Proceedings of Machine Learning Research **318** (2026)
The 39th Canadian Conference on Artificial Intelligence

Canadian AI 2026
25-29 May 2026,Vancouver,BC

VOLTS: Validated Output through Logit Tree Search for Reliable PDDL Planning with Small Language Models

Nicholas Massad, Amine Trabelsi, François Ferland, Froduald Kabanza,
Université de Sherbrooke, Sherbrooke, QC, Canada

Abstract

Autonomous agents running on edge hardware need dependable task-planning but cannot afford the compute footprint of frontier Large Language Models (LLMs). We show that a single pass with a 4-bit quantized Llama 3.1 8B Small Language Model (SLM) can generate valid plans in the Planning Domain Definition Language (PDDL) while respecting tight memory and latency budgets. VOLTS combines three ideas: **(1) Action-token fine-tuning** on a custom vocabulary where every token encodes a complete grounded action; **(2) A real-time symbolic validator** that checks each candidate token against the current state during decoding; and **(3) Parallel branching search** that explores promising alternatives within the same forward pass. On 2,000 IPC problems (Blocksworld, Logistics, DriverLog, Rover), VOLTS achieves 76.4% plan validity with plans averaging 1.08× the length of Fast Downward solutions, demonstrating that a fine-tuned 8B SLM augmented with in-loop validation drastically narrows the capability gap to frontier LLMs that are orders of magnitude larger (GPT-4o: 7.2%), at a fraction of the compute cost. The ablation baseline–an identically fine-tuned SLM without in-loop validation, achieves only 0.13%, confirming the integrated validation-and-search mechanism is essential.

Keywords: Automated Planning, PDDL, Small Language Models, Neuro-Symbolic AI, Tree Search, Edge Deployment

1. Introduction

Large Language Models (LLMs) have spurred intense interest in automated planning, often formalized using the Planning Domain Definition Language (PDDL) [1]. Yet a growing consensus [2–4] indicates that even frontier models struggle with strict constraint adherence, verifiable correctness, and reliable multi-step reasoning. They hallucinate actions [5], lack self-verification mechanisms [2], and their performance on planning benchmarks remains limited [6]. Iterative hybrid approaches such as ReAct [7], Tree of Thoughts [8], Tree-Planner [9], and LLM-Modulo frameworks [2] improve reliability but require costly multi-pass interactions and still cannot *guarantee* plan validity during generation. In parallel, frontier LLMs are inaccessible to latency- and memory-constrained edge agents (robots, drones, embedded IoT), motivating reliable *on-device* planning with Small Language Models (SLMs).

This paper introduces VOLTS (**V**alidated **O**utput through **L**ogit **T**ree **S**earch), a framework that enables a fine-tuned, 4-bit quantized Llama 3.1 8B SLM to generate PDDL plans with proven validity in a single inference pass. As illustrated in Figure 1, VOLTS uniquely combines: (1) fine-tuning on a custom vocabulary where each token encodes a grounded PDDL action, enabling (2) real-time, per-token symbolic validation during decoding, integrated within (3) a logit-guided parallel tree search. This synergy guarantees that every retained partial plan is symbolically valid, eliminating hallucinations without sacrificing search breadth.

VOLTS does not aim to replace classical planners. Rather, it targets two complementary gaps: *(i)* classical search must ground and evaluate an action space that grows combinatorially with domain objects, whereas a fine-tuned SLM compresses this into a learned

* nicholas.massad@usherbrooke.ca amine.trabelsi@usherbrooke.ca francois.ferland@usherbrooke.ca froduald.kabanza@usherbrooke.ca

distribution proposing only a few probable next actions; and *(ii)* pre-trained language models encode cross-domain world knowledge that may enable generalization to novel tasks with limited retraining, a form of transfer absent from classical planning.

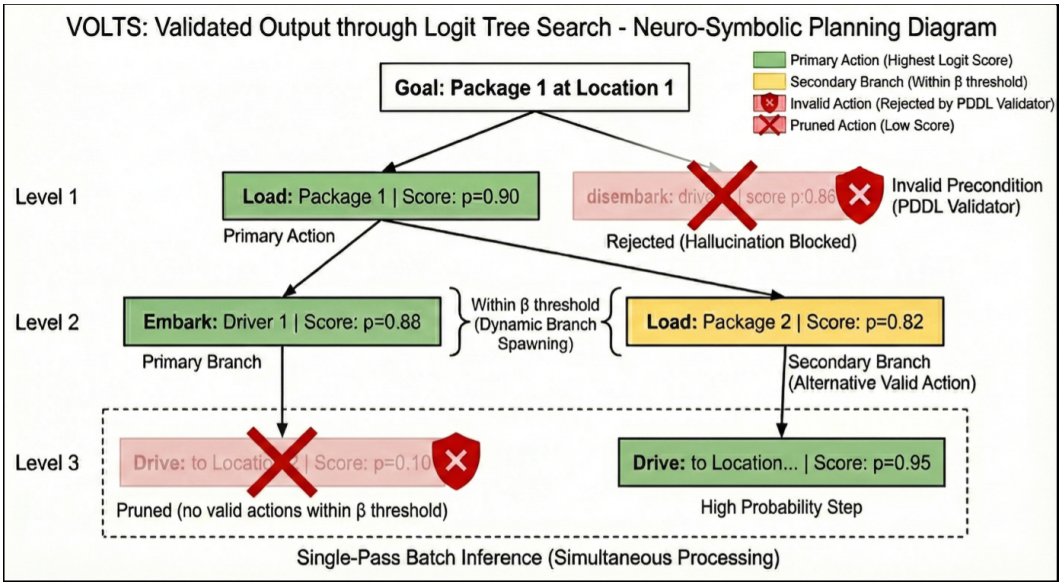

Figure 1. VOLTS explores multiple promising paths in parallel while guaranteeing that every retained partial plan remains symbolically valid. Green nodes = primary actions (highest logit), yellow nodes = secondary branches (within β threshold), red-crossed nodes = invalid actions rejected by the PDDL validator, pruned nodes = low-score actions eliminated.

Our central hypothesis is that fine-tuning equips SLMs with effective planning heuristics discernible in their output logits [10]. VOLTS manages a set of active plan branches; in each step the SLM proposes next actions for all branches simultaneously, which are immediately validated. New branches are spawned from promising alternatives whose logit scores fall within a β-ratio of the leading choice, subject to a limit of $k_{\max}$ concurrent branches. Our evaluation across 2,000 benchmark problems yields 76.4% average plan validity, strongly supporting this approach. The main contributions are:

(1) A novel framework enabling fine-tuned SLMs on a custom domain vocabulary to generate valid PDDL plans through a parallel, validated, logit-guided tree search.
(2) A custom PDDL grounding module for STRIPS [11] domains, designed for real-time action validation inside the decoding loop.
(3) Empirical demonstration that an 8B SLM with VOLTS achieves high planning success across standard PDDL domains, closing most of the capability gap to frontier LLMs (GPT-4o, GPT-o3-mini) at orders-of-magnitude lower compute cost, and vastly outperforming ablated baselines.

The full source code, training scripts, and evaluation pipeline are publicly available at `https://github.com/R3CK0/Volts`.

2. Methodology

VOLTS enables reliable PDDL planning with a 4-bit quantized Llama 3.1 8B by combining domain-specific fine-tuning with in-loop symbolic validation. The fine-tuning uses LoRA adapters initialized via PiSSA [12, 13], reducing computational requirements. Only

the adapter weights and vocabulary-related layers are updated; the quantized base model remains frozen. The training objective is standard autoregressive next-token prediction on valid PDDL plan sequences, using datasets derived from Blocksworld, Logistics, DriverLog, and Rover (∼20,000 examples each).

2.1. **Custom Vocabulary**

Standard sub-word tokenization fragments PDDL actions across multiple tokens (e.g., `analyze_rock_sample` becomes seven sub-tokens), making per-token validation impossible. We construct a custom vocabulary by extracting all unique grounded actions and parameter instances from the target PDDL domains, assigning each a unique token ID. The model's embedding and output layers are resized to accommodate this vocabulary and trained alongside the LoRA adapters. With each complete action encoded as a single token, the validator can check every candidate immediately at each decoding step.

2.2. **Action Grounding Module**

We developed a custom PDDL action validator in Python that dynamically assesses candidate action tokens at each generation step. Given the current symbolic state and a candidate action, the validator checks whether the corresponding PDDL preconditions are satisfied, returning a Boolean and, if valid, the resulting state. Its speed is critical for real-time integration within the decoding loop; existing PDDL validators are too slow due to subprocess overhead or library dependencies not designed for rapid per-action grounding.

2.3. **Logit-Based Tree Search Inference**

The VOLTS inference strategy manages a set $\mathcal{B}$ of active branches, each consisting of a partial plan P_i and its current symbolic state s_i. At each step, a batched input from all active branches is processed by the SLM in a single forward pass, yielding logit vectors for each branch. For each branch, candidate next-action tokens are validated against the current state by the grounding module. If no valid action exists, the branch is pruned. Otherwise, the highest-scoring valid action a_{primary} extends the branch and the state is updated; if the new state satisfies the goal, the plan is returned. To explore alternatives, any other valid action whose logit score falls within a β-ratio (≤ 1.15) of a_{primary} spawns a new branch, subject to $k_{\max}$ concurrent branches. The loop terminates when a solution is found, all branches are pruned, or the maximum plan length $T_{\max}$ is reached. This batched parallel processing with dynamic β-controlled branching efficiently explores diverse valid plan trajectories while guaranteeing PDDL-semantic correctness by construction. A detailed step-by-step algorithm and worked example are provided in Appendix A.

3. **Experimental Setup**

We evaluate on four PDDL benchmark domains from PlanBench [3] and the IPC STRIPS track: Blocksworld, Logistics, DriverLog, and Rover (500 problems each, 2,000 total). Fine-tuning datasets comprise ∼20,000 problem–plan pairs per domain generated by the classical planner Fast Downward [14].

Baselines include: **Fast Downward** [14] (classical planner; 100% validity reference); **GPT-4o** and **GPT-o3-mini** (zero-shot prompted with the PDDL domain and problem files; the exact prompt templates used are given in Appendix B); **Vanilla Llama 3.1 8B** (no fine-tuning); and a **Fine-tuned Llama 3.1 8B ablation** using identical PEFT techniques and custom vocabulary as VOLTS but with standard autoregressive decoding (no in-loop validation or branching).

The branching factor β was set to 1.15 (calibrated by reducing from 10 until perplexity reached at most 3 [15]; values above 1.45 caused significant quality degradation). Maximum concurrent branches $k_{\max} = 4$ (hardware-limited). All local inference used an NVIDIA RTX 4090 (24 GB VRAM). Performance is measured by **plan validity** (percentage of plans that are semantically valid and achieve the goal) and **plan length ratio** (generated plan length divided by Fast Downward's solution; 1.0 is optimal).

4. Results

Table 1 summarizes overall performance across all four domains.

Table 1. Overall performance summary across all four PDDL domains (2,000 problems). Plan length ratio is relative to Fast Downward; values below 1.0 indicate shorter plans, but only the validity column controls whether the plan is actually usable.

Method	Avg. Validity	Plan Length Ratio
Fast Downward (Symbolic)	100%	1.00
Vanilla Llama 3.1 8B	0.0%	—
Fine-tuned Llama 3.1 8B (Ablation)	0.13%	5.88
GPT-4o (zero-shot)	7.2%	0.52
GPT-o3-mini (reasoning, zero-shot)	48.9%	0.55
VOLTS (Ours, 8B SLM)	**76.4%**	**1.08**

SLM vs. frontier LLM. The primary motivation behind comparing VOLTS against GPT-4o and GPT-o3-mini is *deployability*: frontier LLMs with hundreds of billions of parameters cannot be deployed on edge hardware, where autonomous agents actually need to plan. VOLTS, running on a 4-bit quantized 8B model that fits on a single consumer GPU, achieves 76.4% validity, substantially higher than GPT-4o (7.2%) and noticeably higher than GPT-o3-mini (48.9%). The comparison is asymmetric: the GPT models are evaluated zero-shot, whereas VOLTS benefits from domain-specific fine-tuning on ∼20,000 examples per domain. Rather than a confounder, we view this as the point: the edge-deployment thesis of VOLTS is that when the target is a small, resource-constrained model, investing a modest offline fine-tuning budget plus in-loop validation yields *better* plans than calling a frontier model that cannot run on the device anyway. Prompt templates used for GPT-4o and GPT-o3-mini are reproduced in Appendix B to allow exact replication.

The ablation is the critical comparison. The identically fine-tuned baseline *without* VOLTS's validation-and-search layer achieves only 0.13% validity (Blocksworld: 0.02%, Logistics: 0.15%, DriverLog: 0.13%, Rover: 0.22%). The 76.4% vs. 0.13% gap confirms that fine-tuning alone is insufficient; the integrated validation and guided parallel search are what make the SLM a reliable planner.

Domain-specific validity. VOLTS achieves 64.6% on Blocksworld, 74.6% on Logistics, 83.6% on DriverLog, and 82.8% on Rover. The lower Blocksworld validity likely reflects the tighter action interdependencies in stacking domains, where a single suboptimal early choice cascades into dead-end branches more readily than in transport-style domains. The high Rover and DriverLog scores reflect their more loosely coupled structure: in transport- and mission-style domains, most early mistakes can be recovered by later actions, so β-guided branching more easily finds a valid trajectory.

Plan length quality. Valid VOLTS plans average 1.08× Fast Downward's solution length. Domain-specific ratios: Blocksworld (1.4×), Logistics (1.01×), DriverLog (1.12×), Rover (0.77×). The Rover ratio below 1.0 arises because Fast Downward's heuristic occasionally returns suboptimal plans in this domain, while VOLTS's branching sometimes discovers shorter valid paths. The fine-tuned ablation baseline averages 5.88× baseline

length on the rare occasions it produces a valid plan. GPT-4o (0.52×) and GPT-o3-mini (0.55×) show shorter ratios, but these reflect their low validity rates, they solve only the simplest problems with inherently short plans.

5. Conclusion

We presented VOLTS, a neuro-symbolic framework that enables a 4-bit quantized Llama 3.1 8B to generate valid PDDL plans through a single-pass, logit-guided tree search with integrated real-time validation. The key finding is that the VOLTS *architecture*, not fine-tuning alone, is what makes reliable SLM-based planning possible, as demonstrated by the 76.4% vs. 0.13% validity gap against the ablation baseline. Unlike iterative LLM-Modulo or critique-and-revise approaches [2, 7, 16], VOLTS validates per token inside a single inference pass, eliminating costly external cycles. The custom PDDL vocabulary enables instant action-level validation, and dynamic β-controlled branching explores promising alternatives efficiently within a batch. By aligning with the LLM-Modulo paradigm [2] in a particularly tight integration, VOLTS opens a practical path to reliable on-device planning for resource-constrained agents where frontier LLMs [17] cannot be deployed.

Limitations and Future Work. VOLTS's reliance on substantial domain-specific training data (20k–50k examples) is a bottleneck for new domains, motivating few-shot PEFT and cross-domain transfer techniques. Future work will target explicit plan-cost optimization, support for richer PDDL features (numeric fluents, temporal actions), adaptive branching strategies, and broader application to structured generation on edge devices, including agentic systems combining VOLTS-planned actions with learned perception [18].

References

[1] M. Fox and D. Long. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 61–124. DOI: 10.1613/jair.1129.

[2] S. Kambhampati, K. Valmeekam, L. Guan, et al. "LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks". In: *Proceedings of the 41st International Conference on Machine Learning (ICML)*. Vol. 235. Proceedings of Machine Learning Research. PMLR, 2024, pp. 22895–22907. URL: https://proceedings.mlr.press/v235/kambhampati24a.html.

[3] K. Valmeekam, M. Marquez, A. Olmo, S. Sreedharan, and S. Kambhampati. "PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change". In: *Advances in Neural Information Processing Systems*. Vol. 36. 2023, pp. 38975–38987.

[4] M. Aghzal, E. Plaku, G. J. Stein, and Z. Yao. "A Survey on Large Language Models for Automated Planning". In: *arXiv* (2025). arXiv: 2502.12435.

[5] D. Gosmar and D. A. Dahl. "Hallucination Mitigation Using Agentic AI Natural Language-Based Frameworks". In: *arXiv* (2025). arXiv: 2501.13946.

[6] X. Huang, W. Liu, X. Chen, et al. "Understanding the Planning of LLM Agents: A Survey". In: *arXiv* (2024). arXiv: 2402.02716.

[7] S. Yao, J. Zhao, D. Yu, et al. "ReAct: Synergizing Reasoning and Acting in Language Models". In: *Proceedings of the 11th International Conference on Learning Representations (ICLR)*. 2023. URL: https://openreview.net/forum?id=WE_vluYUL-X.

[8] S. Yao, D. Yu, J. Zhao, et al. "Tree of Thoughts: Deliberate Problem Solving with Large Language Models". In: *Advances in Neural Information Processing Systems*. Vol. 36. 2023. URL: https://openreview.net/forum?id=5Xc1ecxO1h.

[9] M. Hu, Y. Mu, X. Yu, et al. "Tree-Planner: Efficient Close-Loop Task Planning with Large Language Models". In: *Proceedings of the 12th International Conference on Learning Representations (ICLR)*. 2024. URL: https://openreview.net/forum?id=Glcsog6zOe.

[10] K. Li, O. Patel, F. Viégas, H. Pfister, and M. Wattenberg. "Inference-Time Intervention: Eliciting Truthful Answers from a Language Model". In: *Advances in Neural Information Processing Systems*. Vol. 36. 2023. DOI: 10.5555/3666122.3667919.

[11] R. E. Fikes and N. J. Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Artificial Intelligence* 2.3–4 (1971), pp. 189–208. DOI: 10.1016/0004-3702(71)90010-5.

[12] E. J. Hu, Y. Shen, P. Wallis, et al. "LoRA: Low-Rank Adaptation of Large Language Models". In: *Proceedings of the 10th International Conference on Learning Representations (ICLR)*. 2022. URL: https://openreview.net/forum?id=nZeVKeeFYf9.

[13] F. Meng, Z. Wang, and M. Zhang. "PiSSA: Principal Singular Values and Singular Vectors Adaptation of Large Language Models". In: *Advances in Neural Information Processing Systems*. Vol. 37. 2024. URL: https://openreview.net/forum?id=6ZBHIEtdP4.

[14] M. Helmert. "The Fast Downward Planning System". In: *Journal of Artificial Intelligence Research* 26 (2006), pp. 191–246. DOI: 10.1613/jair.1705.

[15] S. Basu, G. S. Ramachandran, N. S. Keskar, and L. R. Varshney. "Mirostat: A Neural Text Decoding Algorithm that Directly Controls Perplexity". In: *Proceedings of the 9th International Conference on Learning Representations (ICLR)*. 2021. URL: https://openreview.net/forum?id=W1G1JZEIy5_.

[16] M. Besta, N. Blach, A. Kubicek, et al. "Graph of Thoughts: Solving Elaborate Problems with Large Language Models". In: *Proceedings of the 38th AAAI Conference on Artificial Intelligence*. Vol. 38. 16. 2024, pp. 17682–17690. DOI: 10.1609/aaai.v38i16.29720.

[17] OpenAI, J. Achiam, S. Adler, et al. "GPT-4 Technical Report". In: *arXiv* (2024). arXiv: 2303.08774.

[18] C. Rivera, G. Byrd, W. Paul, et al. "ConceptAgent: LLM-Driven Precondition Grounding and Tree Search for Robust Task Planning and Execution". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025.

Appendix A. VOLTS Algorithm

Algorithm 1 presents the full VOLTS inference procedure. At each iteration the fine-tuned SLM proposes candidate actions for every active branch in a single batched forward pass; the grounding-based validator filters these candidates, the primary action extends each branch, and β-controlled spawning creates new branches from promising alternatives. Figure 2 shows a Blocksworld trace illustrating how primary paths, β-spawned branches, and pruned branches are produced across multiple steps within single-pass batch inference.

Algorithm 1 VOLTS Inference (Parallel Validated Logit Tree Search)

Require: Initial PDDL state s_0, Goal G, Fine-tuned SLM M_θ, PDDL Validator V, Max branches $k_{\max}$, Branching factor β, Max plan length $T_{\max}$
Ensure: Valid PDDL plan R_{success} or Failure

Initialize active branches $\mathcal{B} \leftarrow \{(\emptyset, s_0)\}$ ▷ Set of (plan, state) tuples
Initialize iteration step $t \leftarrow 0$
while $\mathcal{B} \neq \emptyset$ **and** $t < T_{\max}$ **do**
  $\mathcal{B}_{\text{next}} \leftarrow \emptyset$ ▷ Branches for the next iteration
  $\mathcal{B}_{\text{new}} \leftarrow \emptyset$ ▷ Branches newly spawned this iteration
  Prepare batch input $\mathcal{I}$ from all $(P_i, s_i) \in \mathcal{B}$
  Compute logits for the batch: $\mathcal{L} = M_\theta(\mathcal{I})$ ▷ Returns ℓ_i per branch i
  for all branch $(P_i, s_i) \in \mathcal{B}$ with logits $\ell_i \in \mathcal{L}$ **do**
    Find top candidates $A_{\text{cand}} \leftarrow \text{TopKTokens}(\ell_i)$
    Identify valid actions $A_{\text{valid}} \leftarrow \{(a, \text{score}(a)) \mid a \in A_{\text{cand}},\ V(s_i, a) = \text{True}\}$
    if $A_{\text{valid}} = \emptyset$ **then**
      continue ▷ Prune this branch
    end if
    Select primary action $a_{\text{primary}} \leftarrow \text{argmax}_{a \in A_{\text{valid}}} \text{score}(a)$
    Update primary branch state $s'_i \leftarrow \text{ApplyAction}(s_i, a_{\text{primary}})$
    Update primary branch plan $P'_i \leftarrow P_i + [a_{\text{primary}}]$
    if $s'_i \models G$ **then**
      return P'_i ▷ Goal reached in this branch
    end if
    Add updated primary (P'_i, s'_i) to $\mathcal{B}_{\text{next}}$
    for all other valid $(a_{\text{other}}, \text{score}(a_{\text{other}})) \in A_{\text{valid}}$, $a_{\text{other}} \neq a_{\text{primary}}$ **do**
      if $\text{score}(a_{\text{primary}})/\text{score}(a_{\text{other}}) < \beta$ **and** $|\mathcal{B}| + |\mathcal{B}_{\text{new}}| < k_{\max}$ **then**
        $s^*_{\text{new}} \leftarrow \text{ApplyAction}(s_i, a_{\text{other}})$; $P^*_{\text{new}} \leftarrow P_i + [a_{\text{other}}]$
        if $s^*_{\text{new}} \models G$ **then**
          return P^*_{new} ▷ Goal reached in new branch
        end if
        Add $(P^*_{\text{new}}, s^*_{\text{new}})$ to $\mathcal{B}_{\text{new}}$
      end if
    end for
  end for
  $\mathcal{B} \leftarrow \mathcal{B}_{\text{next}} \cup \mathcal{B}_{\text{new}}$; $t \leftarrow t + 1$
end while
return Failure ▷ No plan found

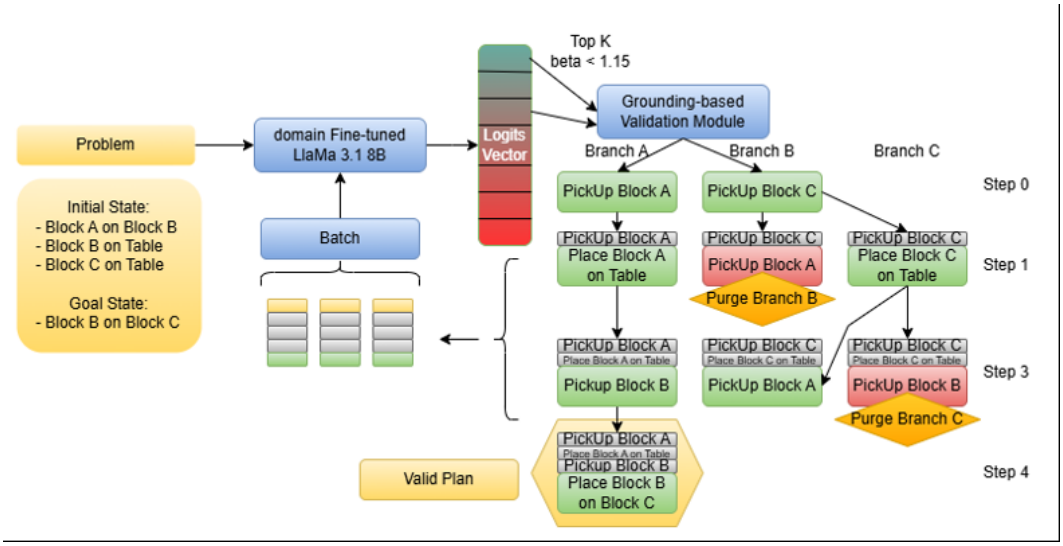

Figure 2. VOLTS logit-based tree-search trace on a Blocksworld problem. The SLM emits logit vectors over the custom action vocabulary; the grounding module validates top candidates, promoting the highest-scoring valid action (green), spawning β-branches (yellow), rejecting invalid candidates (red), and pruning dead-end branches. All branches are processed simultaneously via batched inference.

Appendix B. Prompt Templates for GPT-4o and GPT-o3-mini

For the zero-shot baselines we prompt GPT-4o and GPT-o3-mini with the full PDDL domain, one worked example problem with its known valid plan, and the target problem. The model is asked to return *only* the plan, as a flat list of grounded actions in the form `action_name, param_1, param_2, ..., param_k`, one per line. No chain-of-thought or commentary is accepted; any deviation is parsed as a malformed plan.

System: You are a PDDL plan generator. Given a PDDL domain, an example problem with its valid plan, and a target problem, you output only the plan for the target problem, with one action per line, in the format `action_name, param_1, param_2, ..., param_k`. Do not produce any other text, explanation, or formatting.

User:

`<PDDL domain definition>`

Example problem:

`<PDDL problem instance>`

Example plan:

```
pick-up, a
stack, a, b
pick-up, c
stack, c, a
...
```

Target problem:

`<PDDL problem instance>`

Plan:

The same template is used across all four domains with domain-specific PDDL text substituted in.