TOWARDS BUDGET-AWARE AGENTS: DO AGENTS KNOW WHAT THEY WILL SPEND?

Anonymous authors
Paper under double-blind review

ABSTRACT

Foundation-model agents operate under growing resource constraints yet rarely know how much budget they will spend. We call this capability *budget awareness* and formalize it as *progressive interval estimation*: mid-execution, can the agent provide an interval that covers its remaining budget and predict when its continuation will not complete within budget? We score this with a rollout-replay protocol that re-queries the agent on every trajectory prefix, decomposing estimation into feasibility prediction, early failure detection, and interval estimation. We evaluate five frontier models across four environments, including internal token budgets (Sokoban, Search-R1, SWE-bench) and external multi-dimensional budgets (Warehouse), and train Qwen-7B estimators with SFT and RL. Task success is only a loose proxy for estimation quality: frontier models differ significantly as estimators, but not in order of success. Estimation fails in structured ways: agents underestimate remaining cost on search and coding, recognize failure late, and widen intervals with task scale rather than continuation uncertainty. At a matched loss of solved tasks, the models' infeasibility signal saves more tokens than a hindsight-tuned spending rule in 9 of 15 model–environment cells. SFT makes binary feasibility learnable in-domain, whereas interval estimation stays task-specific.

1 INTRODUCTION

Foundation-model agents are increasingly deployed in longer horizons and higher-stakes tasks: a coding agent consumes tokens per reasoning step, a web agent spends API calls per search query, and a supply-chain agent commits real dollars and warehouse capacity per procurement decision. Both internal compute budgets and external resource budgets grow rapidly with deployment horizon, yet existing benchmarks track spending only after the fact. We ask whether the agent itself knows, mid-execution, what it is about to spend—a capability we call *budget awareness*: the ability to estimate remaining budget and declare when the task has become infeasible. An agent lacking this cannot abort hopeless tasks, request more resources, or allocate budget across sub-goals.

Two gaps prevent systematic study of this capability. **First**, agent research (Liu et al., 2026; Ding et al., 2026; McCleary & Ghawaly, 2026) usually calculates token consumption as a post-hoc metric, but hardly ever asks whether the agent could *self-estimate* how much budget it would need. **Second**, most evaluation protocols collect a single-point prediction at task start, which mismatches long-horizon agentic tasks where feasibility evolves turn by turn: a project manager re-estimates the remaining timeline at milestones, gives a range for estimations, and flags when finishing becomes infeasible.

To address these limitations, we propose *progressive interval estimation* as a new agent capability that is *range-valued* and *progressive throughout execution*. We record a full agent rollout with the budget cap hidden from the acting agent, then query the same agent separately at every turn: *given current progress, how much budget remains to finish? Provide an interval, or declare that you will not finish within budget.* We decompose this capability into three sub-capabilities (feasibility prediction, early failure detection, and interval estimation) and ask five frontier models to perform it across four environments (Sokoban, Search-R1, SWE-bench, and a real-data-informed Warehouse simulation with three coupled budget dimensions) covering both internal and external budget modalities. Across these experiments, we find:

- **Task success is a loose proxy for estimation quality.** Across the full capability range the two co-vary (r=0.81). Among the four frontier models, hit rates differ significantly (pre-specified

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

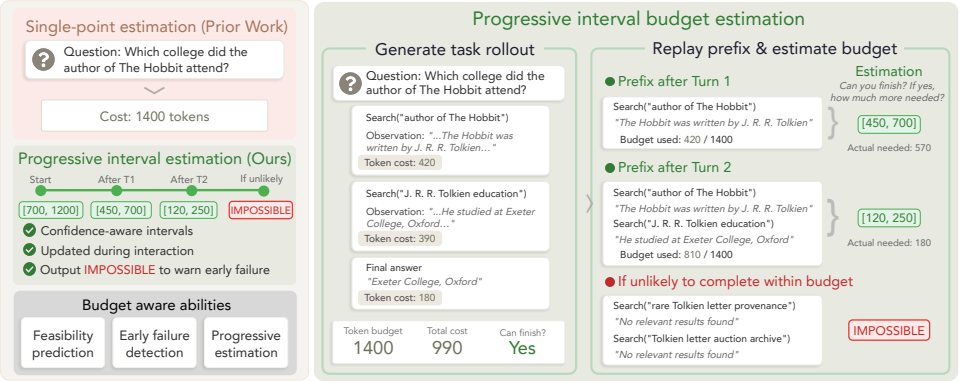

Figure 1: Progressive interval estimation. We record a rollout with the budget hidden from the agent (exceeding it counts as failure), then re-query the same agent on every prefix for an interval over the remaining budget or an `impossible` declaration, scored against the realized outcome.

cluster-permutation test, combined p=0.011), yet success orders them only loosely: 5 of 17 model pairs are reversed, and on SWE-bench the lowest-success frontier model estimates best (§4).

- **Estimates are biased and failure is recognized too late.** On search and coding, median estimates fall below the realized cost in 9 of 10 model cells, and weaker models are *more* optimistic; on Sokoban, frontier models lean conservative. Only 18% of failed rollouts are flagged before 60% of the budget is spent, and a one-feature rule on budget spent matches every model's feasibility F_1 (§5).
- **Binary feasibility is learnable in-domain; interval estimation is task-specific.** On an in-domain Sokoban probe, supervised fine-tuning raises balanced feasibility accuracy from 25.5% to 91.1%; a perfect format-only repair is capped at 52.4%. Across tasks, the feasibility format transfers but interval coverage falls to 0–10% (§5.1, App. I).
- **The signal is actionable.** Stopping at the first `impossible` saves up to 57% of tokens on unsolved runs; at a matched loss of solved tasks it beats a hindsight-tuned spending rule in 9 of 15 cells, including every model on Sokoban (§6).

In general, our results show that budget is less a metric for after-the-fact accounting than a control signal that resource-limited agents currently lack.

2 Two Failures of Single-Point Budget Estimation

We start with the simplest budget probe: at task start, ask the agent for a single-point estimate of total tokens, scored against the realized rollout. We test five frontier models on Sokoban and Search-R1 with both first-turn and later-turn estimates (App. B.3). Two failures emerge across all models. These observations draw on the rollouts analyzed in §5; they motivate the protocol rather than serve as independent evidence.

The first failure is directional bias. Aggregated over environments, underestimates of the realized budget outnumber overestimates for all five models (Figure 2), and weaker models on a task are *more* optimistic; frontier models are conservative on Sokoban (§5.2).

The second failure is that first-turn and later-turn estimates do not agree. On Sokoban, Gemini's feasibility macro-F_1 improves by +35.6 points from first-turn to all-turn evaluation; on Search-R1, Qwen3-235B moves the opposite way by 7.4 points (§5.3). First-turn judgment is neither a consistent over- nor underestimate; it is simply a different judgment whose direction is model- and task-specific.

These failures motivate *progressive interval estimation* (§3): intervals replace points, an `impossible` option replaces silent underestimation, and per-turn queries replace k=0 probes.

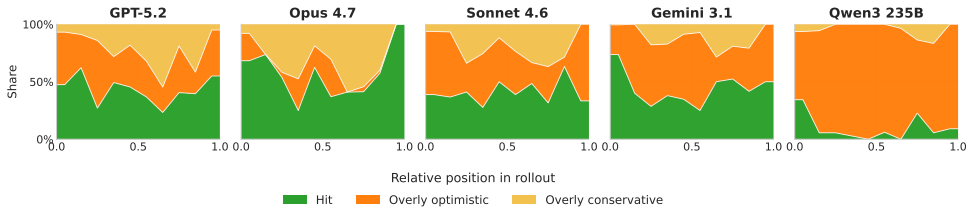

Figure 2: Direction of interval misses on feasible prefixes, aggregated over environments per model. Optimistic misses dominate in aggregate for every model; per environment, frontier models lean conservative on Sokoban (§5.2, App. Figure 10).

3 Agent Budget Awareness

3.1 Budget Modalities: Internal and External

Consider an agent executing a multi-turn trajectory $\tau = \{(o_t, z_t, a_t)\}_{t=1}^{T}$, where o_t is the observation (e.g., environment state or tool output) received at turn t, z_t is the agent's reasoning trace, and a_t is the action. At each turn, the agent commits budget in two fundamentally different ways.

Internal budget. Internal budget is the compute generated by the model's own reasoning. Let c_t^{in} denote the fresh token count at turn t. The cumulative internal cost up to turn k is $C_k^{\text{in}} = \sum_{t=1}^{k} c_t^{\text{in}}$, and the remaining internal cost from turn $k{+}1$ onward is $R_k^{\text{in}} = C_T^{\text{in}} - C_k^{\text{in}}$. The agent operates under a total cap B^{in}, requiring $C_T^{\text{in}} \le B^{\text{in}}$. A trajectory whose total cost exceeds B^{in} counts as a failure, even if it solves the task. Internal-budget estimation tests whether the model can predict its own compute footprint: how many tokens will the remaining reasoning, planning, and action steps cost?

External budget. External budget is the cost the agent commits in the environment. It is generally multi-dimensional: let $\mathbf{c}_t^{\text{ex}} \in \mathbb{R}^D$ denote the D-dimensional cost vector at turn t, with cumulative usage and remaining cost $\mathbf{C}_k^{\text{ex}} = \sum_{t=1}^{k} \mathbf{c}_t^{\text{ex}}$, where $\mathbf{R}_k^{\text{ex}} = \mathbf{C}_T^{\text{ex}} - \mathbf{C}_k^{\text{ex}}$, subject to per-dimension constraints $C_T^{\text{ex},(d)} \le B^{(d)}$ for $d = 1, \ldots, D$. External budget forces the agent to reason about coupled resource constraints: holding more inventory raises revenue but eats warehouse capacity, while drawing credit improves short-term cash but creates repayment pressure later.

3.2 Progressive Interval Estimation and Rollout-Replay

Progressive interval estimation. At every turn k, given the trajectory prefix $\tau_{\le k} = \{(o_t, z_t, a_t)\}_{t=1}^{k}$ and cumulative usage C_k, the estimator returns

$$\hat{y}_k = \begin{cases} [\hat{R}_k^{\text{lo}},\ \hat{R}_k^{\text{hi}}] & \text{if feasible,} \\ \texttt{impossible} & \text{otherwise.} \end{cases} \tag{1}$$

The output has three properties: *uncertainty* (interval width); *progressiveness* (updates each turn); and *infeasibility awareness* (an explicit `impossible` option for early stopping or rerouting). The first two address the two single-point failures; the third makes the signal actionable.

Rollout-replay protocol. To separate budget estimation ability from task completion ability, we use a two-phase procedure (Figure 1). **(1) Rollout generation.** The agent executes the task without being shown the budget. We log the full trajectory τ together with per-turn cost c_t and final outcome; a rollout whose total cost exceeds the budget counts as a failure, even if it solves the task. **(2) Prefix replay and estimation.** For each non-terminal turn $k \in \{1, \ldots, T{-}1\}$, we replay the prefix $\tau_{\le k}$ with a usage summary and ask the agent for $\hat{y}_k$ (Eq. 1). Each prediction is scored against true remaining cost $R_k = C_T - C_k$ and the true outcome.

Scope of the label and budget information. Feasibility is *agent-relative*: it asks whether the acting agent, continuing from its own prefix, finishes within budget, rather than whether an optimal policy could solve the underlying task state. Reasoning errors are thus a cause of budget exhaustion, not an alternative to it. Throughout, "infeasible" and `impossible` are shorthand for this operational label: the agent's own continuation will not complete within budget. The acting agent does not see the cap during rollout generation, whereas the estimator is told the cap explicitly because estimation presupposes a resource constraint. The prompt defines the target as the realized cost of the logged continuation and illustrates it with a worked example (App. J); the estimator is asked what the rollout

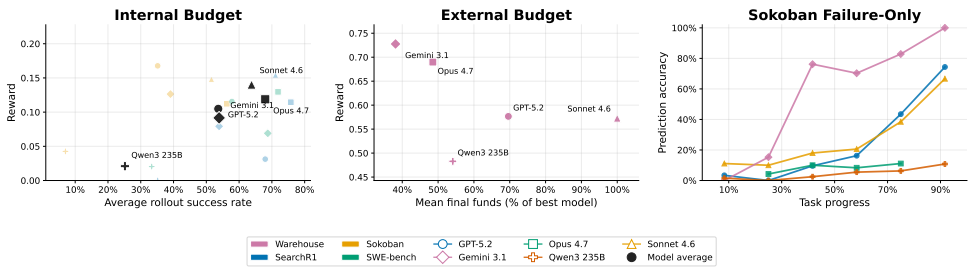

Figure 3: Left and middle: estimation quality is only weakly related to task performance for both internal and external budgets. Right: on failed Sokoban trajectories, estimation accuracy increases as more task progress is observed, with the largest gains appearing later in the trajectory.

will spend, not how cheaply the task could be done. We do not clip or invalidate intervals whose upper bound exceeds the remaining cap.

The target is also *continuation-dependent*. A prefix does not determine one immutable future cost because the same stochastic agent may continue differently. Rollout-replay scores the continuation that is actually realized—the cost a deployed agent would incur—so hit rate, aggregated over many prefixes, is the marginal coverage of the agent's stated range over realized continuations. In App. G.8, we branch shared prefixes into multiple continuations to measure how much variation comes from the sampled continuation and whether predicted width responds to it.

3.3 Sub-Capability Metrics

A budget estimator must do three things: decide whether the task can succeed under budget, detect failure early, and give a tight range covering the realized cost when success is possible.

(1) Feasibility prediction. *Can the agent tell whether the task will succeed under the remaining budget?* This is the coarsest, binary level of budget awareness. Let $y_k \in \{\text{feasible}, \text{impossible}\}$ be the ground-truth label at turn k (whether the agent succeeds within the budget), and $\hat{y}_k \in \{[\cdot,\cdot],\ \texttt{impossible}\}$ the model's output. We treat any interval as a "feasible" prediction and report

$$\text{Macro-}F_1 = \tfrac{1}{2}\big(F_1^{\text{feasible}} + F_1^{\text{impossible}}\big), \tag{2}$$

computed over the full trajectory (*all-turn*) or the first turn only (*first-turn*).

(2) Early failure detection. *For tasks that ultimately fail, can the agent recognize that early?* This is critical for budget control: catching failure early makes early stopping and budget saving possible. Restricted to the positive class $y_k = \texttt{impossible}$,

$$\text{Fail-}F_1 = \frac{2 \cdot \text{Prec}_{\text{imp}} \cdot \text{Rec}_{\text{imp}}}{\text{Prec}_{\text{imp}} + \text{Rec}_{\text{imp}}}. \tag{3}$$

A high Fail-F_1 means the alarm fires when failure is real and stays silent on feasible trajectories.

(3) Remaining budget estimation. *When the task will finish within budget, can the agent predict that and give an accurate cost range?* Restricted to feasible prefixes (the rollout solves the task within budget), a prediction earns

$$S_k = \mathbf{1}\big[R_k \in [\hat{R}_k^{\text{lo}}, \hat{R}_k^{\text{hi}}]\big] \cdot \max\Big(0,\ 1 - \tfrac{\hat{R}_k^{\text{hi}} - \hat{R}_k^{\text{lo}}}{R_k}\Big), \tag{4}$$

and an `impossible` prediction earns 0. Reward averages S_k over feasible prefixes; hit rate is the fraction whose interval covers R_k. Throughout, API failures are excluded and unparsable outputs count as incorrect. The first factor is *coverage* (1 iff the realized R_k falls in the predicted interval); the second is *tightness*, where a perfect prediction $[R_k, R_k]$ scores 1 and an interval as wide as R_k itself scores 0. We fix no nominal coverage level, but the prompt asks for an interval that covers the realized cost, which only a high-coverage range achieves reliably; observed coverage of 9–58% (Table 1) falls short of any such level. Because Eq. 4 rewards narrow intervals that happen to cover, we treat hit rate and midpoint error as primary and the reward as a summary. A centrality term does not improve median midpoint error (App. G.6). We also report the median midpoint relative error, $|\frac{\hat{R}_k^{\text{lo}} + \hat{R}_k^{\text{hi}}}{2} - R_k| / R_k$.

Table 1: Overall rollout and budget-estimation results. Success is the fraction of rollouts that solve the task; a prefix is feasible only if its rollout also stays within budget. Hit and Reward are computed over feasible prefixes (Eq. 4); rollout-clustered 95% CIs are in App. G.2. F1@1 and F1@All denote first-turn and all-turn feasibility macro-F_1, respectively. Fail-F_1 measures detection of `impossible` cases. Warehouse has no binary success label (n/a). Warehouse Hit requires all three intervals to cover; its Reward† averages feasibility correctness and per-dimension coverage without a width term, so it is not comparable to the token-budget Reward (Eq. 4).

	Task Performance		Feasibility Prediction			Interval Quality	
Model	**Success**	**Turns**	**F1@1**	**F1@All**	**Fail F1**	**Hit**	**Reward**
SWE-bench							
Claude Opus 4.7	**71.9%**	12.12	41.1%	37.5%	26.7%	53.8%	**0.130**
Claude Sonnet 4.6	68.8%	20.66	32.2%	37.7%	23.4%	39.6%	0.118
Gemini 3.1 Pro Preview	68.8%	37.36	39.2%	**54.0%**	**43.0%**	41.1%	0.069
GPT-5.2 Instant	57.8%	21.52	**43.5%**	40.2%	21.2%	**58.0%**	0.115
Qwen3 235B	33.3%	62.92	26.3%	19.4%	5.6%	9.0%	0.020
Search-R1							
Claude Opus 4.7	**75.8%**	1.78	39.4%	**40.5%**	5.6%	23.1%	0.114
Claude Sonnet 4.6	71.1%	1.87	37.9%	33.3%	0.0%	**36.5%**	**0.154**
GPT-5.2 Instant	68.0%	3.69	**40.2%**	38.3%	0.0%	21.4%	0.031
Gemini 3.1 Pro Preview	53.9%	2.09	24.5%	24.8%	0.0%	20.7%	0.079
Qwen3 235B	35.2%	9.94	23.6%	16.2%	**16.9%**	0.0%	0.000
Sokoban							
Claude Opus 4.7	**56.2%**	5.04	44.8%	45.2%	15.1%	**46.6%**	0.112
Claude Sonnet 4.6	51.6%	5.65	**46.4%**	53.6%	33.9%	45.1%	0.148
Gemini 3.1 Pro Preview	39.1%	5.63	23.9%	**59.6%**	**73.7%**	23.9%	0.126
GPT-5.2 Instant	35.2%	9.02	27.9%	40.6%	32.8%	36.2%	**0.168**
Qwen3 235B	7.0%	10.77	4.0%	6.1%	7.6%	16.0%	0.043
Warehouse (Reward†)							
GPT-5.2 Instant	n/a	12.00	35.0%	63.4%	56.9%	24.7%	0.577
Claude Opus 4.7	n/a	12.00	33.3%	63.2%	55.7%	35.9%	0.690
Claude Sonnet 4.6	n/a	12.00	33.3%	64.9%	59.1%	17.4%	0.573
Gemini 3.1 Pro Preview	n/a	12.00	**42.6%**	**68.0%**	**63.5%**	**52.3%**	**0.728**
Qwen3 235B	n/a	12.00	34.7%	58.1%	51.9%	17.3%	0.483

3.4 Experimental Setup

Internal-budget environments. **Sokoban** (Junghanns & Schaeffer, 1998): a planning task where agents push boxes to target locations on a 8×8 grid, capped at 2,500 tokens. **Search-R1** (Jin et al., 2025): multi-hop information retrieval, capped at 3,500 tokens; a short-horizon regime (1.8–3.7 turns for frontier models). **SWE-bench** (Jimenez et al., 2023): agents resolve GitHub issues under a 160-turn cap; feasibility uses a per-model median token budget that balances labels (App. B.1).

External-budget environment. **Warehouse** (App. C): a supply-chain simulation informed by real distributor demand data, with three coupled budget dimensions (cost, time, warehouse occupancy). We evaluate it with balanced (50/50) *challenge-conditioned feasibility probes*: budgets are set exogenously, as by a finance team, and the agent judges its trajectory against them (App. B.4).

Training and Evaluation We evaluate five frontier models: GPT-5.2 Instant (OpenAI, 2026), Claude Opus 4.7 (Anthropic, 2026a), Claude Sonnet 4.6 (Anthropic, 2026b), Gemini 3.1 Pro (Google, 2026), and Qwen3-235B (Yang et al., 2025). We use Qwen2.5-7B-Instruct (Qwen et al., 2025) for SFT and RL, and use a combined reward for RL to prevent collapse (Appendix E).

Scale. We generate 128 rollouts per model on Sokoban, Search-R1, and Warehouse, and 64 rollouts on SWE-bench. Each non-terminal turn yields one estimation sample via the rollout-replay protocol.

4 Task Success Is a Loose Proxy for Estimation Quality

Across a wide capability range, task success and estimation quality co-vary; among frontier models, they come apart.

4.1 Frontier Estimators Differ, but Not in Order of Success

Success ranks models only across the capability range. Including Qwen3-235B, the weakest actor, success and hit rate correlate strongly within environments (r=0.81, p=0.001); among the four frontier models, only weakly (r=0.33, p=0.35; Figures 4 and 3).

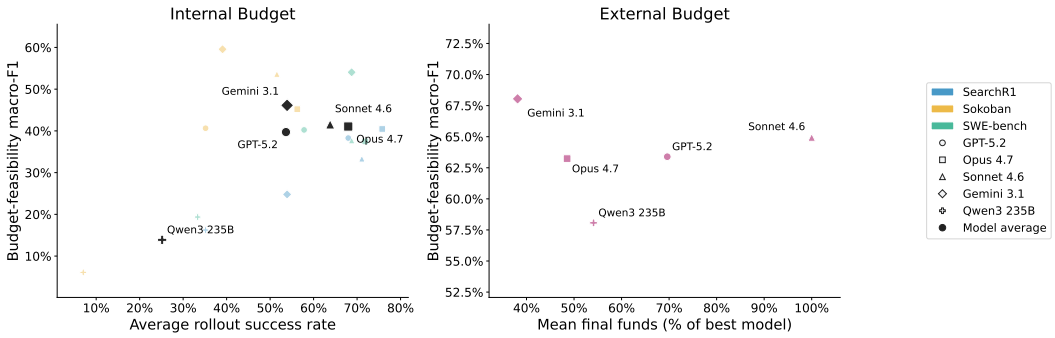

Figure 4: Progressive interval estimation evaluates an agent's budget-estimation ability separately from its task performance. The weak relationship between task success and estimation quality shows that doing the task well does not mean knowing how much budget remains.

Frontier models differ, but not in order of success. A pre-specified permutation test, which shuffles whole rollouts across models, finds that frontier hit rates differ on Sokoban (p=0.048) and SWE-bench (p=0.038), less clearly on Search-R1 (p=0.14; combined p=0.011). Success orders these differences only loosely: of 17 non-tied frontier pairs, 5 are reversed. On SWE-bench, GPT-5.2 has the lowest frontier success (57.8%) but the highest hit rate (58.0%, vs. 39.6% for Sonnet and 41.1% for Gemini at 68.8% success); across all 18 pairwise tests these two reversals are the strongest (q=0.085, Benjamini–Hochberg; App. G.1).

Simple rules match feasibility judgments. Feasibility F_1 is weak in absolute terms: a constant predictor matches or beats the model in 9 of 15 internal-budget cells, and a supervised one-feature rule, which predicts feasible while the fraction of budget spent stays below a threshold fit by rollout-grouped cross-validation, matches or beats every model (App. G.2). Models underuse even their own spending history. Removing turn count and per-turn usage from GPT-5.2's Sokoban prompt leaves its feasibility F_1 at this floor (0.406 vs. 0.406) but widens its intervals from 522 to 593 tokens (p=0.032; App. G.5), and self-rated confidence and interval width correlate weakly with the estimation score ($|r|\leq$0.23 over 10 model-task pairs; App. G.7).

Self-estimation is the target. Each model estimates its own rollouts because a deployed agent must forecast the trajectory it will actually run; cross-model differences therefore include the trajectories each model produces. Agents also beat a linear-extrapolation baseline $\widehat{R}_{\text{lin}} = (C_k/k)T - C_k$ on Warehouse occupancy in 23 of 25 model-progress bins, most strongly early in the rollout, while extrapolation stays competitive for cost (App. Figure 9).

4.2 No Model Dominates All Sub-Capabilities Across Environments

We decompose budget awareness into three sub-capabilities: feasibility prediction (binary), early failure detection, and interval estimation. Table 1 reports them across all model-environment pairs.

No single model leads on all three. Gemini leads every Warehouse metric and achieves the highest feasibility and failure-detection F_1 on Sokoban (59.6%, 73.7%), yet has the lowest frontier hit rate there (23.9%). On SWE-bench, GPT-5.2 has the best hit rate (58.0%) but the lowest frontier Fail-F_1 (21.2%). Each model has its own profile: when it judges feasibility correctly, when it raises infeasibility alarms, and how tight its intervals are on successful tasks.

Feasibility is necessary but not sufficient. Feasibility F_1 and interval hit rate are moderately related (r=0.38 across the 20 pairs): correct feasibility judgments often come with intervals that miss.

5 Why Does Budget Estimation Fail?

Task success does not explain estimation quality among frontier models, so we ask why estimation fails. We test four hypotheses:

- **Hypothesis 1 (capability gap).** Models cannot predict (a) feasibility or (b) sharp intervals.
- **Hypothesis 2 (optimistic prior).** Models have a general bias to underestimate remaining budget.
- **Hypothesis 3 (static estimation).** Models cannot leverage execution outcomes to update estimates.
- **Hypothesis 4 (late recognition).** Models detect failure only after most of the budget is spent.

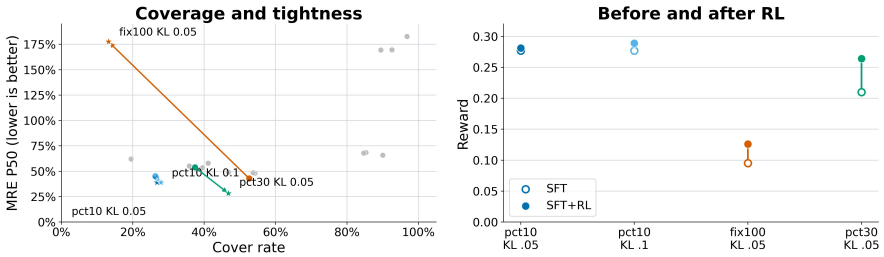

Figure 5: Training tradeoffs for interval estimation. Left: coverage versus midpoint error for SFT checkpoints and their SFT+RL continuations. Right: reward before and after RL from the same SFT starts. RL improves estimation only when it starts from a suitable SFT initialization; without an appropriate SFT warm-start, training collapses.

The tests separate what models can and cannot do. Hypotheses 2 (in aggregate and in 15 of 20 model-environment pairs) and 4 hold. Hypothesis 1a is rejected, since binary feasibility is highly trainable; Hypothesis 1b holds, since training trades coverage against sharpness. Hypothesis 3 is rejected: estimates update, though not always toward the truth.

5.1 Training Feasibility and Intervals (Hypothesis 1)

We train Qwen-7B on Sokoban budget estimation using supervised fine-tuning (SFT) followed by reinforcement learning (RL). If Hypothesis 1 holds as an immutable capability gap, fine-tuning should leave performance largely unchanged; otherwise, training should close the gap (App. E).

Training probe. We use a 6×6/one-box Sokoban probe, the hardest regime with enough successful Qwen2.5-7B trajectories for a balanced training set (success@1 14.84%, vs. 0.78% on the main 8×8/two-box setting; Appendix G.3).

Binary feasibility is highly trainable, and the gain is not only formatting. The base Qwen-7B reaches 25.5% balanced feasibility accuracy; SFT raises it to 91.1%, with no RL needed. Because the base model almost never predicts `impossible`, perfectly repairing every malformed output leaves impossible recall at 4.7% and caps balanced accuracy at 52.4%, so the remaining 38.7 points cannot come from format repair (Appendix G.4). Hypothesis 1 is rejected for binary feasibility.

Coverage is bought with width. The base model covers 10.5% of remaining budgets. Trained checkpoints trace a coverage–sharpness frontier set by the SFT target width (Figure 5; App. Table 5): none of the 22 exceeds 55% coverage at a median midpoint error below 50%, and reaching 85% coverage requires fixed 500-token targets with 68% median error. The best balance, SFT+RL from `pct30`, covers 47% at 28% median error. These SFT targets set width by rule (proportional or fixed), so the trained estimator inherits a width policy that carries no uncertainty information. Frontier models, which receive no such targets, show the same pattern: in a branching control, their predicted width tracks task scale but not the spread of realized continuation costs (App. G.8).

5.2 Optimism Dominates Across Models and Tasks (Hypothesis 2)

On every interval prediction for feasible prefixes (the samples scored by Eq. 4) across the four environments, we record whether the predicted interval misses the realized remaining budget on the optimistic side (predicted budget below realized) or the conservative side (predicted budget above realized). Hypothesis 2 predicts that optimistic misses dominate uniformly.

Most models underestimate remaining budget. Aggregated per model, optimistic misses outnumber conservative ones for all five models (56–92% of misses; Figure 2). Per model-environment pair, optimism dominates in 15 of 20. Because this compares the direction of misses, it does not depend on a coverage level: an unbiased central interval misses equally often on both sides at any level. Midpoints agree: on Search-R1 and SWE-bench the median midpoint falls below the realized cost in 9 of 10 model cells (-22% to -85%). Sokoban is the exception: the four frontier models' midpoints exceed it ($+8\%$ to $+45\%$), consistent with a planning task whose remaining moves are visible on the board. Hypothesis 2 holds on search and coding but not on Sokoban.

The bias is not explained by task difficulty alone. Within an environment, weaker models are *more* optimistic about finishing the task, not less (Spearman $\rho=-0.96$ between success rate and optimistic

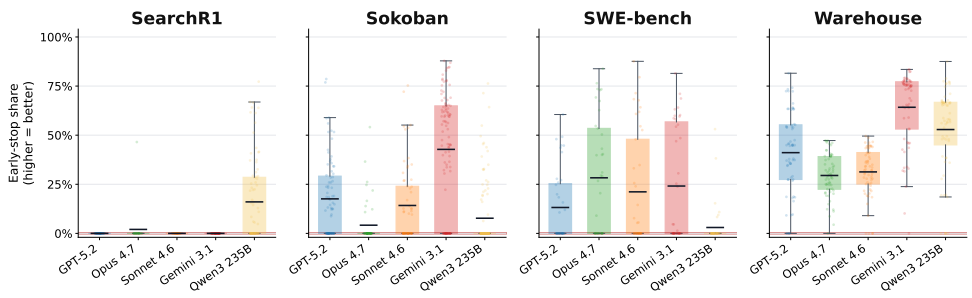

Figure 6: Failure is recognized late: across environments, models often label failed trajectories as `impossible` only after much of the token budget has already been spent.

share of misses on SWE-bench; the same direction holds on Search-R1 and Sokoban). The pattern is consistent with overconfidence under limited self-awareness.

5.3 Estimates Update, but Not Always Toward the Truth (Hypothesis 3)

For each model and environment, we compare feasibility macro-F_1 from the first-turn prediction with macro-F_1 from all-turn predictions. If Hypothesis 3 holds, the two should be similar, because the model cannot use mid-execution evidence; otherwise, they should differ.

Later turns produce different estimates than early ones, but not always better. On Sokoban, Gemini's macro-F_1 improves by $+35.6$ points from first-turn to all-turn evaluation. On SWE-bench, Qwen drops by 7.0 points. Across the 20 model-environment pairs in Table 1, all-turn F_1 is higher than first-turn F_1 in 14 pairs and lower in 6 (App. Figure 13). Hypothesis 3 is rejected: estimates update with execution, and the updates improve feasibility F_1 in 14 of 20 pairs.

5.4 Failure Is Recognized Too Late to Act On (Hypothesis 4)

On trajectories that ultimately fail (the agent does not complete the task within budget), we measure when within the trajectory the model first predicts `impossible`. Hypothesis 4 predicts that the alarm fires only late, after most of the budget has been spent. The right panel of Figure 3 shows this pattern on Sokoban: failure prediction accuracy improves as the rollout progresses, but the largest gains appear only after substantial budget has already been consumed.

Models keep predicting "feasible" long after the task is doomed. Across the 849 failed internal-budget rollouts, only 18% are flagged `impossible` before 60% of the budget is spent and 29% before 80%; the rest are flagged later or never (Figure 6). Accuracy naturally rises as the budget runs out; the failure is the absence of an early signal. Hypothesis 4 is supported. **This late recognition wastes substantial compute**, which we quantify in §6.1.

5.5 Discussion

The four tests separate budget-awareness failures into two regimes. The first is *decision*: binary feasibility is highly trainable (Hypothesis 1a rejected) and late recognition (Hypothesis 4 supported) reflects the model's failure to convert partial evidence into an actionable output. SFT closes most of the binary feasibility gap, and the late-recognition gap is large enough that even a coarse offline early-stop policy recovers much of the wasted compute. The second regime is *estimation*: sharp interval estimation (Hypothesis 1b supported), persistent optimism (Hypothesis 2 supported), and failure to allocate width according to continuation uncertainty (App. G.8) reflect deeper limitations in how models reason about budget.

Decision-side failures are directly tractable: early stopping acts on them (§6.1). Estimation-side failures call for training: moving the coverage–sharpness frontier requires RL from a matched SFT start (§6.2).

6 Acting on and Training the Budget Signal

6.1 Offline Early Stopping

On each logged trajectory, we simulate stopping at the first turn whose replayed estimate is `impossible`, trading solved tasks lost to false aborts against compute saved on doomed runs.

Table 2: Offline early stopping. Each cell: tokens saved on unsolved rollouts (%) / solved tasks stopped (points), ignoring the budget; [bracketed]: savings of a spending rule tuned in hindsight to stop no more solved tasks.

	GPT-5.2	Opus 4.7	Sonnet 4.6	Gemini 3.1	Qwen3 235B
Sokoban	19.0 / 3.9 [6.3]	4.3 / 0.0 [0.0]	14.8 / 4.8 [2.0]	56.9 / 18.5 [42.8]	11.0 / 0.8 [9.6]
Search-R1	0.0 / 0.0 [0.0]	2.6 / 0.0 [0.0]	0.0 / 1.6 [16.1]	0.0 / 1.2 [0.0]	17.5 / 0.0 [16.1]
SWE-bench	19.5 / 6.2 [8.2]	29.1 / 21.9 [46.5]	45.2 / 17.2 [51.7]	50.7 / 12.5 [40.4]	7.0 / 1.7 [13.8]

The signal carries information beyond spending. Stopping at the first `impossible` saves up to 57% of tokens on unsolved rollouts (Table 2). Against a rule that stops once a fixed fraction of the budget is spent, tuned in hindsight to lose no more solved tasks, the model saves more in 9 of 15 cells and less in 4, winning on every Sokoban model (e.g., 19.0% vs. 6.3% for GPT-5.2). The cost depends on the budget: on SWE-bench, where the budget is each model's median cost, most stopped solved rollouts already exceed it, so they count as failures under the budget but cost 6–22 points of unconstrained task success. Search-R1 rollouts end before the alarm fires.

Why offline. Rollout-replay fixes the trajectory distribution, so the savings are attributable to the estimator alone. Feeding estimates back into the acting agent would change its subsequent actions and entangle estimator quality with policy adaptation.

6.2 Training Dynamics: Width, Epoch, and Initialization

Three choices set where a trained estimator lands (App. Figure 12). SFT target width sets the coverage–tightness point: 10%-width targets give 26% coverage at 45% midpoint relative error (MRE), and 1,000-token targets about 90% at 170% MRE. More SFT epochs trade optimism for conservatism (impossible-recall rises from 77% to 87%). RL improves coverage and precision only from `pct30` SFT; it plateaus from `pct10` and loses coverage from `fix100` (52.6% to 13.2%; Figure 5, right). These results come from a single probe and seed.

7 Related Work

Budgets and self-monitoring in LLM agents. Prior work studies constrained decision making (Achiam et al., 2017; Liu et al., 2022; Mazumdar et al., 2024), cost-aware LLM agents (Liu et al., 2026; Ding et al., 2026; McCleary & Ghawaly, 2026), and adaptive computation (Graves, 2016; Chen et al., 2020; Zhou et al., 2026; Snell et al., 2024; Muennighoff et al., 2025). Other work covers self-assessment (Kadavath et al., 2022; Xiong et al., 2023; Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2023) and long-horizon failures (Wang et al., 2024b; Barke et al., 2026; Yao et al., 2022b; Zhou et al., 2023; Ma et al., 2019a). We instead test mid-run budget estimation (App. A).

Prediction intervals and calibration. Prediction-interval work studies coverage, sharpness, and conformal validity (Lei et al., 2018; Romano et al., 2019; Amini et al., 2020; Levi et al., 2022). Unlike static predictors, our remaining-budget distribution evolves along the trajectory. Output-length prediction for scheduling (Zheng et al., 2023) and token-budget-aware reasoning (Han et al., 2024) estimate or control the length of a single response; we estimate the remaining cost of a multi-turn trajectory.

8 Conclusion and Limitations

We formalize budget awareness as *progressive interval estimation* and evaluate five frontier models on four budget environments. Task success is a loose proxy for estimation quality: frontier models differ significantly as estimators, but not in order of success. Estimation fails in structured ways: underestimation on search and coding, late failure detection, and widths that track task scale rather than uncertainty. The infeasibility signal carries information beyond spending, and binary feasibility is learnable in-domain, whereas interval estimation is task-specific. A prefix counts as infeasible when this agent, continuing from it, fails to finish within budget. Our intervals carry no nominal level, so hit rate measures marginal coverage over realized continuations; conditional coverage at a fixed level requires several continuations per prefix (App. G.8). Closed-loop stopping, where estimates change what the agent does next, is the natural next step.

AI USE STATEMENT

We used generative AI assistants (Claude and ChatGPT family models) to support several aspects of this work: (1) aiding and polishing the writing of specific manuscript passages; (2) retrieval and discovery of relevant literature and related work; (3) research ideation and experiment execution, including assistance with boilerplate code, data processing, and evaluation workflows; and (4) generation of synthetic data used in our experiments. All AI-assisted outputs were reviewed, verified, and revised by the human authors. The authors made all final decisions regarding the research design, experimental methodology, analysis, interpretation, and conclusions, and take full responsibility for the final content of this submission.

ETHICS STATEMENT

This work involves no human subjects and no personal data. The Warehouse environment is built from real enterprise supply-chain data; all company, supplier, and product identifiers were replaced with pseudonyms before use, and no raw identifiers appear in prompts, trajectories, or released artifacts (App. K). Budget awareness is meant to reduce wasted compute and resources. Because an estimator that aborts tasks can also stop work that would have succeeded, we report false-abort rates alongside every savings figure (§6.1).

REPRODUCIBILITY STATEMENT

Environments, budget caps, and the rollout-replay protocol are specified in App. B.1 and App. B.2; Warehouse dynamics, data processing, and probe construction (seed 42) in App. C and App. B.4; exact model identifiers and inference settings in App. B.6; all estimation prompts in App. J; and SFT/RL data, reward, and hyperparameters in App. E. Because frontier APIs change over time, every reported metric is computed from logged rollouts and estimation outputs rather than from live API calls, and all metrics and statistical tests can be recomputed from the released estimation outputs (App. G.2).

REFERENCES

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pp. 22–31. Pmlr, 2017.

Alexander Amini, Wilko Schwarting, Ava Soleimany, and Daniela Rus. Deep evidential regression. *Advances in neural information processing systems*, 33:14927–14937, 2020.

Anthropic. Adaptive thinking. Claude API Documentation, 2026a. URL `https://platform.claude.com/docs/en/build-with-claude/adaptive-thinking`. Accessed 2026-04-29.

Anthropic. Introducing claude sonnet 4.6. Anthropic News, 2026b. URL `https://www.anthropic.com/news/claude-sonnet-4-6`. Accessed 2026-04-29.

Shraddha Barke, Arnav Goyal, Alind Khare, Avaljot Singh, Suman Nath, and Chetan Bansal. Agentrx: Diagnosing ai agent failures from execution trajectories. *arXiv preprint arXiv:2602.02475*, 2026.

Nicolas Carrara, Edouard Leurent, Romain Laroche, Tanguy Urvoy, Odalric-Ambrym Maillard, and Olivier Pietquin. Budgeted reinforcement learning in continuous state space. *Advances in neural information processing systems*, 32, 2019.

Xinshi Chen, Hanjun Dai, Yu Li, Xin Gao, and Le Song. Learning to stop while learning to predict. In *International conference on machine learning*, pp. 1520–1530. PMLR, 2020.

Youngseog Chung, Willie Neiswanger, Ian Char, and Jeff Schneider. Beyond pinball loss: Quantile methods for calibrated uncertainty quantification. *Advances in Neural Information Processing Systems*, 34:10971–10984, 2021.

Shumin Deng, Ningyu Zhang, Nay Oo, and Bryan Hooi. Towards a unified view of answer calibration for multi-step reasoning. In *Proceedings of the 2nd Workshop on Natural Language Reasoning and Structured Explanations (@ ACL 2024)*, pp. 25–38, 2024.

Wenxuan Ding, Nicholas Tomlin, and Greg Durrett. Calibrate-then-act: Cost-aware exploration in llm agents. *arXiv preprint arXiv:2602.16699*, 2026.

Google. Gemini 3.1 pro preview. Google AI for Developers, 2026. URL `https://ai.google.dev/gemini-api/docs/models/gemini-3.1-pro-preview`. Accessed 2026-04-29.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware LLM reasoning. *arXiv preprint arXiv:2412.18547*, 2024.

Zhiyuan Hu, Chumin Liu, Xidong Feng, Yilun Zhao, See-Kiong Ng, Anh Tuan Luu, Junxian He, Pang Wei Koh, and Bryan Hooi. Uncertainty of thoughts: Uncertainty-aware planning enhances information seeking in large language models. *arXiv preprint arXiv:2402.03271*, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Andreas Junghanns and Jonathan Schaeffer. Sokoban: Evaluating standard single-agent search techniques in the presence of deadlock. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pp. 1–15. Springer, 1998.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, et al. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*, 2022.

Sanyam Kapoor, Nate Gruver, Manley Roberts, Katherine Collins, Arka Pal, Umang Bhatt, Adrian Weller, Samuel Dooley, Micah Goldblum, and Andrew G Wilson. Large language models must be taught to know what they don't know. *Advances in Neural Information Processing Systems*, 37: 85932–85972, 2024.

Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*, pp. 3301–3310. PMLR, 2019.

Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. Accurate uncertainties for deep learning using calibrated regression. In *International conference on machine learning*, pp. 2796–2804. PMLR, 2018.

Jing Lei, Max G'Sell, Alessandro Rinaldo, Ryan J Tibshirani, and Larry Wasserman. Distribution-free predictive inference for regression. *Journal of the American Statistical Association*, 113(523): 1094–1111, 2018.

Dan Levi, Liran Gispan, Niv Giladi, and Ethan Fetaya. Evaluating and calibrating uncertainty prediction in regression tasks. *Sensors*, 22(15):5540, 2022.

Donghao Li, Ruiquan Huang, Cong Shen, and Jing Yang. Near-optimal conservative exploration in reinforcement learning under episode-wise constraints. In *International Conference on Machine Learning*, pp. 19527–19564. PMLR, 2023.

Dancheng Liu, Amir Nassereldine, Ziming Yang, Chenhui Xu, Yuting Hu, Jiajie Li, Utkarsh Kumar, Changjae Lee, Ruiyang Qin, Yiyu Shi, et al. Large language models have intrinsic self-correction ability. *arXiv preprint arXiv:2406.15673*, 2024.

Hanbing Liu, Chunhao Tian, Nan An, Ziyuan Wang, Pinyan Lu, Changyuan Yu, and Qi Qi. Budget-constrained agentic large language models: Intention-based planning for costly tool use. *arXiv preprint arXiv:2602.11541*, 2026.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

Zuxin Liu, Zhepeng Cen, Vladislav Isenbaev, Wei Liu, Steven Wu, Bo Li, and Ding Zhao. Constrained variational policy optimization for safe reinforcement learning. In *International Conference on Machine Learning*, pp. 13644–13668. PMLR, 2022.

Chih-Yao Ma, Jiasen Lu, Zuxuan Wu, Ghassan AlRegib, Zsolt Kira, Richard Socher, and Caiming Xiong. Self-monitoring navigation agent via auxiliary progress estimation. *arXiv preprint arXiv:1901.03035*, 2019a.

Chih-Yao Ma, Zuxuan Wu, Ghassan AlRegib, Caiming Xiong, and Zsolt Kira. The regretful agent: Heuristic-aided navigation through progress estimation. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pp. 6732–6740, 2019b.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Potsawee Manakul, Adian Liusie, and Mark Gales. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 9004–9017, 2023.

Abhijit Mazumdar, Rafal Wisniewski, and Manuela L Bujorianu. Safe reinforcement learning for constrained markov decision processes with stochastic stopping time. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pp. 1923–1928. IEEE, 2024.

Kyle McCleary and James Ghawaly. Quantifying the accuracy and cost impact of design decisions in budget-constrained agentic llm search. *arXiv preprint arXiv:2603.08877*, 2026.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

OpenAI. GPT-5.2 Instant Update. OpenAI Help Center, 2026. URL `https://help.openai.com/en/articles/6825453-chatgpt-release-notes`. Accessed 2026-04-29.

Tim Pearce, Alexandra Brintrup, Mohamed Zaki, and Andy Neely. High-quality prediction intervals for deep learning: A distribution-free, ensembled approach. In *International conference on machine learning*, pp. 4075–4084. PMLR, 2018.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL `https://arxiv.org/abs/2412.15115`.

Yaniv Romano, Evan Patterson, and Emmanuel Candes. Conformalized quantile regression. *Advances in neural information processing systems*, 32, 2019.

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.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, pp. 2464–2469. IEEE, 2016.

Chen Tessler, Daniel J Mankowitz, and Shie Mannor. Reward constrained policy optimization. *arXiv preprint arXiv:1805.11074*, 2018.

Akifumi Wachi and Yanan Sui. Safe reinforcement learning in constrained markov decision processes. In *International Conference on Machine Learning*, pp. 9797–9806. PMLR, 2020.

Haoyu Wang, Tao Li, Zhiwei Deng, Dan Roth, and Yang Li. Devil's advocate: Anticipatory reflection for llm agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 966–978, 2024a.

Renxi Wang, Haonan Li, Xudong Han, Yixuan Zhang, and Timothy Baldwin. Learning from failure: Integrating negative examples when fine-tuning large language models as agents. *arXiv preprint arXiv:2402.11651*, 2024b.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 409–424, 2018.

Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms. *arXiv preprint arXiv:2306.13063*, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022b.

Gal Yona, Roee Aharoni, and Mor Geva. Can large language models faithfully express their intrinsic uncertainty in words? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7752–7764, 2024.

Dewen Zeng, Nan Du, Tao Wang, Yuanzhong Xu, Tao Lei, Zhifeng Chen, and Claire Cui. Learning to skip for language modeling. *arXiv preprint arXiv:2311.15436*, 2023.

Liyuan Zheng and Lillian Ratliff. Constrained upper confidence reinforcement learning. In *Learning for Dynamics and Control*, pp. 620–629. PMLR, 2020.

Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. Response length perception and sequence scheduling: An LLM-empowered LLM inference pipeline. In *Advances in Neural Information Processing Systems*, 2023.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Xiaofan Zhou, Huy Nguyen, Bo Yu, Chenxi Liu, and Lu Cheng. Adaptive stopping for multi-turn llm reasoning. *arXiv preprint arXiv:2604.01413*, 2026.

A Extended Related Work

Budgeted decision making versus budget estimation. Prior work on constrained and budgeted sequential decision making typically formulates budget as an external constraint or environment-provided cost signal, and studies how to maximize reward while satisfying safety, resource, or episode-wise constraints Achiam et al. (2017); Tessler et al. (2018); Carrara et al. (2019); Wachi & Sui (2020); Zheng & Ratliff (2020); Liu et al. (2022); Li et al. (2023); Mazumdar et al. (2024). Recent LLM-agent work further introduces explicit resource limits into agentic systems, including token budgets, tool-use limits, monetary API costs, adaptive stopping rules, budget-aware planning, and cost-aware exploration Zhou et al. (2026); Liu et al. (2026); Ding et al. (2026); McCleary & Ghawaly (2026). However, these studies primarily ask how an agent should act when a budget is known, whereas our work asks whether the agent can estimate the remaining budget required to complete a task from an intermediate trajectory state, with uncertainty-aware intervals and infeasibility warnings rather than only realized cost or point forecasts Romano et al. (2019); Kuleshov et al. (2018). This distinction is especially important for deployed agents that must manage multiple coupled resources, including model-side internal budgets such as tokens and tool calls and environment-side external budgets such as money, time, inventory, and warehouse capacity. It shows why budget estimation should be evaluated as a standalone capability, rather than inferred from task success or total budget use. Zhou et al. (2026); Liu et al. (2026); Ding et al. (2026); McCleary & Ghawaly (2026).

Prediction intervals, calibration, and uncertainty-aware budget estimation. Prediction-interval and calibration work provides the closest methodological precedent because it studies uncertainty-aware prediction: given an input x, the predictor outputs an interval $[\hat{y}_{\text{low}}, \hat{y}_{\text{high}}]$ that should contain the true target y with a desired probability, while also remaining as sharp as possible Lei et al. (2018); Pearce et al. (2018); Kuleshov et al. (2018); Romano et al. (2019); Amini et al. (2020); Chung et al. (2021); Levi et al. (2022). Our setting also uses interval-valued predictions. However, the prediction target is fundamentally different. Given a trajectory prefix $\tau_{\leq t}$, the agent must estimate how much additional budget is needed to complete the task from the current state. This remaining requirement depends on partial progress, accumulated mistakes, environment feedback, and the agent's future continuation behavior. Thus, the target is not a static supervised label attached to an input, but an online, trajectory-dependent quantity that changes as the agent acts; standard calibration assumptions for fixed predictors on static data therefore only partially capture the difficulty of budget estimation Lei et al. (2018); Romano et al. (2019); Levi et al. (2022).

Moreover, interval coverage alone is insufficient for deployed agents. A remaining-budget interval may cover the realized cost, yet still be operationally poor if it is too wide, systematically optimistic, or fails to warn that the task has become infeasible under the available budget. For example, an agent that predicts a broad interval only after exhausting most of its budget is less useful than one that identifies infeasibility early and recommends stopping, requesting more resources, or changing strategy. We therefore evaluate uncertainty-aware budget estimation not only through interval hit rate, but also through feasibility prediction and early infeasibility warning, treating budget estimation as an online self-assessment capability rather than a standard offline calibration problem.

Adaptive compute as control rather than budget estimation. Adaptive-compute and test-time scaling methods are closely related in that they explicitly reason about computational resources, but their objective is usually to learn a compute-control policy rather than to evaluate an agent's self-estimation ability. Early-exit, dynamic-routing, learning-to-stop, and test-time scaling methods decide whether to halt, continue, route to a different module, or allocate more inference compute in order to improve the performance–cost tradeoff Graves (2016); Teerapittayanon et al. (2016); Wang et al. (2018); Kaya et al. (2019); Chen et al. (2020); Zeng et al. (2023); Snell et al. (2024); Muennighoff et al. (2025). Such decisions can be effective without requiring the model to explicitly state how much additional budget is needed, how uncertain that estimate is, or whether task completion remains feasible under the remaining budget. Moreover, this line primarily focuses on internal computation, such as layers, tokens, samples, reasoning steps, or inference-time compute, whereas deployed agents also incur external action-induced costs through tool calls, API usage, elapsed time, monetary spending, inventory changes, or other environment commitments. Our setting therefore treats remaining-budget estimation as a distinct capability: the agent must expose an uncertainty-aware belief about future resource requirements across both model-side computation and environment-side action costs, rather than merely choosing a stopping or routing action.

Self-monitoring and corrective reasoning. A line of work studies whether language models can assess their own correctness, express uncertainty, recognize when they do not know an answer, or

Similar Work	Agentic setting	Trajectory-level	Online during execution	Interval / calibrated UQ	Token budget	Financial budget	Endogenous future cost	Failure-specific analysis
Can LLMs Express Their Uncertainty? Xiong et al. (2023)	✗	✗	✗	∼	✗	✗	✗	∼
Conformalized Quantile Regression Romano et al. (2019)	✗	✗	✗	✓	✗	✗	✗	✗
Adaptive Stopping for Multi-Turn LLM Reasoning Zhou et al. (2026)	∼	✓	✓	✓	∼	✗	∼	✗
Budget-Constrained Agentic LLMs (INTENT) Liu et al. (2026)	✓	✓	✓	✗	∼	✓	✓	✗
Calibrate-Then-Act Ding et al. (2026)	✓	✓	✓	✗	✗	✗	∼	✗
Budget-Constrained Agentic Search (BCAS) McCleary & Ghawaly (2026)	✓	✓	✓	✗	✓	✗	∼	✗
Our work	✓	✓	✓	✓	✓	✓	✓	✓

Figure 7: Delta map between our formulation and the closest prior work. Rows are representative neighboring papers; columns are the main components of the problem definition. ✓ means the component is directly modeled, ∼ means partially related, and ✗ means largely absent.

detect inconsistencies in generated content Kadavath et al. (2022); Xiong et al. (2023); Kapoor et al. (2024); Yona et al. (2024); Deng et al. (2024); Manakul et al. (2023); Hu et al. (2024). Related methods use reflection, self-feedback, intrinsic critique, or tool-based criticism to improve model outputs and agent performance over multiple attempts Madaan et al. (2023); Shinn et al. (2023); Gou et al. (2023); Liu et al. (2024); Wang et al. (2024a). These works are closely related because they treat self-assessment as a core model capability, but their focus is usually correctness awareness or corrective reasoning: whether the current answer is likely correct, whether the model should revise it, or how feedback can help produce a better next response. Budget awareness asks a different question: even if the agent can judge or improve its answer, can it estimate how much resource is still needed to complete the task, how uncertain that estimate is, and whether completion remains feasible under the available budget?

Trajectory-level agent evaluation and budget self-monitoring. Interactive agent benchmarks provide the long-horizon, tool-mediated settings in which budget estimation becomes meaningful, since agents must reason, act, replan, and use memory over trajectories whose costs accumulate across turns Yao et al. (2022b;a); Zhou et al. (2023); Liu et al. (2023); Shridhar et al. (2020). Recent failure-analysis work further treats failure as a trajectory-level phenomenon, studying root causes, negative rollouts, and where agents fail rather than only final success labels Wang et al. (2024b); Barke et al. (2026). Closest in spirit, progress-monitoring methods use auxiliary estimates of task progress to guide search, backtracking, and action choice Ma et al. (2019a;b). Our focus is different from both retrospective failure diagnosis and progress estimation: given a partial trajectory, we ask whether the agent can estimate the remaining token or financial budget, express uncertainty, and warn that completion may be infeasible under the available resources. This exposes selective over-optimism on failed rollouts, a failure mode not captured by standard success, path-efficiency, or progress metrics.

The central gap highlighted by Figure 7 is that prior work typically covers only one or two sides of the space at a time. Our work is positioned at the intersection of these dimensions and formulates online remaining-budget interval estimation as a distinct capability of agents.

B Detailed Experimental Settings

B.1 Environments and Tasks

Our benchmark spans four environments, SearchR1, Sokoban, and SWE-bench use token budgets; Warehouse uses a joint financial budget(cost/time/inventory) over multiple resources.

Sokoban. We use a Sokoban environment to study budget estimation under irreversible planning. The rollout agent solves procedurally generated 8×8 puzzles with two boxes and search depth 30 under a 2,500-token budget. For rollout generation, we cap each response at 800 tokens and truncate dialogue history at 2,500 context tokens.

SearchR1. We use the SearchR1 environment backed by a HotpotQA-derived parquet dataset and a retrieval server. At each turn, the agent must either issue one search query or submit a final answer

under a 3,500-token budget. Frontier models finish in 1.8–3.7 turns on average, so Search-R1 is a deliberately short-horizon regime: it tests whether an agent can estimate its cost when little trajectory has been observed.

SWE-bench SWE-bench evaluates coding agents on realistic GitHub issue-resolution tasks. We include it because coding agents are common in practice, and coding tasks are token-intensive. Their token usage often comes from repeated repository inspection, targeted edits, test execution, error analysis, and repair loops. This makes SWE-bench a natural benchmark for evaluating remaining-budget estimation. Rollouts run under a 160-turn cap. Because coding trajectories have no natural token cap, feasibility labels use a token budget set per rollout family at the median realized token count of that family; each model therefore faces roughly balanced labels, and SWE-bench comparisons are relative to each model's own budget.

Warehouse. We use a warehouse-management control environment with coupled resource constraints. At each step, the agent must allocate inventory, financing, and cash-flow decisions so the business can continue toward a profitable outcome without violating limits. This benchmark tests a different notion of budget awareness: instead of internal reasoning cost, the agent must estimate whether it can still finish while respecting external operational budgets. Detailed budget construction appears in Section B.4.

Token accounting. Each environment counts the tokens its budget constrains. Sokoban and Search-R1 caps are context budgets, so a turn's cost is its input plus output tokens. SWE-bench agents resend a long, largely cached history every turn; counting it would make cost quadratic in turn count and dominated by repetition, so a turn's cost excludes history reused from earlier turns. The estimation prompt states each environment's accounting explicitly (App. J), and token rewards are compared only within an environment.

B.2 Evaluation Protocol

Estimator and rollout roles are separated in all experiments. Each rollout is first generated and logged by a rollout model, and the estimator is then evaluated on prefixes of that logged trajectory. In the main experiments, the estimator uses the same model as the rollout generator, corresponding to self-estimation.

At each evaluation point, we replay the rollout prefix as dialogue history and ask the estimator to predict the budget needed from the next turn onward. The estimator only sees information available up to the current turn, including the replayed history, the budget limits, and summaries of completed turns. For token-budget tasks, these summaries include per-turn token usage; for Warehouse, they include cumulative progress and resource usage so far. The estimator does not receive future turns, future tool outputs, terminal success labels, or the realized remaining budget. We also exclude the final terminal step from estimation, since there is no future budget to predict after the trajectory has ended.

The estimator must output either an interval `<answer>[est_low, est_high]</answer>` for the remaining total token usage from the next turn onward, or `<answer>impossible</answer>` if it predicts that the trajectory can no longer finish successfully within the budget. For Warehouse, the estimator receives the target cash threshold and three resource budgets, `time_weeks`, `warehouse_item_weeks`, and `cumulative_cost_usd`, together with cumulative usage so far. It must predict whether the trajectory can still reach the target cash while satisfying all three constraints; if feasible, it outputs one interval for each remaining resource, otherwise it outputs impossible.

Each non-terminal rollout prefix yields one evaluation sample. If a trajectory has T turns, it contributes $T-1$ samples, because the final turn is not estimated. This construction preserves the online estimation setting: the estimator always predicts from a partial trajectory using only information available at that point.

SWE-bench requires additional subsampling because coding-agent trajectories are much longer than those in the other benchmarks. In the current SWE-bench runs, this produces 712–3,715 candidate prefixes per rollout family, so using all prefixes would make long coding trajectories dominate the evaluation set. We therefore cap each SWE-bench rollout family at 512 estimation prefixes. To construct this split, we sort rollouts by their number of assistant turns, partition them into eight nearly equal-sized length buckets, and sample prefixes with a seeded randomized round-robin strategy across buckets and rollouts using random seed 42. This procedure buckets at the rollout level rather than the prefix level, so longer rollouts do not receive proportionally more sampling mass simply

because they contain more prefixes. This makes the split fair because it balances coverage across rollout-length regimes and rollout instances, preventing unusually long trajectories from contributing a disproportionate number of highly correlated prefixes while keeping the SWE-bench evaluation size comparable to the other benchmarks.

B.3 Pilot Study Setup

The pilot study in §2 establishes the two failures of single-point budget estimation that motivate progressive interval estimation. We describe its setup here.

Tasks. We use two internal-budget environments. **Sokoban** (Junghanns & Schaeffer, 1998) is a planning task on an 8×8 grid with a $2{,}500$-token cap; it admits a clean ground-truth optimal budget per puzzle. **Search-R1** (Jin et al., 2025) is a multi-hop information-retrieval task with a $3{,}500$-token cap; the realized budget is variable and driven by retrieval depth. The two tasks are deliberately different in shape, so a failure showing up on both is unlikely to be an artifact of one task's structure.

Elicitation. For each rollout we elicit two estimates from the same model. A *first-turn estimate* is taken at task start, before any action, asking the model to predict the total tokens it will spend on the task; this matches the single-point prompting protocol used in prior work. A *later-turn estimate* is taken after replaying the logged prefix $\tau_{\leq k}$ of the same rollout, asking the model to predict the *remaining* tokens from turn $k{+}1$ onward. The later-turn estimate is therefore a sequence of single-point predictions, one per non-terminal turn, made by the same model that produced the rollout.

Models and scale. We evaluate the same five frontier models as in the main paper: Claude Opus 4.7 (Anthropic, 2026a), Claude Sonnet 4.6 (Anthropic, 2026b), GPT-5.2 Instant (OpenAI, 2026), Gemini 3.1 Pro (Google, 2026), and Qwen3-235B (Yang et al., 2025). We sample 128 rollouts per (model, task) pair.

Scoring. *Optimism analysis* (Figure 2, first failure): for each prediction $\hat{B}$ we compare it to the realized rollout cost B and classify it as *optimistic* ($\hat{B} < B$) or *conservative* ($\hat{B} > B$); we report the share of each across rollout-progress bins. *First-turn vs. later-turn comparison* (Figure 13, second failure): we collapse each single-point prediction into a binary feasibility label by checking whether $\hat{B}$ stays under the budget cap, and compute feasibility macro-F_1 over either first-turn predictions only or all per-turn predictions (definition in Eq. 2). The two are then plotted against each other on equal axes.

B.4 Warehouse Budget Construction

Warehouse is evaluated as a multi-resource financial estimation task that models a manufacturing firm making weekly operational decisions, such as producing goods, replenishing inventory, drawing supplier credit, repaying debt, and factoring accounts receivable to improve cash flow. Full details on the underlying data source, design goals, calibration choices, and reward structure are deferred to Appendix C. For each rollout prefix, the estimator receives the dialogue history, current cash, completed-week summaries, cumulative resource usage, and per-step resource consumption so far. It must predict whether the rollout can still reach the target final cash threshold while staying within all resource budgets. If feasible, it outputs one interval for each remaining resource; otherwise, it outputs `impossible`. This setting creates coupled trade-offs: increasing inventory may improve sales but raises warehouse occupancy, drawing credit may improve short-term cash but adds repayment pressure, and delaying production may reduce costs but lower final cash. Thus, Warehouse tests whether estimators can reason about realistic multi-resource constraints rather than a single token budget.

Warehouse contains three budget dimensions. The first is `time_weeks`, which limits the total planning horizon. The second is `warehouse_item_weeks`, which measures cumulative warehouse occupancy over time, i.e., inventory integrated across weeks. The third is `cumulative_cost_usd`, which limits total operational spending.

Why we construct feasibility probes. Warehouse is a continuous-optimization task (more cash is better, fewer resources used is better) and therefore has no natural binary success label. To evaluate budget awareness in this environment, we construct *challenge-conditioned feasibility probes*: each instance pairs a logged rollout with a sampled (target cash, time, warehouse, cost) tuple, and feasibility is defined as whether that rollout still satisfies the target and all three resource constraints. We balance reachable and unreachable probes 50/50 so that macro-F_1, Fail-F_1, and calibration are identifiable

Benchmark	Budget Type	Rollouts	Horizon	Estimation Target
SearchR1	Token	128	Up to 3500 tokens; one search/action per turn	Remaining tokens within 3,500.
Sokoban	Token	128	Up to 2500 tokens; up to 3 actions per turn	Remaining tokens within 2,500.
SWE-bench	Token	64	up to 160 turns	Remaining tokens under rollout-family-specific median budgets.
Warehouse	Financial	128	2 weeks per turn; totally 11 turns	Remaining time, warehouse item-weeks, and cost while reaching target cash.

Table 3: Experiment matrix for the four evaluation benchmarks. Each row states the budget modality, rollout count, horizon constraint, and remaining-budget target predicted from each logged prefix.

on both sides; under a heavily skewed split (e.g., 90/10), a model that always predicts "feasible" would score deceptively well. The thresholds are exogenous by design. In deployment, budgets are usually set by someone other than the agent (e.g., a cost ceiling fixed by finance), and the agent must judge its own trajectory against a limit it did not choose; sampling thresholds around realized totals concentrates probes near the feasibility boundary, where that judgment is hardest. This 50/50 balance is an evaluation-design choice, not a claim about deployment prevalence: we do not assert that real supply-chain decisions fail half the time, only that controlled coverage of clearly-feasible, borderline-feasible, borderline-infeasible, and clearly-infeasible regions is needed to test whether models know their own state.

Sampling procedure. The estimator uses the `half_reachable` budget preset with random seed 42. We first shuffle rollouts and assign half to a reachable group and half to an unreachable group. For reachable rollouts, the target cash is sampled from $U(0.50, 1.00)\times$ final cash, clipped so that it does not exceed final cash; the time budget is set to the realized trajectory length, and the warehouse and cost budgets are sampled uniformly between $1.0\times$ and $1.2\times$ their realized final totals. For unreachable rollouts, the target-cash, time-budget, warehouse-budget, and cost-budget ratios are independently sampled from $U(0.50, 2.00)$ until the logged rollout fails the feasibility check. Thus unreachable cases may be caused by an overly high target cash, too-tight resource budgets, or a combination of both. Because budgets are generated relative to each rollout's realized totals, some regularities hold across the probe set: reachable probes have a time budget equal to the realized horizon, and their resource budgets lie within 1.0–$1.2\times$ realized totals. These are properties of the generator and are not observable from a single probe, so they do not affect zero-shot evaluation.

Scoring. Time advances by a fixed two weeks per turn, so remaining time is determined by the horizon and serves as a sanity dimension; occupancy and cost carry the estimation difficulty. An interval prediction counts as a hit only if all three intervals cover their realized remaining values. The Warehouse reward in Table 1 is the mean of feasibility correctness and the three per-dimension coverage indicators; it has no width term and is therefore not comparable to the token-budget reward of Eq. 4.

B.5 Experiment Matrix

The experiment matrix is shown in Table 3.

B.6 Model Identifiers and Inference Settings

To ensure reproducibility, we report exact deployment metadata for every model used in rollout generation and estimation. The display names in the main text are shorthand; the appendix table gives the exact API model identifier and invocation configuration.

For each model, we log display name, provider, exact API model id or local checkpoint hash, date queried, endpoint or region, reasoning configuration, max output tokens, temperature, top-p if used, stop settings, and sampling-seed policy.

The term "Low Thinking" denotes a constrained provider-native reasoning configuration. Depending on the backend, this corresponds to controls such as `reasoning_effort=low`, `output_config.effort=low`, or `thinking_mode=low`. The appendix table reports the exact control and value for each model.

Table 4 lists the concrete entries used in this paper.

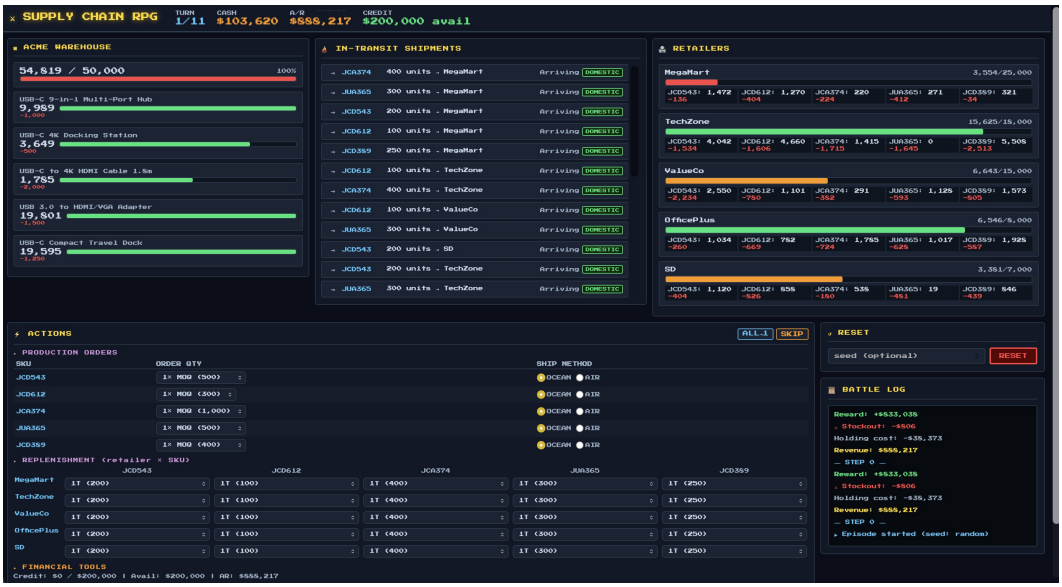

Figure 8: Overview of the Warehouse environment. The agent operates a manufacturing firm that orders from OEMs, ships internationally to its own warehouse, and fulfills weekly demand from five retailer accounts under joint money/space/time constraints.

C Warehouse Environment Details

C.1 Data Source

The demand panel is sourced from a real mid-sized US consumer-electronics distributor (anonymized as *Acme*) and its five downstream retail accounts, collapsed into archetypes (*MegaMart*, *TechZone*, *ValueCo*, *OfficePlus*, *SupplyDirect*) covering mass discount, electronics specialty, value/household, office/B2B, and small-account direct channels. We obtained **22 consecutive weeks** of unit-level weekly sell-through per (retailer, SKU) pair across five high-volume product families (USB-C hub, 4K dock, USB-C/HDMI cable, USB display adapter, travel dock). Manufacturers and order quantities are *not* in the dataset—those are agent decisions in our environment. Wholesale prices, retailer warehouse capacities, payment terms (Net-30 across the board), and stockout/overstock chargeback rates were taken from retail vendor agreements with light rounding; manufacturing costs, MOQs, production lead times, and unit weights were chosen to reflect typical consumer-electronics OEM contract terms. We did not synthesize or smooth the demand series—week-to-week variance, step-changes, and the occasional collapsing series (e.g., one *OfficePlus* SKU) are present in the raw data.

At each episode reset the five demand series belonging to a retailer are randomly permuted across the five game SKUs, so SKU identities do not inherit fixed demand profiles and the agent must read demand from the observation rather than from any learned prior over individual SKUs. Per-step base demand is the sum of two consecutive weeks of the assigned series with additive Gaussian noise scaled by the series standard deviation. We do not layer additional seasonality on top—the 22-week window already captures it.

C.2 Design Goal: Stressing Money, Space, and Time

A competent policy must trade off three orthogonal pressures at once: **money** (tight $500K initial cash, 30/70 split mfg payment, Net-30 receivables, holding + OpEx burn, a $200K credit line at 0.8% per step, and 5% AR factoring), **space** (50,000-unit Acme warehouse plus per-retailer DCs of 7K–25K units, with overstock chargebacks above 95% occupancy), and **time** (production lead 25–45 days, ocean vs. air international transit at 32 vs. 6 days, domestic 3–4 days, Net-30 cash collection). Every mechanism is anchored to at least one of these axes.

C.3 Calibration Decisions

Two parameters depart from default settings for the reasons below; the rest are anchored to real-world values.

Initial inventory is set to 0. Earlier calibrations pre-stocked Acme and the retailers with several weeks of expected demand. Trajectories were then largely shaped by the starting conditions rather than by the policy: pre-stocked inventory absorbed the early turns of demand regardless of what the agent did, and roughly 3 of 11 turns ran on autopilot before policy choices began to matter. We now start every (retailer, SKU) pair and the Acme warehouse at **zero units** so that outcomes are driven by model decisions end-to-end and the early-episode cash crunch (no AR matured yet, no inventory to ship) is genuine.

International transit and production lead times are compressed. Real Shenzhen→US-West-Coast door-to-door shipping is 45–60 days ocean and 7–10 days air; OEM production for the more complex SKUs realistically runs 60–90 days once material procurement is included. Plugged in directly, a single ocean order placed at $t = 0$ would arrive around day $90 + 60 = 150$—turn 11 of an 11-turn episode at our 14-day step granularity, meaning 4–5 turns elapse before any production decision surfaces. We instead use ocean $= 32 \pm 4$ days, air $= 6 \pm 1$ days, and production lead $= 25$–45 days by SKU. The fastest end-to-end loop (cable, air) then lands in ≈ 2 steps and the slowest (dock, ocean) in ≈ 5.5 steps—short enough that the agent observes consequences within the episode, long enough that time pressure is load-bearing.

Other calibration choices, briefly. Initial cash is \$500K (not \$2M) so the working-capital decision actually binds—at ∼\$116K weekly GMV and Net-30 terms, ∼\$464K of cash is tied up in any one AR cycle. The mfg payment is a **30/70 split** (deposit at order, balance on completion) rather than 100%-on-order, so cash flow is sensitive to the projected trajectory across the production horizon and orders can stall in queue if cash is short on completion. Production quantities are quantized to **1×/2×/3× MOQ** with 0% / 8% / 15% volume discounts, mirroring how factories actually quote. Episode length is **22 weeks ÷ 14 days per step = 11 steps**—short enough that endgame myopia is diagnosable, long enough for two full ocean-production cycles. Bankruptcy is a **two-stage floor** (\$50K per-step penalty whenever cash < 0, hard truncation at cash $< -\$200$K) rather than a single hard rule, so a small early-turn underprediction does not end the episode prematurely.

C.4 Reward and Termination

The per-step reward is

$$r_t \;=\; \pi_t \;-\; p_t^{\text{stockout}} \;-\; p_t^{\text{overstock}} \;-\; 50{,}000 \cdot \mathbf{1}[\text{cash}_t < 0], \tag{5}$$

where π_t is within-step operating profit (revenue minus holding cost minus fixed OpEx), and p_t^{stockout}, $p_t^{\text{overstock}}$ are the chargeback penalties from unfilled retailer demand and from >95%-occupied retailer DCs. Manufacturing costs, international and domestic shipping costs, credit interest, and AR-factoring fees affect cash but **do not** enter the reward—they are investments and financing costs whose payoff arrives, if at all, through future operating profit. This is the central credit-assignment challenge: a production decision at t pays back through revenue at $t+4$ to $t+6$. Including these costs in the reward collapses the environment to a near-myopic control problem and removes precisely the long-horizon coordination that motivated the design. The episode terminates at $t = T$ (default 11) and is force-truncated if cash drops below $-\$200$K.

C.5 Theoretical Maximum Reward (Defined but Unused)

For completeness we derive a hypothetical ceiling on cumulative reward under an idealized policy that (i) realizes mean demand each step with no noise, (ii) keeps Acme inventory exactly equal to the step's total demand so holding cost is minimal but no stockouts occur, and (iii) faces no cash constraint. Under those assumptions, period revenue, holding cost, and OpEx all become deterministic, and the cumulative ceiling is

$$R_{\max} \;=\; \sum_{t=1}^{T} \left[\sum_{r,s} d_{r,s} \cdot p_{r,s} \cdot w \;-\; h \cdot I_t \cdot w \;-\; \text{OpEx} \cdot w \right], \tag{6}$$

where $d_{r,s}$ is mean weekly demand of SKU s at retailer r, $p_{r,s}$ is the wholesale price, w is the number of weeks per step (default 2), $h = \$0.35$/unit/week is the holding rate, $I_t = \sum_s d_s \cdot w$ is the per-step ideal Acme warehouse level (just enough to cover the step's demand), and OpEx $= \$8{,}000$/week is the fixed operating cost. A "combined score" variant adds back ending cash and the manufacturing-cost value of ending inventory at both Acme and the retailers, treating leftover stock as recoverable capital.

Display Name	YAML Key	Provider	Model ID	Reasoning	Max Tokens	Temp.
GPT-5.2 Instant	OpenAI-5.2-Instant	openai	`gpt-5.2`	`reasoning_effort: none`	800	N/A
Claude Opus 4.7 Low Thinking	Claude-Opus-4.7-low-thinking	anthropic	`claude-opus-4-7`	`output_config.effort: low`	800	N/A
Claude Sonnet 4.6 Low Thinking	Claude-Sonnet-4.6-low-thinking	anthropic	`claude-sonnet-4-6`	`output_config.effort: low`	800	N/A
Gemini 3.1 Pro Preview	OpenRouter-Gemini-3.1-Pro-Preview	openrouter	`google/gemini-3.1-pro-preview`	`thinking_mode: low`	800	0
Qwen3 235B	qwen/qwen3-235b-a22b-2507	openrouter	`qwen/qwen3-235b-a22b-2507`	`thinking_mode: low`	800	0

Table 4: Model deployment identifiers and inference settings used for rollout generation and budget estimation. The table maps each display name to its provider, exact model ID, reasoning configuration, maximum output length, and temperature.

D Additional Experimental Visualizations

This appendix presents analysis for budget-estimation behavior. Figure 11 separates the internal token-budget setting from the external Warehouse setting. For internal tasks, interval width is normalized by the total token budget. For Warehouse, we report the same quantity for time, warehouse item-weeks, and cost. Estimators adjust interval width as rollout evidence accumulates.

Figure 10 analyzes the direction of estimation errors across rollout progress. Across models, optimistic misses dominate conservative misses, meaning that agents more often underestimate the remaining budget than overestimate it. This bias persists throughout execution, although conservative misses become more visible in later rollout stages. Together, these results show that budget-estimation errors are not symmetric: models are systematically biased toward thinking the remaining task will be cheaper than it actually is.

Figure 4 further separates task success from budget-estimation quality. Across benchmarks and models, higher rollout success does not necessarily imply better feasibility prediction. This shows that budget awareness is not simply a byproduct of stronger task-solving ability, but a distinct capability that must be evaluated directly.

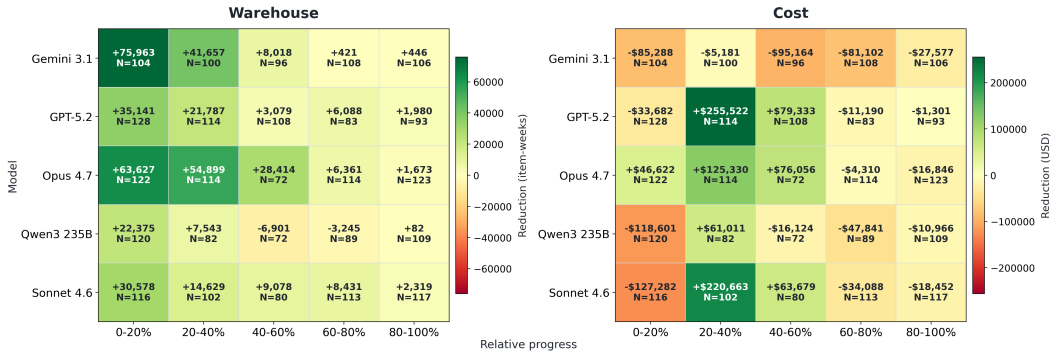

Figure 9: Agent midpoint estimates versus linear extrapolation on Warehouse, where positive cells indicate lower absolute error for the agent than for the extrapolation baseline.

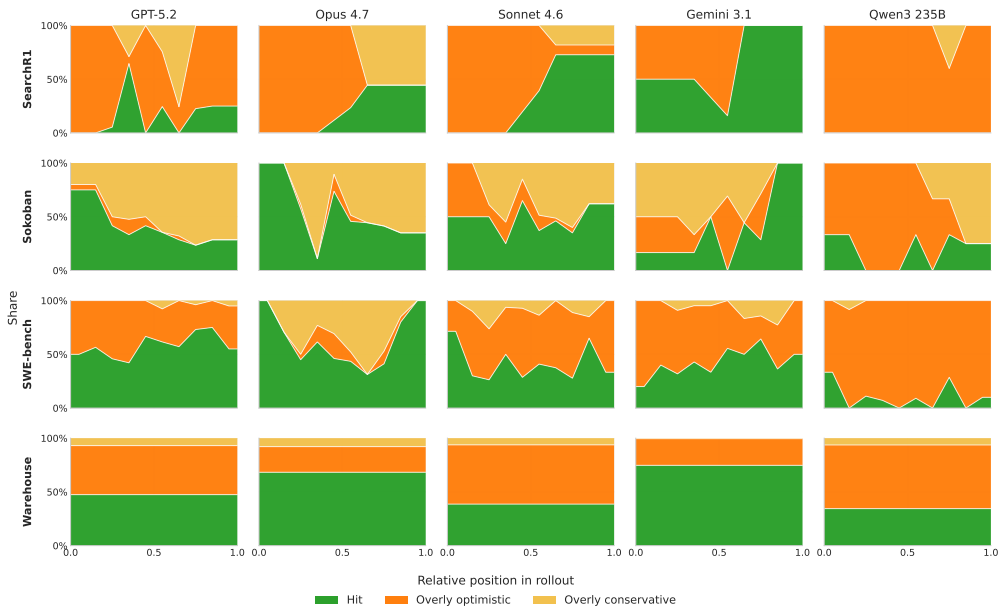

Figure 10: Optimistic misses dominate budget estimates. Orange regions (predicted budget below realized) exceed yellow (predicted budget above realized) in 15 of 20 model-environment pairs; the exceptions are the four frontier models on Sokoban and Opus on SWE-bench.

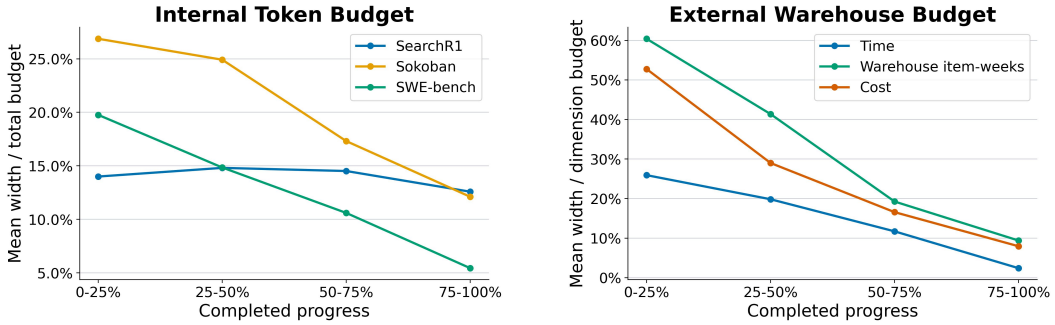

Figure 11: Predicted interval width changes over task progress. Internal-task widths are normalized by token budget; Warehouse reports separate normalized widths for time, warehouse item-weeks, and cost.

E Detailed Settings for Estimation RL and SFT

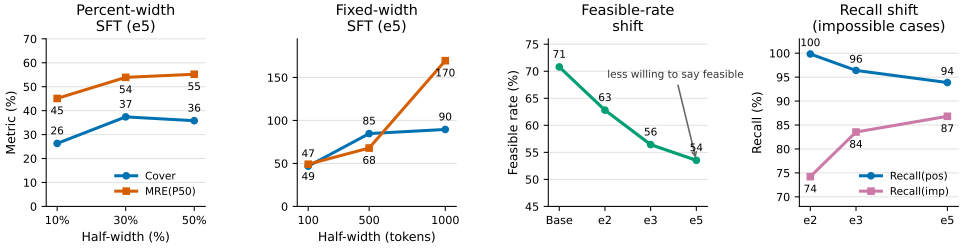

Figure 12: SFT choices control the behavior of the Sokoban budget estimator. Wider interval targets improve coverage but increase midpoint error, while longer SFT makes the model more conservative by reducing feasible predictions and improving recall on impossible cases.

The SFT and RL experiments use Qwen/Qwen2.5-7B-Instruct as the base model. The probe task is Sokoban 6x6 with one box and a 2500-token budget. The SFT training set contains 794 samples. The balanced test set contains 380 samples, split evenly between possible and impossible states.

The raw rollouts are first converted into alternating user/assistant message histories. For each trajectory, a probe is inserted before the first action and after every completed turn. At probe-after-turn t, the prompt contains the Sokoban system prompt and the conversation prefix up to t. For $t = 0$, the prompt contains the system prompt and the initial grid only. The completed-turn token usage is computed from the visible user messages and the recorded API output-token counts, so that post-processed assistant text does not undercount hidden generation cost. The true remaining-token label r is the sum of all future assistant output tokens and all future user input tokens after the visible prefix. A probe is labeled possible only if the original rollout succeeds and the visible-prefix tokens plus r fit within the budget; otherwise it is labeled `impossible`. The ablation data use a deterministic trajectory split with seed 42: 40% for SFT, 50% for RL, and 10% for held-out evaluation. Training splits are balanced with a 1:1 possible/impossible ratio after duplicating possible samples twice; trivial samples with $r = 0$ are removed.

SFT targets. SFT turns each probe into a full chat record by appending the ground-truth assistant answer. The main runs use the no-thinking output template, so the assistant target is either `<answer>impossible</answer>` or `<answer>[L, H]</answer>`. For a possible probe with remaining-token label r, the percentage-width targets are

$$L = \max(1, \lfloor r(1-w) \rfloor), \qquad H = \lfloor r(1+w) \rfloor,$$

with $w \in \{0.1, 0.3, 0.5\}$. The fixed-width targets are

$$L = \max(1, \lfloor r - w \rfloor), \qquad H = \lfloor r + w \rfloor,$$

with $w \in \{100, 500, 1000\}$ tokens. We also prepared a point-estimation SFT variant that predicts one integer, but the reported RL interval experiments warm start from interval SFT checkpoints. All SFT runs use FSDP training for 5 epochs with learning rate 5×10^{-6}, global batch size 16, and per-GPU micro-batch size 2. The default SFT script sets maximum sequence length to 9216, while the v3 batch launcher overrides it to 16384. Checkpoints are converted after epochs 2, 3, and 5.

Training-name convention. Run names encode the supervision target and checkpoint used for evaluation. `pct10`, `pct30`, and `pct50` denote interval SFT targets whose half-width is 10%, 30%, or 50% of the true remaining-token label. `fix100`, `fix500`, and `fix1000` denote interval SFT targets with fixed half-widths of 100, 500, or 1000 tokens. The suffixes `e2`, `e3`, and `e5` identify checkpoints after 2, 3, and 5 SFT epochs. An entry such as `SFT pct30 e5 → RL KL 0.05` means that the epoch-5 `pct30` SFT checkpoint initializes GRPO, with actor KL coefficient 0.05. `Zero RL` denotes GRPO started directly from the base model, without SFT warm start. `no think` and `+ think` distinguish whether the output template omits or includes a `<think>...</think>` field. `scalar bug` denotes an invalid zero-RL run that learned scalar outputs instead of the required interval format.

RL setup. RL is applied with GRPO on top of selected SFT checkpoints. The RL parquet records contain the same prompt without the assistant answer, together with the rule-reward ground truth and the numeric `remaining_tokens` field. The final GRPO runs use 8 GPUs, global batch size 64, 16 rollouts per prompt, learning rate 5×10^{-7}, 5 epochs, maximum prompt length 8192, maximum response length 1024, entropy coefficient 0, and no KL term inside the reward. The actor still uses a low-variance KL loss against the reference policy; the reported sweep uses KL coefficients 0.05 and 0.1. Evaluation decodes with vLLM at temperature 0 and a 512-token response cap.

RL reward. The rule reward first extracts the content inside `<answer>...</answer>`. Outputs without this tag, malformed intervals, reversed intervals, and scalar predictions receive zero reward for possible probes. Impossible probes receive reward 0.2 only when the extracted answer is exactly `impossible`. For a possible probe with true remaining tokens r, an interval $[L, H]$ receives reward only when $L \le r \le H$:

$$R = 1.8 \cdot \max\left(0, 1 - \frac{H - L}{r}\right).$$

This reward favors tight intervals that still cover the true value. The 1.8-to-0.2 weighting discourages the all-impossible policy while keeping impossible-state recognition in the objective.

F Additional Data for RL

Table 5 reports the full SFT/RL sweep. The zero-RL runs show that RL from scratch does not reliably learn the interval format. Both no-thinking and thinking variants collapse to all-impossible answers. The scalar-bug run has high classification accuracy, but it does not produce valid intervals. These results show that SFT is needed before RL.

Table 5: Sokoban budget-probe results for the SFT and RL sweep. Acc. is feasible/impossible classification accuracy; Rec. pos/imp are recalls for feasible and impossible probes; Cover is interval coverage on feasible probes; MRE is midpoint relative error; R is the combined reward.

Model	Acc.	Rec. pos	Rec. imp	Cover	MRE P50	MRE P90	R
Qwen2.5-7B-Instruct	25.5%	46.3%	4.7%	10.5%	65.5%	277.2%	0.021
Zero RL scalar bug	90.5%	93.2%	87.9%	0.0%	0.0%	0.0%	0.088
Zero RL no think	50.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.100
Zero RL + think	50.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.100
SFT pct10 e2	87.6%	100.0%	75.3%	26.3%	45.1%	84.0%	0.265
SFT pct10 e3	89.5%	95.3%	83.7%	19.5%	62.0%	135.4%	0.224
SFT pct10 e5	90.3%	93.7%	86.8%	26.3%	45.1%	84.2%	0.277
SFT pct30 e2	88.2%	99.5%	76.8%	39.5%	53.5%	86.7%	0.204
SFT pct30 e3	90.0%	96.3%	83.7%	38.4%	51.8%	84.3%	0.205
SFT pct30 e5	91.1%	95.3%	86.8%	37.4%	53.9%	86.4%	0.210
SFT pct50 e2	86.6%	100.0%	73.2%	41.1%	57.9%	848.1%	0.094
SFT pct50 e3	89.7%	96.3%	83.2%	54.2%	48.0%	86.0%	0.099
SFT pct50 e5	90.5%	94.7%	86.3%	35.8%	55.2%	303.4%	0.101
SFT fix100 e2	86.6%	100.0%	73.2%	53.7%	48.6%	83.7%	0.085
SFT fix100 e3	90.5%	97.4%	83.7%	52.6%	42.9%	83.6%	0.095
SFT fix100 e5	90.0%	92.6%	87.4%	46.8%	49.1%	88.4%	0.105
SFT fix500 e2	86.8%	99.5%	74.2%	90.0%	65.9%	427.3%	0.074
SFT fix500 e3	90.0%	96.8%	83.2%	85.3%	68.4%	294.0%	0.083
SFT fix500 e5	90.3%	93.7%	86.8%	84.7%	67.9%	441.8%	0.087
SFT fix1000 e2	86.3%	100.0%	72.6%	96.8%	182.7%	620.0%	0.073
SFT fix1000 e3	90.0%	96.3%	83.7%	92.6%	169.6%	627.3%	0.084
SFT fix1000 e5	90.0%	93.2%	86.8%	89.5%	169.5%	639.7%	0.087
SFT pct10 e5 $\rightarrow$ RL KL 0.05	88.9%	90.0%	87.9%	26.8%	38.9%	84.0%	0.281
SFT pct10 e5 $\rightarrow$ RL KL 0.1	89.2%	90.5%	87.9%	27.9%	38.9%	84.0%	0.289
SFT fix100 e3 $\rightarrow$ RL KL 0.05	90.3%	93.7%	86.8%	13.2%	177.6%	350.0%	0.126
SFT pct30 e5 $\rightarrow$ RL KL 0.05	90.0%	92.6%	87.4%	46.8%	28.2%	87.8%	0.264

Notes. `SFT` denotes supervised fine-tuning; `RL` denotes the subsequent GRPO stage; and `Zero RL` denotes GRPO from the base model without an SFT warm start. `pct10`, `pct30`, and `pct50` are SFT target intervals with widths equal to 10%, 30%, or 50% of the true remaining token count. `fix100`, `fix500`, and `fix1000` are fixed-width target intervals of 100, 500, or 1000 tokens. The suffixes `e2`, `e3`, and `e5` mark checkpoints after 2, 3, and 5 SFT epochs; `KL` is the RL KL-penalty coefficient. `+ think` includes a short reasoning field before the answer, while `no think` omits it. `scalar bug` is an invalid zero-RL run that outputs scalar values instead of required intervals. `Acc.` is binary feasibility accuracy. `Rec. pos` and `Rec. imp` are recall on possible and impossible states. `Cover` is computed on truly possible states where the model outputs an interval $[L, H]$ and measures whether the true remaining count r satisfies $L \leq r \leq H$. `MRE` is midpoint relative error, $\mathrm{MRE} = |((L+H)/2) - r|/r$; `P50` and `P90` are its median and 90th percentile. `R` is the average sparse reward used for RL evaluation.

G Robustness and Mechanism Analyses

This section collects controls that distinguish budget awareness from task competence, formatting, arithmetic extrapolation, confidence reporting, and single-continuation noise. We report the complete numerical results so that the scope statements in the main paper are directly auditable.

G.1 Statistical Tests for Task-Success Decoupling

Pre-specified omnibus test. Within each internal-budget environment we test whether the four frontier models' hit rates on feasible prefixes differ. The statistic is the between-model sum of squares of hit rates, and the null distribution permutes whole rollouts across models (20,000 permutations), so

correlated prefixes of one rollout stay together. Hit rates differ on Sokoban (p=0.048) and SWE-bench (p=0.038) but not clearly on Search-R1 (p=0.143); Fisher's combination gives p=0.011.

All pairwise comparisons. Table 6 reports every frontier pair in every internal-budget environment (18 tests), rather than selected pairs. Hit-rate differences use a rollout-clustered bootstrap (20,000 resamples; two-sided), and q applies the Benjamini–Hochberg correction across all 18 tests. A pair is a reversal when the model with lower task success has the higher hit rate. Of 17 non-tied pairs, 12 are ordered consistently with success and 5 are reversed; the two strongest reversals are GPT-5.2 against Sonnet and Gemini on SWE-bench (q=0.085). No pairwise difference survives a family-wise (Holm) correction, so we rely on the omnibus test for significance and on the pairwise table for direction.

Table 6: All frontier pairwise comparisons of interval hit rate on feasible prefixes. Success is the rollout success rate (%); the hit difference is A minus B in points; q is Benjamini–Hochberg adjusted across the 18 tests. R marks reversals.

Env.	Pair (A vs. B)	Success A / B	Hit diff.	p	q	R
Sokoban	GPT-5.2 vs. Opus	35.2 / 56.2	−10.4	0.074	0.182	
Sokoban	GPT-5.2 vs. Sonnet	35.2 / 51.6	−8.9	0.142	0.226	
Sokoban	GPT-5.2 vs. Gemini	35.2 / 39.1	+12.3	0.151	0.226	✓
Sokoban	Opus vs. Sonnet	56.2 / 51.6	+1.5	0.785	0.887	
Sokoban	Opus vs. Gemini	56.2 / 39.1	+22.7	0.006	0.085	
Sokoban	Sonnet vs. Gemini	51.6 / 39.1	+21.2	0.011	0.085	
Search-R1	GPT-5.2 vs. Opus	68.0 / 75.8	−1.7	0.815	0.887	
Search-R1	GPT-5.2 vs. Sonnet	68.0 / 71.1	−15.1	0.024	0.085	
Search-R1	GPT-5.2 vs. Gemini	68.0 / 53.9	+0.7	0.906	0.906	
Search-R1	Opus vs. Sonnet	75.8 / 71.1	−13.5	0.101	0.182	✓
Search-R1	Opus vs. Gemini	75.8 / 53.9	+2.4	0.785	0.887	
Search-R1	Sonnet vs. Gemini	71.1 / 53.9	+15.8	0.093	0.182	
SWE-bench	GPT-5.2 vs. Opus	57.8 / 71.9	+4.2	0.548	0.759	✓
SWE-bench	GPT-5.2 vs. Sonnet	57.8 / 68.8	+18.4	0.015	0.085	✓
SWE-bench	GPT-5.2 vs. Gemini	57.8 / 68.8	+16.9	0.021	0.085	✓
SWE-bench	Opus vs. Sonnet	71.9 / 68.8	+14.2	0.063	0.182	
SWE-bench	Opus vs. Gemini	71.9 / 68.8	+12.8	0.092	0.182	
SWE-bench	Sonnet vs. Gemini	68.8 / 68.8	−1.4	0.837	0.887	

Correlations. Within each internal-budget environment we center task success and hit rate across models and correlate the centered values, testing with a permutation that shuffles models within environments (20,000 permutations). Across all five models the correlation is r=0.81 (p=0.001); across the four frontier models it is r=0.33 (p=0.35).

Early-stopping reference rule. The bracketed values in Table 2 use a rule that stops once the fraction of the budget spent reaches a threshold t. For each cell, t is chosen in hindsight to maximize tokens saved on unsolved rollouts while stopping no more solved rollouts than the model does, so the rule is an optimistic reference rather than a deployable baseline.

G.2 Baselines and Confidence Intervals

Table 7 reports rollout-clustered 95% bootstrap intervals (2,000 resamples) for the interval metrics of Table 1, together with two feasibility baselines. The constant baseline predicts the better of always-feasible and always-impossible for each cell. The threshold baseline predicts feasible while the fraction of the budget already spent is at most t, with t chosen on training folds of a 5-fold, rollout-grouped cross-validation; it is supervised, whereas the models are evaluated zero-shot. Every metric is recomputed from the released estimation outputs.

G.3 Training Difficulty and Scope

We evaluated greedy success@1 for Qwen2.5-7B-Instruct on 128 fixed solvable Sokoban seeds at three difficulties. The 6×6/one-box probe is the hardest tested regime with enough successful trajectories to construct a balanced possible/impossible training set. The main 8×8/two-box regime yields only one success, so a probe set drawn from those rollouts would be approximately 99% impossible.

G.4 Why the SFT Gain Is Not Only Format Repair

The base estimator is strongly biased toward feasible predictions. Repairing a malformed output can therefore add a feasible prediction and improve recall on possible probes, but it cannot increase recall on impossible probes. Even an oracle that repairs every formatting failure and makes positive recall 100% leaves impossible recall at 4.7%, so balanced accuracy is capped at $(100 + 4.7)/2 = 52.4\%$.

Table 7: Interval metrics with rollout-clustered 95% CIs, and feasibility macro-F_1 (%) of each model against a constant and a one-feature threshold baseline. Feasible prefixes are listed with rollouts in parentheses.

Env.	Model	Feasible	Hit (%)	Reward	F_1 model	F_1 constant	F_1 threshold
Sokoban	GPT-5.2	188 (45)	36.2 [27.2, 44.4]	0.168 [0.116, 0.222]	40.6	41.7	53.2
Sokoban	Opus	206 (71)	46.6 [38.5, 53.8]	0.112 [0.087, 0.136]	45.2	36.9	60.2
Sokoban	Sonnet	215 (65)	45.1 [37.1, 53.2]	0.148 [0.109, 0.191]	53.6	35.0	55.6
Sokoban	Gemini	46 (22)	23.9 [11.1, 38.6]	0.126 [0.046, 0.217]	59.6	45.9	59.8
Sokoban	Qwen3	25 (4)	16.0 [0.0, 33.3]	0.043 [0.000, 0.077]	6.1	49.4	49.2
Search-R1	GPT-5.2	201 (84)	21.4 [15.9, 27.4]	0.031 [0.017, 0.047]	38.3	38.3	53.7
Search-R1	Opus	52 (43)	23.1 [13.0, 34.5]	0.114 [0.059, 0.173]	40.5	37.4	61.2
Search-R1	Sonnet	52 (39)	36.5 [25.0, 49.0]	0.154 [0.101, 0.212]	33.3	33.8	65.3
Search-R1	Gemini	29 (27)	20.7 [7.4, 35.5]	0.079 [0.029, 0.140]	24.8	39.7	38.3
Search-R1	Qwen3	97 (41)	0.0 [0.0, 0.0]	0.000 [0.000, 0.000]	16.2	47.6	64.4
SWE-bench	GPT-5.2	200 (25)	58.0 [46.9, 68.5]	0.115 [0.091, 0.141]	40.2	37.9	70.2
SWE-bench	Opus	143 (27)	53.8 [43.9, 63.4]	0.130 [0.081, 0.178]	37.5	41.8	69.1
SWE-bench	Sonnet	164 (23)	39.6 [29.9, 50.4]	0.118 [0.082, 0.161]	37.7	40.5	65.1
SWE-bench	Gemini	190 (25)	41.1 [30.7, 50.9]	0.069 [0.047, 0.094]	54.0	37.4	73.5
SWE-bench	Qwen3	122 (15)	9.0 [4.1, 15.1]	0.020 [0.008, 0.034]	19.4	43.2	66.3

Table 8: Qwen2.5-7B-Instruct Sokoban success by difficulty. Intervals are Wilson 95% confidence intervals over 128 fixed solvable seeds.

Condition	Success@1	Wilson 95% CI
6×6, 1 box (training probe)	19/128 (14.84%)	[9.71, 22.02]
7×7, 2 boxes	3/128 (2.34%)	[0.80, 6.66]
8×8, 2 boxes (main setting)	1/128 (0.78%)	[0.14, 4.29]

SFT reaches 91.1%; the remaining 38.7-point improvement requires learning the feasibility decision rather than merely the output syntax.

Table 9: Format-only upper bound compared with the observed SFT result.

Estimator	Recall possible	Recall impossible	Balanced accuracy
Base Qwen2.5-7B	46.3%	4.7%	25.5%
Oracle format repair upper bound	100.0%	4.7%	52.4%
SFT pct30 epoch 5	95.3%	86.8%	91.1%

G.5 Trajectory-Statistic Shortcut Controls

We remove turn count and per-turn token usage from GPT-5.2's Sokoban estimation prompt while retaining the board and action history. The paired comparison uses 661 predictions from 128 rollout clusters and 5,000 cluster-bootstrap replicates. Feasibility is unchanged, while intervals widen when the arithmetic cues are removed.

A second shortcut baseline deterministically extrapolates from accumulated usage and turn count. Table 11 reports paired absolute-error reduction for Warehouse occupancy; positive values mean the agent is more accurate. Agents beat the arithmetic baseline in 23 of 25 model-progress cells, with the largest margins early in execution. The corresponding heatmap is Figure 9.

G.6 Centrality-Reward Ablation

The RL rule reward prioritizes coverage and width. We add a midpoint-centrality penalty with $\lambda \in \{0, 0.5, 1.0\}$ while holding the SFT initialization, data, seed 42, eight H100 GPUs, batch size 64, 16 samples per prompt, learning rate 5×10^{-7}, KL coefficient 0.05, and five epochs fixed. All checkpoints are cross-scored under the original reward on the same 380-row balanced test set.

Adding centrality does not improve median MRE, and $\lambda = 0.5$ is worse than the original reward on both coverage and median MRE. The $\lambda = 1$ variant improves tail-sensitive mean MRE and mean width, so centrality is a secondary refinement rather than the missing explanation for the interval-coverage ceiling.

Table 10: Paired shortcut control. Full includes turn count and per-turn token usage; NoStats removes both fields.

Metric	Full	NoStats	Δ	p
Can-finish macro-F_1	0.4060	0.4056	-0.0004	0.964
Mean interval width (tokens)	522.0	592.6	$+70.6$	0.032

Table 11: Agent absolute-error reduction over linear extrapolation for Warehouse occupancy, in item-weeks.

Model	0–20%	20–40%	40–60%	60–80%	80–100%
Gemini 3.1	75,963	41,657	8,018	421	446
Opus 4.7	63,627	54,899	28,414	6,361	1,673
GPT-5.2	35,141	21,787	3,079	6,088	1,980
Sonnet 4.6	30,578	14,629	9,078	8,431	2,319
Qwen3 235B	22,375	7,543	$-6,901$	$-3,245$	82

G.7 Confidence Controls

We append a 0–100 self-rated confidence field to the estimator prompt and rerun all ten available model-task pairs. None of these measures explains the strict score well ($|r| \leq 0.27$), so budget awareness is not reducible to reported confidence or interval width. Token log-probabilities are unavailable for four of the five frontier models because their closed APIs do not expose them.

G.8 Multi-Continuation Sensitivity and Width Allocation

A prefix does not uniquely determine future cost, since the same stochastic policy can produce several continuations. We test whether estimates read the shared prefix and whether interval width responds to the residual uncertainty.

How much variation comes from the sampled suffix? We fix three-turn prefixes and sample up to eight independent continuations. Table 14 decomposes estimator-midpoint variation into the contribution from sampled suffix identity and the shared prefix. The suffix explains 22–47% (mean 41%), leaving 53–78% associated with the prefix.

Does predicted width track empirical continuation spread? In the stricter design, 32 fixed prefixes each branch into exactly eight continuations, while the estimator predicts one interval before branching. We correlate its width with the standard deviation of realized remaining cost across the eight branches.

Heterogeneity is $I^2 = 0.000$, so the four models do not separate into trackers and non-trackers. No per-model test survives Benjamini–Hochberg correction. Controlling for task mean cost yields partial correlations of 0.000, 0.076, -0.258, and 0.195 for Opus, Sonnet, Gemini, and Qwen, respectively; none is significant. The apparent weak association is therefore explained by expensive tasks having both larger absolute spreads and wider intervals. Width tracks scale, not uncertainty.

Failed continuations end at termination, so their realized cost is a lower bound; the successful-only subset (6–19 groups per model) points the same way. Across both analyses, we find no evidence that predicted width tracks continuation uncertainty.

G.9 Evaluation-Scope Controls

Self-estimation requires estimator and rollout generator to be the same model, because a deployed agent must forecast the trajectory it will actually run.

Table 12: Centrality-reward ablation. The pre-specified median MRE is best at the original $\lambda = 0$ reward.

Reward	Accuracy	Coverage	Median MRE	Mean MRE	Mean rel. width
Original ($\lambda = 0$)	90.00%	45.79%	30.41%	56.29%	60.05%
Central ($\lambda = 1.0$)	90.00%	46.32%	31.42%	47.26%	52.03%
Central ($\lambda = 0.5$)	90.00%	44.74%	35.95%	50.10%	53.67%

Table 13: Correlation between proposed confidence measures and the strict interval score across ten model-task pairs.

Predictor	N	Pearson r
Self-rated confidence	10	$+0.115$
Mean normalized width	10	-0.227
Interval hit rate (coverage alone)	10	$+0.272$

Table 14: Multi-continuation midpoint sensitivity. GPT-5.2 is independently replicated twice.

Model	Groups	Median midpoint range	Variance from suffix
Opus 4.7	76	525 tokens	47.3%
Sonnet 4.6	82	600 tokens	45.6%
Gemini 3.1 Pro	64	275 tokens	22.1%
GPT-5.2 replicate 1	157	400 tokens	40.2%
GPT-5.2 replicate 2	159	430 tokens	45.9%
Qwen3-235B	40	250 tokens	43.0%

The early-stopping analysis is offline by design. Feeding an estimate into the actor's context would change subsequent actions, so a closed-loop result would measure the coupled estimator–policy system rather than estimator quality alone. Rollout-replay fixes the trajectory distribution and isolates the savings attributable to the estimator.

Table 15: Meta-analysis of empirical continuation spread versus predicted interval width.

Analysis	Pearson r	95% CI	p
Four-model random-effects meta-analysis	0.154	$[-0.045, 0.341]$	0.128
GPT-5.2 replication meta-analysis	0.014	$[-0.336, 0.360]$	0.941

H First-Turn versus All-Turn Feasibility

Figure 13 supports the Hypothesis 3 analysis in §5.3.

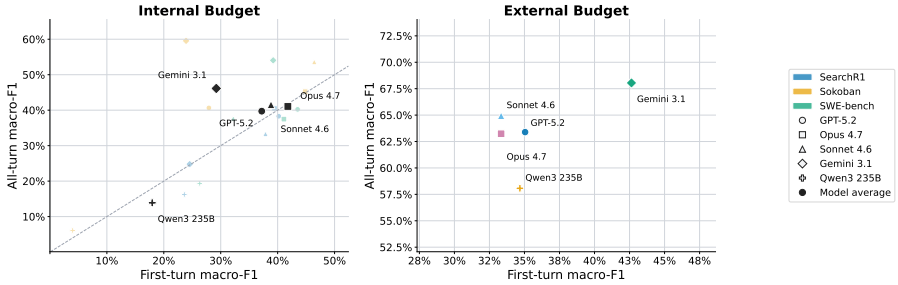

Figure 13: First-turn vs all-turn feasibility F_1. Points scatter on both sides of the equality line, so first-turn predictions do not summarize what the same agent would say after seeing partial progress.

I Extended Training Results

Training extends across tasks and model families. Table 16 extends the SFT-then-GRPO pipeline to four budget-estimation settings spanning three tasks (Search-R1, Sokoban, SWE-bench) and three model families (Qwen2.5-7B, Qwen3-4B, Llama-3.1-8B). In-task training raises the reward in every setting; on Sokoban and SWE-bench the gain comes from feasibility prediction, while Search-R1 also learns covering intervals. Across tasks, the feasibility format carries over (85–88% valid) and 17–36% of the in-task reward is retained, while interval coverage is task-specific: precise interval estimation is learned per task rather than as a general metacognitive capability.

Table 16: Extended SFT+GRPO budget-estimator results across tasks and models. In-task: trained and evaluated on the same task. Cross-task: trained on source, evaluated on target (Gain/retention shows absolute gain for in-task and retention rate for cross-task). MRE_{50} is median midpoint relative error over interval predictions.

Setting	Learner / transfer	Unit	$R_{\text{base/src}}$	$R_{\text{train/cross}}$	Gain / retention	Format	Acc.	Cover	MRE_{50}
In-task training									
Search-R1	Qwen2.5-7B	Tokens	0.029	0.258	+0.229	100.0%	56.2%	56.7%	8.8%
Sokoban	Llama-3.1-8B	Tokens	0.006	0.155	+0.149	100.0%	78.8%	0.0%	85.2%
Sokoban	Qwen3-4B	Tokens	0.002	0.143	+0.141	88.5%	73.0%	0.0%	2712.5%
SWE-bench	Qwen2.5-7B	Tokens	0.000	0.058	+0.058	52.8%	28.9%	0.0%	–
Cross-task transfer									
Sokoban → Search-R1	Qwen3-4B	Tokens	0.143	0.052	36%	87.7%	40.8%	10.5%	92.4%
Sokoban → Search-R1	Llama-3.1-8B	Tokens	0.155	0.026	17%	86.9%	44.6%	0.0%	94.4%
Search-R1 → Sokoban	Qwen2.5-7B	Tokens	0.258	0.090	35%	84.4%	50.0%	0.0%	99.6%

J Prompts for Agent Estimation and SFT/RL

This section summarizes the prompt format used for agent budget estimation. All variants follow the same structure. The prompt defines the budget unit, lists the context fields, and ends with strict output rules. The compact templates below show the key instructions.

SearchR1 token-budget prompt.

Based on the provided SearchQA rollout context, you are given the following:

1. You have completed `{completed_turns}` turns.
2. Each turn, your token consumption is `{turn_token_usage_text}`.
3. You need to finish the task within `{max_context_window_tokens}` total tokens.

Now, estimate:

1. Whether the search agent can still finish the task successfully within `{max_context_window_tokens}` total tokens (input + output).
2. If yes, how many additional tokens (input + output) are still needed to finish the task, starting from the next turn. Return an estimation interval: at least est_low tokens and at most est_high tokens.
3. If no, answer "impossible".
4. Prioritize the can-finish judgment over interval tightness. If you think the task can finish within budget, make the interval as tight as possible while still covering the true remaining token budget.

Example: For a three-turn interaction, suppose only Turn 1 has been completed. The full interaction is: Turn 1: input X1 tokens, output Y1 tokens; Turn 2: input X2 tokens, output Y2 tokens; Turn 3: input X3 tokens, output Y3 tokens. You will receive: turn_token_usage_text: Turn 1: input X1 tokens, output Y1 tokens You should estimate: X2 + Y2 + X3 + Y3

Output exactly one of the following:

`<think>[YOUR THINKING]</think><answer>[est_low, est_high]</answer>`

or

`<think>[YOUR THINKING]</think><answer>impossible</answer>`

Sokoban token-budget prompt.

Based on the provided rollout context, you are provided below information:

1. You have completed `{completed_turns}` turns.
2. Each turn, your token consumption is `{turn_token_usage_text}`.
3. You need to finish the task within `{max_context_window_tokens}` tokens.

Now, estimate:

1. Whether you can finish the task successfully within `{max_context_window_tokens}` total tokens (input + output).
2. If yes, how many additional tokens (input + output) are still needed to finish the task, starting from the next turn. Return an estimation interval: at least est_low tokens and at most est_high tokens.
3. If no, answer "impossible".
4. You should try your best to estimate whether the task can finish within budget (most important). If you think the task can finish within budget, your interval should be as tight as possible while still covering the true remaining token budget.

Example: For a three-turn interaction, suppose only Turn 1 has been completed. The full interaction is: Turn 1: input X1 tokens, output Y1 tokens; Turn 2: input X2 tokens, output Y2 tokens; Turn 3: input X3 tokens, output Y3 tokens. You will receive: turn_token_usage_text: Turn 1: input X1 tokens, output Y1 tokens You should estimate: X2 + Y2 + X3 + Y3

Output exactly one of the following:

`<think>[YOUR THINKING]</think><answer>[est_low, est_high]</answer>`

or

`<think>[YOUR THINKING]</think><answer>impossible</answer>`

SWE-bench token-budget prompt.

Based on the provided SWE-bench rollout context, you are given the following:

1. The coding agent has completed `{completed_turns}` turns.
2. Per-turn token usage so far, excluding reused history from earlier turns, is: `{turn_token_usage_text}`.
3. The full task must finish within `{max_context_window_tokens}` total tokens.

Estimate:

1. Whether the agent can still finish the software issue successfully within `{max_context_window_tokens}` total tokens.
2. If yes, how many additional tokens (input + output) are still needed from the next turn onward. Return an interval: at least est_low tokens and at most est_high tokens.
3. If no, answer "impossible".
4. Prioritize the can-finish judgment over interval tightness. If the task still looks finishable, keep the interval as tight as possible while still covering the true remaining token budget.

Think about typical SWE-bench costs such as repository inspection, targeted code edits, running validation commands, reading failures, and one or two repair iterations.

Output exactly one of the following:

`<think> [YOUR THINKING] </think><answer>[est_low, est_high]</answer>`

or

`<think> [YOUR THINKING] </think><answer>impossible</answer>`

External Warehouse prompt.

System prompt. You are an evaluation agent for historical warehouse-management rollouts. Determine whether the rollout can still finish successfully within the remaining resource budgets while reaching the target cash threshold. If it can, estimate the remaining time, warehouse cumulative occupancy, and cumulative cost still needed from the next turn onward. Follow the required output format exactly.

Based on the provided warehouse rollout context, you are given the following information:

1. You have completed `{completed_weeks}` weeks in `{completed_turns}` turns.
2. Current cumulative usage so far:
   - time_weeks: `{current_time_weeks}`
   - warehouse_item_weeks: `{current_warehouse_item_weeks}`
   - cumulative_cost_usd: `{current_cost_usd}`
3. Current cash is `{current_cash_usd}` USD. To count as finished, final cash must reach at least `{target_cash_usd}` USD.
4. Historical resource consumption by completed step is: `{resource_consumption_text}`
5. The rollout must finish within all three budgets:
   - time_weeks <= `{budget_time_weeks}`
   - warehouse_item_weeks <= `{budget_warehouse_item_weeks}`
   - cumulative_cost_usd <= `{budget_cost_usd}`

Now, estimate:

1. Whether the rollout can still finish successfully within all three budgets while also reaching the target cash.
2. If yes, how much additional usage is still needed from the next turn onward. Return one interval for each metric.
3. If no, answer "impossible".
4. Prioritize the can-finish judgment over interval tightness. If you think the rollout can finish within budget, make each interval as tight as possible while still covering the true remaining value.

Output exactly one of the following:

`<think>[YOUR THINKING]</think><answer>time_weeks:[est_low, est_high], warehouse_item_weeks:[est_low, est_high], cumulative_cost_usd:[est_low, est_high]</answer>`

or

`<think>[YOUR THINKING]</think><answer>impossible</answer>`

Sokoban SFT/RL budget-probe prompt.

System prompt. You're a helpful assistant. You are solving the Sokoban puzzle. Push all boxes to targets. You are given the grid and zero-indexed coordinates of the player, boxes, and targets. You can push but not pull boxes, and cannot push a box through a wall. Your available actions are: Up, Down, Left, Right. You may output at most 3 action(s) in a single turn, separated by the action separator " || ".

User estimation prompt.
Based on the provided rollout context, you are provided below information:

1. You have completed `{completed_turns}` turns.
2. Each turn, your token consumption is `{turn_token_usage_text}`.
3. You need to finish the task within `{max_context_window_tokens}` tokens.

Now, estimate:

1. Whether you can finish the task successfully within `{max_context_window_tokens}` total tokens (input + output).
2. If yes, how many additional tokens (input + output) are still needed to finish the task, starting from the next turn. Return an estimation interval: at least est_low tokens and at most est_high tokens.
3. If no, answer "impossible".
4. You should try your best to estimate whether the task can finish within budget (most important). If you think the task can finish within budget, your interval should be as tight as possible while still covering the true remaining token budget.

Example: For a three-turn interaction, suppose only Turn 1 has been completed. The full interaction is: Turn 1: input X1 tokens, output Y1 tokens; Turn 2: input X2 tokens, output Y2 tokens; Turn 3: input X3 tokens, output Y3 tokens. You will receive: turn_token_usage_text: Turn 1: input X1 tokens, output Y1 tokens You should estimate: X2 + Y2 + X3 + Y3

Output exactly one of the following:

`<answer>[est_low, est_high]</answer>`
or
`<answer>impossible</answer>`

K Data Anonymization

The Warehouse environment is constructed from real enterprise supply-chain data. Before any use in experiments or release, all records were anonymized: company names, supplier identifiers, product codes, and any other fields that could uniquely identify an organization or individual were replaced with pseudonyms (e.g., *Acme* for the distributor and archetype names such as *MegaMart* for retail accounts; App. C.1). No raw identifiers appear in any prompt, trajectory, or publicly released artifact. A manual review of a random sample of 200 records confirmed that no personally identifying information or offensive content remained after anonymization.