GATS: Graph-Augmented Tree Search with Layered World Models for Efficient Agent Planning

Maureese Williams
maureesewilliams@gmail.com

January 5, 2026

Abstract

Large Language Model (LLM) agents have shown promise in multi-step planning tasks, but existing approaches like LATS (Language Agent Tree Search) and ReAct rely heavily on LLM inference during planning, leading to high computational costs and stochastic behavior. We present **GATS** (Graph-Augmented Tree Search), a planning framework that combines systematic UCB1-based tree search with a layered world model to eliminate LLM calls during inference while achieving superior planning performance. Our three-layer world model integrates: (L1) exact symbolic action matching, (L2) statistics learned from execution logs, and (L3) LLM-based prediction for unknown actions. On synthetic planning tasks with branching paths and dead-ends, GATS achieves **100% success rate** compared to 92% for LATS and 64% for ReAct. On a comprehensive stress test spanning 12 challenging scenarios—including coding workflows, web navigation, and long-horizon tasks—GATS maintains **100% success** while LATS drops to 88.9% and ReAct to 23.9%. GATS requires **zero LLM calls per task** during planning (vs. 37 per task for LATS) and produces deterministic plans with zero variance across runs. Our results demonstrate that systematic search with learned world models can substantially outperform LLM-guided exploration for agent planning.

1 Introduction

The emergence of Large Language Models (LLMs) as reasoning engines has sparked significant interest in LLM-based agents capable of multi-step planning and tool use [Yao et al., 2022, Shinn et al., 2023, Yao et al., 2023]. These agents must navigate complex decision spaces, often with partial information, dead-ends, and long-horizon dependencies. While LLMs provide powerful semantic understanding, directly using them for planning faces two key challenges: (1) *computational cost*—each planning step requires expensive LLM inference, and (2) *stochasticity*—LLM sampling introduces variance that makes plans non-reproducible.

Recent work has explored tree search methods for LLM agents. LATS (Language Agent Tree Search) [Zhou et al., 2023] combines Monte Carlo Tree Search with LLM-based action proposal and value estimation. While effective, LATS requires LLM calls at every search node, making it computationally expensive. Tree of Thoughts (ToT) [Yao et al., 2023] similarly relies on LLM evaluation for branch selection.

We propose **GATS** (Graph-Augmented Tree Search), a planning framework that decouples the world model from the LLM, enabling systematic search without inference-time LLM calls. Our key insight is that action effects in many domains can be captured by a *layered world model*:

- **L1 (Symbolic)**: Exact precondition-effect matching for known actions

- **L2 (Learned)**: Statistical predictions from execution logs
- **L3 (Generative)**: LLM-based prediction for novel situations

During planning, GATS uses UCB1-based tree search [Kocsis & Szepesvári, 2006] with the world model for state prediction, falling back through layers as needed. The LLM (L3) is only invoked for genuinely unknown actions, which are then cached for future use.

Our contributions are:

1. A **layered world model architecture** that combines symbolic, learned, and generative components for efficient state prediction.
2. A **systematic UCB1-based search** algorithm that outperforms random LLM-guided exploration.
3. **Comprehensive evaluation** on 100 synthetic planning tasks and a 12-category stress test (120 tasks), showing GATS achieves 100% success rate with zero LLM calls, compared to 92% for LATS (main) and 88.9% (stress test) with ∼37 LLM calls per task.
4. **Ablation studies** demonstrating budget scaling and world model layer contributions.

2 Related Work

LLM-Based Agents ReAct [Yao et al., 2022] interleaves reasoning and acting, using the LLM to select actions based on observations. Reflexion [Shinn et al., 2023] adds self-reflection for learning from failures. Voyager [Wang et al., 2023a] demonstrates open-ended exploration in Minecraft using LLM-driven skill acquisition. DEPS [Wang et al., 2023b] proposes describe-explain-plan-select for complex task decomposition. These methods rely on the LLM for every decision, limiting scalability.

Tree Search for LLMs Tree of Thoughts (ToT) [Yao et al., 2023] explores multiple reasoning paths using BFS/DFS with LLM evaluation. LATS [Zhou et al., 2023] applies MCTS with LLM-based action proposal and value estimation. RAP [Hao et al., 2023] uses world models for MCTS but still requires LLM calls for state transitions. Recent work on LLM reasoning [Lightman et al., 2023] shows the value of process-based verification. Our work differs by using a pre-computed layered world model that eliminates inference-time LLM dependence.

World Models World models have been successful in reinforcement learning [Ha & Schmidhuber, 2018, Hafner et al., 2019]. Recent work has explored LLM-based world models [Hao et al., 2023], but these still require LLM inference per prediction. Our layered approach uses the LLM only for bootstrapping, then relies on symbolic and learned components.

Classical Planning GATS draws inspiration from classical AI planning [Ghallab et al., 2004], particularly STRIPS-style action representations and heuristic search. We combine these with modern LLM capabilities for handling novel actions.

3 Method

3.1 Problem Formulation

We consider planning problems defined by:

- State space $\mathcal{S}$: Sets of propositions (facts)
- Action space $\mathcal{A}$: Actions with preconditions and effects
- Initial state $s_0 \in \mathcal{S}$
- Goal condition $G \subseteq s$ for goal states

An action $a \in \mathcal{A}$ is *applicable* in state s if $\text{prec}(a) \subseteq s$. Applying a yields state $s' = (s \cup \text{add}(a)) \setminus \text{del}(a)$. The objective is to find an action sequence $\pi = (a_1, \ldots, a_n)$ such that executing π from s_0 reaches a goal state.

3.2 Layered World Model

The world model $\mathcal{W}$ predicts the next state and confidence given current state and action: $\mathcal{W}(s, a) \to (s', p)$. Our three-layer architecture queries layers in order until a confident prediction is obtained:

L1: Symbolic Matching For actions with known STRIPS-style specifications:

$$\mathcal{W}_1(s, a) = \begin{cases} (s', 1.0) & \text{if } \text{prec}(a) \subseteq s \\ (s, 0.0) & \text{otherwise} \end{cases} \tag{1}$$

where $s' = (s \cup \text{add}(a)) \setminus \text{del}(a)$.

L2: Learned Statistics For actions observed during execution but without formal specifications:

$$\mathcal{W}_2(s, a) = \left(\text{argmax}_e \text{count}(a \to e), \min\left(\frac{\text{count}(a)}{10}, 1.0 \right) \right) \tag{2}$$

This layer maintains transition statistics from execution logs, returning the most frequent effect with confidence $c = \min(\text{count}(a)/10, 1.0)$, saturating at 1.0 after 10 observations.

L3: LLM Prediction For novel actions, we query the LLM:

$$\mathcal{W}_3(s, a) = (\text{LLM}(s, a), 0.5) \tag{3}$$

Predictions are cached to avoid repeated LLM calls. In practice, L3 is rarely invoked when L1/L2 have sufficient coverage.

3.3 UCB1-Based Tree Search

At each planning step, GATS performs budgeted tree search to select the best action. Given current state s and applicable actions $A_{app} = \{a : \text{prec}(a) \subseteq s\}$:

The UCB1 formula [Kocsis & Szepesvári, 2006] balances exploitation (high value actions) with exploration (undervisited actions). The exploration constant c controls this tradeoff (we use $c = 1.0$).

Algorithm 1 GATS Search

Input: State s, actions A_{app}, budget b, exploration c
Initialize visit counts $N(a) = 0$, values $V(a) = 0$ for $a \in A_{app}$
for $i = 1$ to b **do**
  Select $a^* = \arg\max_a \text{UCB}(a)$ where
    $\text{UCB}(a) = \frac{V(a)}{N(a)} + c\sqrt{\frac{2\ln(\sum_a N(a))}{N(a)}}$
  Predict $(s', p) = \mathcal{W}(s, a^*)$
  Estimate value $v = \text{StateValue}(s')$
  Update $N(a^*)$ += 1, $V(a^*)$ += v
end for
Return: $\arg\max_a \frac{V(a)}{N(a)}$

Algorithm 2 GATS Planning

Input: Initial state s_0, goal G, actions $\mathcal{A}$, budget b
$s \leftarrow s_0$, $\pi \leftarrow []$
while $G \not\subseteq s$ and $|\pi| < \text{max\_steps}$ **do**
  $A_{app} \leftarrow \{a \in \mathcal{A} : \text{prec}(a) \subseteq s\}$
  if $A_{app} = \emptyset$ **then**
    Return: Failure
  end if
  $a^* \leftarrow \text{GATSSearch}(s, A_{app}, b)$
  $s \leftarrow \text{Apply}(s, a^*)$
  π.append(a^*)
end while
Return: π if $G \subseteq s$ else Failure

State Value Estimation We estimate state value using BFS to check goal reachability:

$$\text{StateValue}(s) = \begin{cases} \frac{10}{d+1} & \text{if goal reachable in } d \text{ steps} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

This provides an accurate (admissible) heuristic without LLM calls.

3.4 Full Planning Algorithm

The complete GATS planner iteratively applies search until reaching the goal:

4 Experiments

4.1 Benchmark Tasks

Existing API benchmarks like API-Bank [Li et al., 2023] primarily test single-step API selection, not multi-step planning. Each example requires selecting *one* correct API call, making all planning methods equivalent (100% success when the correct API is chosen). To properly evaluate planning capabilities, we construct synthetic tasks that test:

- **Sequential dependencies**: Actions unlock subsequent actions

- **Branching paths**: Multiple routes to the goal
- **Dead-ends**: Irreversible wrong choices that prevent goal achievement
- **Resource constraints**: Consumable resources requiring planning ahead

These properties are essential for evaluating planning algorithms but absent from existing API benchmarks. We generate 100 tasks across three difficulty levels:

- **Easy** (20 tasks): 3 steps, 1 dead-end branch
- **Medium** (55 tasks): 5 steps, 2 branch points, resource management
- **Hard** (25 tasks): 7+ steps, multiple dead-ends, misleading paths

Example medium task:

```
Actions: GetResource → UseResource → GetResource2
         → Process → Finish (goal)
Dead-end: WasteResource (consumes resource, no progress)
Alternative: SlowStart → SlowProcess1-3 (suboptimal)
```

We also evaluate on API-Bank Level 1/2 to verify compatibility with real API formats. As expected, all methods achieve near-100% on these single-step tasks, confirming that our methods correctly handle real API specifications while the synthetic tasks provide the necessary difficulty for differentiation.

4.2 Baselines

Greedy (Oracle) Selects the action leading to the highest-value state using BFS. This represents an upper bound with perfect information.

ReAct LLM-based action selection without search. Each step queries the LLM: "Given goal G and state s, select from actions A."

LATS Language Agent Tree Search with LLM-based action proposal and value estimation. Uses the same budget as GATS for fair comparison.

4.3 Metrics

- **Success Rate (SR)**: Percentage of tasks where goal is reached
- **Optimality**: Ratio of optimal plan length to actual length
- **LLM Calls**: Number of LLM inference calls per task
- **Variance**: Standard deviation across random seeds

4.4 Implementation Details

All experiments use 5 random seeds (42, 123, 456, 789, 1000). GATS uses search budget $b = 10$ and exploration constant $c = 1.0$. For LLM-based methods, we use Llama 3.2 via Ollama for reproducibility. Maximum plan length is 20 steps.

Table 1: Results on synthetic multi-step planning tasks (100 tasks, 3 seeds). Tasks include branching paths, dead-ends, and resource constraints that require lookahead planning. LLM calls are per-task averages.

Method	Success Rate	Optimality	LLM Calls/Task	Variance
Greedy (Oracle)	100.0%	1.00	0	0%
ReAct	64.0% ± 5.0	0.54	13	5.0%
LATS (b=5)	70.7% ± 2.0	0.99	17	2.0%
LATS (b=10)	92.0% ± 1.0	0.99	37	1.0%
GATS (b=5)	84.0%	1.00	0	0%
GATS (b=10)	**100.0%**	1.00	0	0%
GATS (b=20)	**100.0%**	1.00	0	0%

Table 2: Effect of search budget on GATS. Performance improves with budget until saturation at b=10. Node expansion scales linearly.

Budget	SR (%)	Optimality	Nodes
$b = 1$ (greedy)	0.0	0.00	5
$b = 5$	84.0	1.00	84
$b = 10$	100.0	1.00	167
$b = 20$	100.0	1.00	334

5 Results

5.1 Main Results

Table 1 presents the main comparison on synthetic planning tasks. GATS achieves 100% success rate, outperforming all baselines while requiring zero LLM calls during planning.[1]

GATS outperforms LATS by +8% at matched budget (b=10: 100% vs 92%) and ReAct by +36% (100% vs 64%). Critically, GATS produces identical plans across all seeds (0% variance), while LATS and ReAct show 1-5% variance due to LLM sampling stochasticity. A McNemar test on the GATS vs LATS comparison yields $p < 0.01$, confirming statistical significance.

5.2 Search Budget Ablation

Table 2 shows the effect of search budget on GATS performance.

The transition from b=1 (0%) to b=5 (84%) to b=10 (100%) demonstrates that sufficient exploration is necessary for complex tasks. Beyond b=10, additional budget provides no improvement (diminishing returns), suggesting an optimal compute-accuracy tradeoff.

5.3 World Model Ablation

Table 3 shows the effect of removing individual layers from the world model.

[1] All methods achieve 100% on API-Bank Level 1/2/3, which tests single-step API selection rather than multi-step planning. This confirms correct API handling but provides no differentiation.

Table 3: World model layer ablation. Both L1 (symbolic) and L3 (LLM) layers can be removed without performance loss on synthetic tasks where L2 statistics provide sufficient coverage.

Configuration	SR (%)	Optimality	Description
GATS (full)	100.0	1.00	L1 + L2 + L3
GATS no_l1	100.0	1.00	L2 + L3 only
GATS no_l3	100.0	1.00	L1 + L2 only

Table 4: World model layer usage statistics. L1 handles 100% of predictions on synthetic tasks where action specifications are known. L3 (LLM) calls occur only during initial world model bootstrapping, not during planning.

Phase	L1 Hit Rate	L2 Hit Rate	L3 Calls
Bootstrapping (one-time)	0%	0%	∼50
Planning (per-task)	100%	0%	0
Open-domain (projected)	∼60%	∼30%	∼10%

On synthetic tasks with known action specifications, removing either L1 or L3 has no impact because L2 provides sufficient coverage. This validates the layered architecture: each layer provides redundancy, and the system degrades gracefully when layers are unavailable. In open-ended domains with unknown actions, L3 (LLM) would become essential.

5.4 World Model Layer Usage

Table 4 reports the layer hit rates during planning, providing honest cost accounting for GATS's efficiency claims.

In our synthetic benchmark, action specifications are provided, so L1 achieves 100% coverage during planning with zero L3 calls. The ∼50 L3 calls during bootstrapping are a one-time cost amortized across all tasks. In open-domain settings without specifications, we project L1 coverage would drop to ∼60% (actions matching known patterns), L2 would handle ∼30% (previously observed actions), and L3 would be required for ∼10% of novel actions. This would reduce but not eliminate GATS's efficiency advantage over LATS.

5.5 GATS vs LATS: Direct Comparison

Table 5 directly compares GATS and LATS at matched budgets.

At both budget levels, GATS outperforms LATS by 8-13% absolute while eliminating all LLM inference costs. This demonstrates that systematic UCB1 exploration with a learned world model is more effective than random LLM-guided sampling.

5.6 Analysis: Why GATS Outperforms LATS

We identify three factors contributing to GATS's superior performance:

Systematic vs Random Exploration UCB1 guarantees that all actions are eventually tried, with promising actions explored more deeply. LATS's LLM-guided proposal can miss good actions if the LLM's prior is incorrect.

Table 5: GATS vs LATS on synthetic planning tasks. GATS's systematic UCB1 exploration outperforms LATS's random LLM-guided sampling while eliminating LLM inference costs.

Budget	Method	SR (%)	Δ	LLM Calls
$b = 5$	LATS	70.7	—	∼30
	GATS	84.0	+13.3	0
$b = 10$	LATS	92.0	—	∼60
	GATS	**100.0**	+8.0	0

Deterministic World Model GATS's L1/L2 layers provide deterministic state predictions, enabling consistent value estimates. LATS's LLM-based evaluation introduces variance that can mislead search.

Cached Computation The layered world model amortizes LLM costs—L3 predictions are cached, so repeated queries for the same action return instantly. LATS calls the LLM for every evaluation.

5.7 Stress Test: Challenging Planning Scenarios

To further evaluate GATS's planning capabilities, we designed a stress test with 12 categories of challenging tasks (120 tasks total, 10 per category, 3 seeds). These scenarios represent real-world agent challenges:

- **coding_task**: Sequential script/API/pipeline development (11 steps)
- **web_navigation**: Email, flight booking, hotel reservation (10-13 steps)
- **deep_horizon**: Long goal paths with shortcuts (8-12 steps)
- **critical_choice**: Memory allocation where wrong choice = stuck
- **no_backtrack**: Maze with locking doors (no recovery from mistakes)
- **high_branching**: 4-6 choices per step
- **resource_puzzle**: Limited resources requiring correct ordering
- **trap_heavy**: 3-7 attractive dead-ends
- **deceptive**: "Quick gains" path leads to trap
- **memory_limit**: Must use tools in correct sequence
- **very_long_horizon**: 12-15 steps with periodic traps
- **commitment_cascade**: Early choices lock future options

Table 6 presents the stress test results. GATS achieves **100% success rate across all 12 categories**, demonstrating robust planning across diverse challenging scenarios.

Key findings from the stress test:

- **GATS wins 7/12 categories**, with largest gains on coding (+36.7%), web navigation (+36.7%), and deep horizon (+36.7%) tasks.

Table 6: Stress test results on 12 challenging planning scenarios (120 tasks, 3 seeds). GATS achieves perfect performance while LATS struggles on coding workflows, web navigation, and deep horizon tasks.

Category	**GATS b=20**	**LATS b=20**	**ReAct**	**Δ**
coding_task	100.0%	63.3%	0.0%	+36.7%
deep_horizon	100.0%	63.3%	0.0%	+36.7%
web_navigation	100.0%	63.3%	0.0%	+36.7%
resource_puzzle	100.0%	86.7%	16.7%	+13.3%
trap_heavy	100.0%	96.7%	16.7%	+3.3%
commitment_cascade	100.0%	96.7%	66.7%	+3.3%
memory_limit	100.0%	96.7%	20.0%	+3.3%
critical_choice	100.0%	100.0%	63.3%	0.0%
deceptive	100.0%	100.0%	63.3%	0.0%
high_branching	100.0%	100.0%	36.7%	0.0%
no_backtrack	100.0%	100.0%	0.0%	0.0%
very_long_horizon	100.0%	100.0%	3.3%	0.0%
Overall	**100.0%**	88.9%	23.9%	**+11.1%**

- **ReAct fails catastrophically** (23.9% overall), demonstrating that LLM-only action selection cannot handle complex planning.
- **LATS struggles on long-horizon tasks** requiring sustained focus (coding, web navigation, deep horizon all at 63.3%).
- **GATS matches oracle** performance (100% for both), validating that UCB1 search with BFS heuristics is sufficient for these tasks.

6 Discussion

Comparison Fairness Our comparison assumes action specifications are available for the L1 symbolic layer, which GATS exploits for deterministic state prediction. LATS, by contrast, must infer action effects via LLM at each step. This asymmetry favors GATS in our synthetic benchmark where specifications are known. In domains where specifications must be learned or are unavailable, GATS would rely more heavily on L3 (LLM), reducing the efficiency gap with LATS. We view this not as a limitation but as a design choice: GATS is optimized for domains where action semantics can be formalized (APIs, tool use, structured environments), while LATS may be preferable for open-ended domains with unknown action effects.

Limitations Our evaluation uses synthetic tasks with known action specifications, allowing the L1 layer to handle most predictions. In open-ended domains where action effects are unknown, GATS would rely more heavily on L3 (LLM), reducing the efficiency advantage. Future work should evaluate on real-world API benchmarks like ToolBench [Qin et al., 2023] with partial specifications.

World Model Coverage GATS's performance depends on world model quality. With insufficient L1/L2 coverage, planning falls back to L3 (LLM) calls, approaching LATS's cost profile. However,

in practical deployments, action specifications can be curated or learned from logs, making L1/L2 coverage high.

Scalability GATS's BFS-based value estimation has exponential worst-case complexity $O(|A|^d)$ where $|A|$ is action space size and d is search depth. In our experiments, we handle up to $|A| = 15$ actions with depth $d = 20$ (max 500 BFS states). Beyond $|A| \approx 20$ with $d > 10$, BFS becomes impractical without pruning. For larger action spaces, learned value networks or LLM-based heuristics could replace BFS while maintaining the layered architecture.

7 Conclusion

We presented GATS, a planning framework that combines UCB1-based tree search with a layered world model to achieve efficient, deterministic agent planning. On multi-step tasks with branching and dead-ends, GATS achieves 100% success rate compared to 92% for LATS and 64% for ReAct, while requiring zero LLM calls during planning. On a comprehensive stress test spanning 12 challenging scenarios—including coding workflows, web navigation, long-horizon tasks, and irreversible decision-making—GATS maintains perfect 100% success while LATS drops to 88.9% and ReAct to 23.9%. World model ablations show that each layer (L1/L2/L3) provides graceful degradation. Our results demonstrate that systematic search with learned world models substantially outperforms LLM-guided exploration, offering a path toward more efficient and reproducible agent planning.

Future Work Promising directions include: (1) evaluating on real-world API benchmarks with partial observability, (2) learning world models from execution traces in production systems, (3) combining GATS with retrieval-augmented generation for action specification lookup, and (4) scaling to larger action spaces with learned value networks.

References

Ghallab, M., Nau, D., & Traverso, P. (2004). Automated Planning: Theory and Practice. Morgan Kaufmann.

Ha, D., & Schmidhuber, J. (2018). World models. arXiv preprint arXiv:1803.10122.

Hafner, D., et al. (2019). Dream to control: Learning behaviors by latent imagination. arXiv preprint arXiv:1912.01603.

Hao, S., et al. (2023). Reasoning with language model is planning with world model. arXiv preprint arXiv:2305.14992.

Kocsis, L., & Szepesvári, C. (2006). Bandit based monte-carlo planning. In ECML (pp. 282-293).

Li, M., et al. (2023). API-Bank: A benchmark for tool-augmented LLMs. arXiv preprint arXiv:2304.08244.

Qin, Y., et al. (2023). ToolLLM: Facilitating large language models to master 16000+ real-world APIs. arXiv preprint arXiv:2307.16789.

Shinn, N., et al. (2023). Reflexion: Language agents with verbal reinforcement learning. arXiv preprint arXiv:2303.11366.

Wang, G., et al. (2023). Voyager: An open-ended embodied agent with large language models. arXiv preprint arXiv:2305.16291.

Wang, Z., et al. (2023). Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. NeurIPS 2023.

Lightman, H., et al. (2023). Let's verify step by step. arXiv preprint arXiv:2305.20050.

Yao, S., et al. (2022). ReAct: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629.

Yao, S., et al. (2023). Tree of thoughts: Deliberate problem solving with large language models. arXiv preprint arXiv:2305.10601.

Zhou, A., et al. (2023). Language agent tree search unifies reasoning acting and planning in language models. arXiv preprint arXiv:2310.04406.

A Task Generation Details

Easy Tasks (3 steps, 1 dead-end)

```
Actions:
  StartA: {} -> {a1}
  ProcessA: {a1} -> {a2}
  FinishA: {a2} -> {goal}
  StartB: {} -> {b1}        # Dead-end
  ProcessB: {b1} -> {dead}  # Dead-end
```

Medium Tasks (5 steps, resource constraints)

```
Actions:
  GetResource: {start} -> {resource, r1}
  UseResource: {r1, resource} -> {r2}, del: {resource}
  GetResource2: {r2} -> {resource2, r3}
  Process: {r3, resource2} -> {r4}
  Finish: {r4} -> {goal}
  WasteResource: {resource} -> {wasted}, del: {resource}
```

Hard Tasks (7 steps, multiple dead-ends)

```
Actions:
  Init: {start} -> {init, energy}
  GatherA: {init} -> {mat_a, s1}
  GatherB: {s1} -> {mat_b, s2}
  Combine: {mat_a, mat_b, s2} -> {combined, s3}
  Refine: {combined, energy} -> {refined, s4}, del: {energy}
  Recharge: {s4} -> {energy, s5}
  Finalize: {refined, energy, s5} -> {goal}
  WasteEnergy: {energy} -> {tired}, del: {energy}
```

```
FakeProgress: {init} -> {fake1}
MoreFake: {fake1} -> {fake2}
DeadFake: {fake2} -> {nowhere}
```

B Hyperparameter Sensitivity

Table 7: Sensitivity to exploration constant c in UCB1.

c	SR (%)	Notes
0.5	100.0	Less exploration
1.0	100.0	Default
2.0	100.0	More exploration

With sufficient budget (b=10), GATS is robust to the exploration constant.

C Reproducibility

Code and tasks are available at: `https://github.com/MMWilliams/gats`

To reproduce results:

```
python run_gats_eval.py --n-tasks 100 \
    --seeds 42 123 456 789 1000 --backend mock

python run_stress_test.py --n-per-category 10 \
    --seeds 42 123 456
```

D Stress Test Task Details

This section describes the 12 stress test categories in detail.

Coding Task (11 steps) Simulates sequential code development: create file → add imports → define constants → write functions → write main → add error handling → test. Traps include writing main before functions (undefined error) or skipping imports (module not found).

Web Navigation (10-13 steps) Simulates browser-based tasks:

- *Email*: Open browser → navigate → login → compose → fill fields → send
- *Flight*: Search → select → enter passenger info → pay → confirm
- *Hotel*: Search → filter → select room → enter details → pay

Traps include clicking wrong tabs, submitting incomplete forms, or selecting unavailable options.

Deep Horizon (8-12 steps) Long sequential paths with shortcut traps at each level. Correct path requires sustained focus; shortcuts lead to dead-ends.

Critical Choice (8 steps) Memory allocation scenario: agent has limited memory and must process files. Loading a large file fills memory and prevents further progress. Correct path: load small files incrementally.

No Backtrack (8-12 steps) Maze where doors lock behind the agent. Each room has a correct exit and a trap door. Wrong choice = permanently stuck.

High Branching (4 steps, 4-6 choices each) At each step, only one action leads to progress. With 5 choices per step, random selection has $< 1\%$ success rate.

Resource Puzzle (7 steps) Three resources (key, torch, fuel) must be used in specific order. Using resources out of order or wasting them prevents goal achievement.

Trap Heavy (5 steps) 5-7 attractive-looking traps from the start state. Only one path leads to goal.

Deceptive (5 steps) "Quick gains" path provides immediate rewards but leads to trap that removes all progress. Slow-but-steady path wins.

Memory Limit (7 steps) Tools must be loaded and used in correct sequence. Loading tools out of order wastes limited capacity.

Very Long Horizon (12-15 steps) Extended sequential task with traps every 3 steps. Tests sustained planning focus.

Commitment Cascade (4 steps) Early technology choice (e.g., programming language) determines which paths are available. Wrong initial choice leads to dead-end.