if at least one is different. YOU MUST OUTPUT NOTHING ELSE.
     \n Here is your action space: {action_space}.\n Here are some
    examples of goal, plan, observation, reasoning, action from
    episodes that successfully achieved similar goals: {examples}"
user_prompt: f'goal: {goal}\n plan: {plan}\n trajectory: {trajectory
    }\n Set of actions: {set(individual_actions)}, effectively
    equivalent? '
```

C IMPLEMENTATION DETAILS AND COST MODEL

C.1 MODEL CONFIGURATION

Closed-weight models are queried via OpenAI/Anthropic APIs, with max 4096 output tokens for actions. All models are run with `temperature=0.1` unless otherwise noted. For our per-step retrieval mechanism (Section 3), we use `MiniLM-L6-v2` (Wang et al., 2020) as the embedding model. All models use identical system prompts and task descriptions; only in-context exemplars vary. Full prompts in Appendix B.

C.2 Token Usage and Cost Calculation Details

This appendix provides detailed token usage statistics and cost calculations for all methods evaluated in our experiments. All costs are computed by tracking every LLM call made during episode execution, including student queries, teacher queries (when cascading or randomly routing), and LLM verifier calls for semantic agreement checking (AppWorld only). While optimized KV caching of in-context examples has the potential to reduce costs further, we omit caching from our cost model and leave caching-aware costs as future work. We do not attribute any cost to the process of vector database lookup to retrieve examples.

C.3 Cost Calculation Methodology

For each episode, we record input and output token counts for all model invocations and compute total cost using API pricing as of October 2025:

- **GPT-4.1-mini**: \$0.40 per 1M input tokens, \$1.60 per 1M output tokens
- **GPT-4.1**: \$2.00 per 1M input tokens, \$8.00 per 1M output tokens
- **Claude Sonnet 4.5**: \$3.00 per 1M input tokens, \$15.00 per 1M output tokens
- **Llama-3.3-70B** (via NovitaAI, similar costs available on other providers): \$0.13 per 1M input tokens, \$0.39 per 1M output tokens

For each configuration, total episode cost is:

$$\text{Cost}_{\text{episode}} = \frac{1}{10^6}\Big(c_{\text{in}}^{(s)} \cdot T_{\text{in}}^{(s)} + c_{\text{out}}^{(s)} \cdot T_{\text{out}}^{(s)} + c_{\text{in}}^{(t)} \cdot T_{\text{in}}^{(t)} + c_{\text{out}}^{(t)} \cdot T_{\text{out}}^{(t)}\Big),$$

where $T_{\text{in/out}}^{(s)}$ and $T_{\text{in/out}}^{(t)}$ denote total input/output tokens for student and teacher models respectively, and c denotes the corresponding per-token prices.

Normalized cost is computed relative to the teacher baseline:

$$\text{Cost}_{\text{normalized}} = \frac{\text{Cost}_{\text{episode}}}{\text{Cost}_{\text{teacher}}}.$$

With Claude Sonnet 4.5 token pricing, the teacher baseline costs \$0.059 (USD) per episode on ALFWorld, and \$0.589 (USD) per episode on AppWorld.

C.3.1 ALFWorld Token Usage

Table 4 reports average token usage and costs across 134 test episodes for all methods on ALFWorld. "Blend" denotes the random routing baseline at various fractions of teacher LLM calls (ex. Blend (0.2) is 20% teacher LLM usage, 80% student LLM usage).

C.3.2 AppWorld Token Usage

Table 5 reports average token usage and costs across 168 test episodes for all methods on AppWorld. "Blend" denotes the random routing baseline at various fractions of teacher LLM calls (ex. Blend (0.2) is 20% teacher LLM usage, 80% student LLM usage).

C.3.3 Key Observations

Token efficiency of dynamic in-context learning. Comparing Student (ZS) to Student (IC), we observe that adding retrieved exemplars *reduces* trajectory length (ALFWorld: 27.0 $\rightarrow$ 12.1 steps; AppWorld: 19.3 $\rightarrow$ 12.1 steps) despite increasing input tokens per step. This occurs because better-guided students solve tasks more directly, avoiding exploratory dead ends. The net effect is higher input token cost per step but fewer steps overall, yielding favorable cost-performance tradeoffs.

Table 4: Detailed token usage and cost breakdown for ALFWorld. All values are averages per episode.

Method	Student Input	Student Output	Teacher Input	Teacher Output	Steps per Task	Teacher Frac	Acc	Cost (norm.)
Teacher	—	—	16257	684	12	—	0.89	1.0
Student (ZS)	41457	1130	—	—	27	0.00	0.18	0.31
Student (IC)	61966	527	—	—	12	0.00	0.87	0.43
Student (IC + Cascade)	50648	1283	597	26	9.9	0.039	0.96	0.42
Student (Cascade only)	26669	2194	2910	125	19	0.10	0.63	0.42
Blend (0.2)	22561	683	6496	234	20	—	0.64	0.56
Blend (0.4)	13183	436	9657	365	16	—	0.79	0.68
Blend (0.6)	7813	276	12670	482	14	—	0.80	0.83
Blend (0.8)	3106	149	16316	616	13	—	0.87	1.0
GPT-4.1 (ZS)	25400	720	—	—	—	0.00	0.75	0.96
Llama-3.3-70B (ZS)	44655	2275	—	—	—	0.00	0.50	0.11
Llama-3.3-70B (IC)	61727	951	—	—	—	0.00	0.87	0.14
Llama-3.3-70B (IC + Cascade)	55706	2385	1113	50	—	0.06	0.93	0.21

Table 5: Detailed token usage and cost breakdown for AppWorld. All values are averages per episode.

Method	Student Input	Student Output	Teacher Input	Teacher Output	Steps per Task	Teacher Frac	Acc	Cost (norm.)
Teacher	—	—	185460	2183	20	—	0.82	1.0
Student (ZS)	118430	2317	—	—	19	0.00	0.28	0.087
Student (IC)	216563	1623	—	—	12	0.00	0.55	0.15
Student (IC + Cascade)	251683	4829	18385	329	12	0.22	0.66	0.29
Student (Cascade only)	184487	7009	28445	440	18	0.21	0.41	0.30
Blend (0.2)	96831	1809	27597	398	18	—	0.33	0.22
Blend (0.4)	85588	1520	61349	778	19	—	0.47	0.39
Blend (0.6)	57568	1018	74379	1066	18	—	0.58	0.45
Blend (0.8)	23362	376	113894	1527	17	—	0.69	0.64
GPT-4.1 (ZS)	155035	2603	—	—	—	0.00	0.51	0.56
Llama-3.3-70B (ZS)	198953	4893	—	—	—	0.00	0.11	0.047
Llama-3.3-70B (IC)	445542	3438	—	—	—	0.00	0.32	0.10
Llama-3.3-70B (IC + Cascade)	324814	6616	20749	419	—	0.13	0.44	0.19

Cascade overhead. Self-consistency cascades add sampling overhead (multiple student queries per step) but trigger teacher queries on only a small fraction of steps (ALFWorld: 3.9%; AppWorld: 22.2%). Since student queries cost 10-100$\times$ less than teacher queries, the net cost increase from cascading is modest (ALFWorld: 0.434 $\rightarrow$ 0.415 normalized cost; AppWorld: 0.151 $\rightarrow$ 0.286).

Teacher fraction vs. cost. Comparing Blend configurations shows that cost scales approximately linearly with teacher fraction, but random routing provides much worse accuracy than adaptive cascades at equivalent teacher usage (e.g., AppWorld Blend 0.2 vs. IC + Cascade: similar teacher fraction but 0.221 vs. 0.286 normalized cost due to different student base capability).

C.3.4 Amortization Analysis: When Do Inference Savings Offset Demonstration Costs?

A practical concern for deployment is whether the upfront cost of collecting teacher demonstrations is justified by inference-time savings. We analyze the breakeven point where cumulative inference cost reductions exceed the one-time cost of constructing the teacher database $\mathcal{D}$.

Setup. Let C_{demo} denote the total cost of collecting teacher demonstrations on $\mathcal{G}_{\text{train}}$, computed as:

$$C_{\text{demo}} = |\mathcal{G}_{\text{train}}| \cdot C_{\text{teacher}}^{\text{episode}},$$

where $C_{\text{teacher}}^{\text{episode}}$ is the average per-episode cost of running the teacher model (reported in Tables 4 and 5).

At deployment, let C_{baseline} denote the per-episode cost of the baseline approach (e.g., always using the teacher), and C_{ours} denote the per-episode cost of our Student (IC + Cascade) method. After processing N test episodes, the total cost including demonstration collection is:

$$\begin{aligned} \text{Total}_{\text{baseline}} &= N \cdot C_{\text{baseline}}, \\ \text{Total}_{\text{ours}} &= C_{\text{demo}} + N \cdot C_{\text{ours}}. \end{aligned}$$

The breakeven point N^* occurs when $\text{Total}_{\text{ours}} = \text{Total}_{\text{baseline}}$:

$$N^* = \frac{C_{\text{demo}}}{C_{\text{baseline}} - C_{\text{ours}}}.$$

ALFWorld breakeven analysis. Using costs from Table 4:

- Teacher demonstration cost: $C_{\text{demo}} = 500 \times \$0.059 = \$29.50$
- Baseline (Teacher) per episode: $C_{\text{baseline}} = \$0.059$
- Student (IC + Cascade) per episode: $C_{\text{ours}} = \$0.024$
- Per-episode savings: $\Delta C = \$0.059 - \$0.024 = \$0.035$

Breakeven point:

$$N^* = \frac{\$29.50}{\$0.035} \approx 843 \text{ episodes.}$$

After processing 843 test episodes, cumulative savings offset the upfront demonstration cost. For a batch processing scenarios with 1,000,000 tasks, over $34,900 is saved.

AppWorld breakeven analysis. Using costs from Table 5:

- Teacher demonstration cost: $C_{\text{demo}} = 147 \times \$0.59 = \$86.73$
- Baseline (Teacher) per episode: $C_{\text{baseline}} = \$0.59$
- Student (IC + Cascade) per episode: $C_{\text{ours}} = \$0.17$
- Per-episode savings: $\Delta C = \$0.59 - \$0.17 = \$0.42$

Breakeven point:

$$N^* = \frac{\$86.73}{\$0.42} \approx 207 \text{ episodes.}$$

After processing just 207 test episodes, the method becomes cost-effective. For deployment on 1,000,000 tasks, over $419,000 is saved.

C.4 Retrieval Costs

We do not attribute any cost to the process of vector database lookup to retrieve examples. Since our databases include only a few hundred demonstrations, it is feasible to make these costs negligible in practice using standard vector search infrastructure.

D Benchmark Details

D.1 Real-World Motivation

One of the most common use of agents in real-world industry practice involves automating rote workflows such as common business processes, processing customer service tickets, and implementing data processing pipelines. These tasks benefit from rapid prototyping and deployment without extensive model training.

D.2 AppWorld

AppWorld tasks mirror realistic daily business workflows. Tasks are evaluated via automated unit tests checking both state changes and execution traces, ensuring that agents not only achieve the correct final state but also follow reasonable execution patterns.

D.3 ALFWorld

ALFWorld is a standard benchmark in several prior works on agentic self-improvement (Zhao et al., 2024; Fu et al., 2024; Chen et al., 2024; Sarukkai et al., 2025). The availability of a large train set (up to 3500 tasks) enables us to test how our method scales with varying demonstration set sizes.

D.4 Scale and Amortization

Few academic benchmarks (including AppWorld and ALFWorld) exhibit the scale where $|\mathcal{T}_{\text{test}}| \gg |\mathcal{T}_{\text{demo}}|$ by orders of magnitude. In practice, deployment scenarios like large-scale batch processing or production agent services would process many more test instances, making demonstration collection costs increasingly negligible. Our experimental setup approximates this by excluding demonstration costs from reported metrics.

E Mechanistic Role of In-Context Examples in Student Behavior

On AppWorld, to understand why the student model with in-context examples (3ic) failed on tasks where the teacher model succeeded, we performed a two-stage analysis. First, we identified the "difference maker" step–the point where, faced with similar states, the teacher took a fundamentally different action that led to success while the student's approach led to failure. Second, we attributed the student's behavior at this pivotal step to the content of the in-context examples provided, examining whether the examples demonstrated the necessary behavior, provided conflicting guidance, or failed to teach critical patterns. Both analyses used the Gemini-3.0-Flash. The main results of the analysis are included in the main paper–we include both an anecdotal example and the prompts used to conduct the analysis below.

E.1 Anecdotal example of insufficient coverage in retrieved in-context examples

Task: AppWorld task 6 (test set) requires removing a set of songs from a Spotify queue.

Teacher behavior: The teacher model successfully completes the task by deleting songs back-to-front from the queue, using a specific deletion strategy that maintains queue indices correctly.

Student behavior (3-shot): The student with in-context examples fails to remove the songs successfully.

Retrieved in-context examples: The system retrieved three teacher demonstrations (training examples 47, 55, 58). Example 55 demonstrates how to retrieve the song queue, but critically, *none of the three demonstrations show an effective strategy to delete songs from the queue*. Without an example demonstrating the deletion operation, the student lacks the necessary guidance to execute this pivotal step, leading to task failure.

Analysis: This example illustrates a clear coverage gap—the operation needed (queue deletion) was not demonstrated in any of the retrieved examples. The student had examples of related operations (queue retrieval) but lacked coverage of the specific operation required for success.

E.2 Prompt for identifying divergent step between student and teacher

This prompt identifies the "difference maker" step where the teacher succeeded but the student failed.

```
f"""You are analyzing two trajectories for the same task. One
    trajectory is
```

```
from a student model (student) which FAILED, and one is from a teacher
model (teacher) which SUCCEEDED.

Your task is to identify the "difference maker" step - the point where,
faced with the same or similar state/situation, the teacher took a
fundamentally different action or strategy that led to success, while
the student's approach led to failure.

IMPORTANT CONSIDERATIONS:
1. The step numbers in each trajectory may be DIFFERENT (e.g., step 5
   in student vs step 8 in teacher)
2. Look for points where both trajectories were in similar states
   (similar observations, similar context)
3. Identify where one trajectory made a strategic choice that the
   other didn't
4. This may NOT be the first difference - trajectories might converge
   and diverge multiple times
5. Focus on the decision that represents the most significant strategic
   difference

Consider a "difference maker" if:
- The teacher used a different API or approach that solved a critical
  problem the student missed
- The teacher recognized a prerequisite or dependency the student
  overlooked
- The teacher handled an error or edge case that caused the student
  to fail
- The teacher took a more direct or correct path to the solution
- The teacher demonstrated better understanding of the problem domain
  or API requirements

Do NOT consider it a "difference maker" if:
- It's just a minor variation in code syntax
- It's the same strategy with different variable names
- The trajectories were already on different paths due to earlier
  differences

Here are the steps from both trajectories:

STUDENT TRAJECTORY (3ic):
[Steps with reasoning, action, and observation for each step]

TEACHER TRAJECTORY (zeroic):
[Steps with reasoning, action, and observation for each step]

Please respond with ONLY a JSON object in this exact format:
{
  "student_step": <step_number_in_student_trajectory>,
  "teacher_step": <step_number_in_teacher_trajectory>,
  "student_action": "<the action taken in student at that step>",
  "teacher_action": "<the action taken in teacher at that step>",
  "similar_state_description": "<description of the similar state/
    situation both trajectories were in>",
  "explanation": "<detailed explanation of why the teacher's approach
    succeeded where the student's failed>",
  "why_teacher_succeeded": "<explanation of what the teacher did
```

```
    correctly that the student missed>",
  "key_lesson": "<the key lesson or insight that the student should
    learn from the teacher's approach>"
}

If you cannot find a clear difference maker, respond with:
{
  "student_step": null,
  "teacher_step": null,
  "explanation": "<explanation of why no clear difference maker was
    found>"
}"""
```

E.3 Prompt for checking relationship between student action and in-context examples

This prompt analyzes whether pivotal student behavior should be attributed to in-context examples.

```
f"""You are analyzing how in-context examples influenced a model's
behavior at a pivotal step.

The student model WITH in-context examples (3ic) failed, while the
teacher model succeeded.

PREVIOUS ANALYSIS RESULT:
[The JSON output from the difference maker step analysis]

PIVOTAL STEP INFORMATION:
- Step number in student trajectory: {pivotal_step}
- Student file: {student_file}

What content from the in-context examples may have led the 3ic model
to take the failing action, or what did the examples fail to teach
that the teacher model knew?

IN-CONTEXT EXAMPLES PROVIDED AT THIS STEP:
The following examples were provided to the student model at step
{pivotal_step}:
[Example content snippets]

Please analyze:
1. Is there content in these examples that directly shows or
    suggests
   the successful/failing action?
2. Are there patterns, strategies, or API usage patterns in the
   examples that may have influenced the behavior?
3. For the failing case: Did the examples teach something incorrect,
   or fail to teach something important?
4. For the succeeding case: What specific content from the examples
   led to the correct approach?

Please respond with ONLY a JSON object in this exact format:
{
  "attribution_found": <true_or_false>,
  "example_ids_involved": [<list_of_example_ids_that_may_have_
    influenced>],
  "specific_content": "<specific text or patterns from examples that
    may have influenced the behavior>",
  "how_it_influenced": "<explanation of how the example content
    influenced the behavior>",
  "confidence": "<high/medium/low - confidence in this attribution>",
```

```
  "alternative_explanation": "<if attribution is low confidence, what
    else might explain the behavior?>"
}

If no clear attribution can be made, set "attribution_found" to
    false
and explain why."""
```

F TEACHER PERFORMANCE WITH IN-CONTEXT EXAMPLES

In Section 5.1, we report that our Student (IC + Cascade) method achieves 94% accuracy on ALFWorld, exceeding the baseline Teacher accuracy of 89%. A natural question is whether this performance gain comes from the cascade mechanism selecting high-quality teacher responses, or from the boost provided by in-context examples themselves.

To contextualize this result, we ran an additional experiment: Teacher + IC, where the teacher model (`Claude Sonnet 4.5`) is provided with the same retrieved in-context examples at each step as the student receives. This configuration achieves **94% accuracy**, matching Student (IC + Cascade).

This result confirms that the performance gain is attributable to in-context learning: when relevant teacher demonstrations are provided as examples, both teacher and student models benefit. The Student (IC + Cascade) configuration matches teacher performance by combining two mechanisms: (1) boosting the student with in-context examples when they provide consistent guidance, and (2) cascading to the (boosted) teacher when student samples diverge. The fact that both Teacher + IC and Student (IC + Cascade) reach 94% suggests that in-context learning lifts the performance ceiling on this benchmark, even for a teacher learning from its own examples, and our cascade mechanism successfully navigates between the boosted student and teacher to achieve this ceiling cost-efficiently.