大規模言語モデルの探索型デコーディングにおける予算制約に整合的な探索戦略

宮本 空 [1]　大葉 大輔 [1]　岡崎 直観 [1,2,3]
[1] 東京科学大学　[2] 産業技術総合研究所　[3]NII LLMC
{sora.miyamoto@nlp.,daisuke.oba@nlp., okazaki@}comp.isct.ac.jp

概要

大規模言語モデルの探索型デコーディングは解答の精度を高めることができるが、現実には投入可能な計算予算が限られていることがある。既存手法は予算を探索方策に組み込まないため、固定予算での打ち切りと探索の優先度付けが噛み合わず、与えられた予算下での性能を最大化できない。本研究では、残予算に応じて探索の幅と深さを動的に制御し、所与の予算内で解答の精度を最大化する探索型デコーディング手法 **Budget-Guided MCTS** を提案する。数学推論タスク (MATH500/AIME) において、提案手法は予算制約下での解答の精度を向上させた。

1　はじめに

大規模言語モデル (LLM) では、複数の生成を比較・統合することで解答品質を改善できる。最も素朴な反復サンプリングでは、複数の解答候補を多数決や評価器で選別する [1]。探索型デコーディングでは Monte Carlo Tree Search (MCTS) 等で生成経路や解を探索的に洗練する [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]。

本研究では、実運用において各問題に投入できる計算予算（例：消費トークン数や計算時間）が限られている状況を考える。既存の探索型デコーディングの多くは、探索の反復回数・分岐数・幅・深さといった探索量に基づいて探索を進め、残予算を探索中の意思決定に組み込まない。そのため、予算枯渇による探索停止と探索方針が整合せず、固定予算下で性能を最大化しにくい。探索の早期打ち切りによりコスト削減を図る手法もあるが [12]、所与の予算に条件づけた探索資源の配分最適化は行わない。

本研究では、所与の予算制約下で解答品質を最大化する探索型デコーディング手法 **Budget-Guided MCTS (BG-MCTS)** を提案する。BG-MCTS では残予算に応じて探索幅と深さを適応的に制御する。これにより、予算が潤沢な局面では探索を促し、枯渇が近い局面では解への到達を優先させる。なお、LLM の出力トークン数を予算の単位とする。

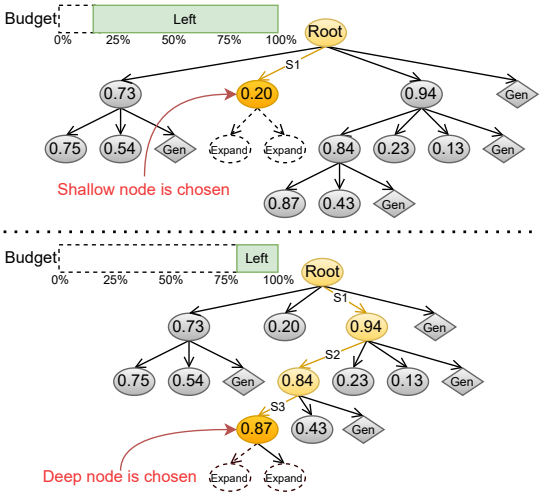


図 1: BG-MCTS における探索ノード選択の概念図。（**上部**）予算が潤沢に残っている際は浅い位置のノードを優先して選択を行い、（**下部**）予算が枯渇している際は深い位置のノードを優先して選択する。

10B 規模の LLM (Llama-3.1-8B-Instruct [13]、Qwen-2.5-7B-Instruct [14]) に対して提案手法を適用し、数学推論ベンチマークである MATH500 [15] と AIME24/25 [16, 17] を用いて評価した。出力トークン予算 10k, 20k, 30k の下で各問題を探索した結果、予算状況を探索方策に考慮しない手法と比べ、BG-MCTS は全ての予算設定で高い正解率を示した。

2　関連研究

候補生成と選別　反復サンプリングは問題に対する解答生成を独立に複数回試行し、多数決や評価器で選別する [1]。しかし、試行間で知見を蓄積・再利用することはできず、生成内容が探索的に洗練されない。一方、深さ方向に生成を限定し、単一の解答を反復的に書き換えて洗練する手法も存在す

る [10, 18]。既生成内容を条件とした改善を重ねられる反面、初期解の誤りや偏りを引きずりやすく、改善が破綻・停滞するリスクがある。

探索型デコーディング　生成過程を木構造として扱い、有望な生成経路および解候補を優先的に探索することの有効性が報告されている [2, 3, 4, 5]。AlphaGo [19] 等で用いられる PUCT に代表される探索スコアに基づき、未探索枝の試行と高評価枝の深掘りを適応的に実行する。しかし、所与の予算に条件づけた探索配分や挙動の最適化は行えない。

探索コストの削減　LiteSearch [12] は、探索木の展開量制御と探索の早期終了により、トークン消費を抑えつつ性能の維持を目指す。ただし、主眼は探索コストの削減にあり、外部から与えられる予算を探索方策に明示的に組み込んで、予算制約下で配分を最適化する枠組みではない。

本研究の位置づけ　以上を踏まえ、探索型デコーディングを「各問題にトークン予算が与えられる」設定として定式化し、残予算を状態として探索方策に明示的に組み込むことで、所与の固定予算の下で最終的な解答品質を最大化することを目的とする。

3　事前知識: MCTS

提案手法の基盤アルゴリズムである Monte Carlo Tree Search (MCTS) [2, 3, 4, 5, 6, 7, 8] の基本概念を導入する。MCTS では、子選択 → 展開 → 評価 → 伝播を反復し、探索木を更新しながら展開を行う。

子選択では、親ノード v の子 $u \in \mathcal{C}(v)$ のうち、以下の PUCT が最大の子を選択する。

$$\mathrm{PUCT}(v,u) = \underbrace{\frac{w_u}{m_u}}_{\text{活用項}} + \underbrace{c\,P(u \mid v)\sqrt{\frac{\ln(m_v)}{m_u}}}_{\text{探索項}} \tag{1}$$

$$w_v = \sum_{x \in \mathcal{T}(v)} Q(x) \tag{2}$$

m_v はノード v 以下の展開済み部分木 $\mathcal{T}(v)$ の大きさ（ノード数）[1]、c は探索強度の定数、$Q(x)$ はノード x に対して報酬モデル等で付与する評価値である。事前分布 $P(u \mid v)$ は子候補のスコアで定義する。

$$P(u \mid v) = \mathrm{softmax}(\{Q(u') \mid u' \in \mathcal{C}(v)\})_u \tag{3}$$

展開・評価では、到達した葉に固定数の子を追加し、それぞれに評価値 $Q(\cdot)$ を付与する。伝播では、新規ノードから根までの経路上の統計量 w, m を更新することで PUCT を更新しつつ探索を進める。

1)　ノード v への訪問回数とする場合もある。

4　提案: Budget-Guided MCTS

出力トークン予算を探索方策に組み込んだ探索型デコーディング手法である **Budget-Guided MCTS (BG-MCTS)** を提案する。BG-MCTS は MCTS (§ 3) を基盤としつつ、残予算に応じて i) 子選択時の探索－活用トレードオフ、および ii) 各ノードから生成する子ノード数（分岐数）を適応的に制御する。前者は探索方向、後者は探索空間に該当する。これらにより、探索初期は広い探索を重視し、終盤は解答到達を優先する挙動を狙う（図 1）。

予算とコストの取り扱い　各問題の出力に使用可能なトークン予算を B、探索の途中段階までに消費したトークン数を C_{used} とする。また、残予算の潤沢度合いを $\rho = 1 - C_{\mathrm{used}}/B$ で定義する。$\rho \in [0,1]$ は探索方策を決定する変数となる。

残予算に基づく子選択　MCTS (§ 3) と同様に、親ノード v の子 $u \in \mathcal{C}(v)$ に対する PUCT を用いるが、BG-MCTS では探索項を残予算比 ρ で重み付けする。また、活用項に用いる部分木統計を深さ補正した $\tilde{w}$ に置き換える。具体的には、子選択スコアを以下で定義し、BG-PUCT(v,u) が最大の子を選択する。

$$\mathrm{BG\text{-}PUCT}(v,u) = \underbrace{\frac{\tilde{w}_u}{m_u}}_{\text{活用項}} + \underbrace{\rho \cdot c\,P(u \mid v)\sqrt{\frac{\ln(m_v)}{m_u}}}_{\text{探索項}} \tag{4}$$

$$\tilde{w}_u = \sum_{x \in \mathcal{T}(u)} \tilde{Q}(x), \quad \tilde{Q}(x) = Q(x) + \underbrace{\kappa(1-\rho)\frac{d(x)}{\hat{d}_{\mathrm{ans}}}}_{\text{補正項}} \tag{5}$$

ここで、$\mathcal{T}(u)$ は u を根とする展開済み木のノード集合 $d(x)$ はノード x の深さである。また、$\hat{d}_{\mathrm{ans}}$ は解ノード深さの推定値であり、解ノードが存在すればその平均深さ、未到達なら展開済み木の最大深さを用いる。探索の進行に伴い残予算比 ρ が減少すると、探索項の寄与が弱まり深さ補正が強まる。そのため、初期は幅広く探索して有望枝を同定し、終盤は高評価枝の深掘りによって予算切れになる（解に至らない）事態を避けるように、挙動が遷移する。

残予算に基づく幅方向探索量の制御　親ノード v から生成する子ノード u の数（分岐数）を可変にする [3] だけでなく、子ノードを追加する親の選び方を予算状況に対応させる — 予算が潤沢なほど兄弟評価値の分散が高いノードを、また枯渇しているほど分散が低いノードを選ぶよう誘導する。具体的には、各非終端ノード v に疑似子ノード $\mathrm{gen}(v)$ を付与

し、実子ら $u \in \mathcal{C}(v)$ の BG-PUCT（式 4）と比較するための追加価値スコア $S_{\text{gen}}(v)$ を付与する:

$$S_{\text{gen}}(v) = \underbrace{\mu_v}_{\text{gen}(v)\text{ の潜在価値}} + \lambda\,\rho\,\underbrace{\sigma_v^2}_{\text{非安定性（探索価値）}} \tag{6}$$

$$\mu_v = \text{mean}_{u\in\mathcal{C}(v)}\, Q(u), \quad \sigma_v^2 = \text{var}_{u\in\mathcal{C}(v)}\, Q(u) \tag{7}$$

λ は重み定数である。子選択時において gen(v) が選択される場合には、v から実際に新規子ノードを 1 つ生成し $\mathcal{C}(v)$ に追加する。これにより、予算状況 ρ に応じ、探索方向を制御する（式 4）だけでなく、探索空間そのものを制御可能にする。

5 実験

5.1 実験設定

探索の終了判定 消費出力トークン数 C_{used} が予算 B に達した時点で終了する。終了後、解答を含みかつ評価値が最大のノードをその探索における最終解とする。解答ノードの判定には正規表現を用いる。解答の正誤判定には LightEval [20] を用いる。

ノードの構成単位 各ノードは、(i) **完全生成**：モデルが停止するまで（あるいは最大長まで）生成する方式、または (ii) **逐次生成**：所定のステップ境界まで生成する方式、のいずれかで構成する。逐次生成では、ステップ境界を表す文字列（“\nStep”）を停止条件とする。また、逐次生成で一度解答に到達した場合でも、生成末尾に “But wait, let me think about the problem again.” を付与することで、解答ノードからの探索継続を可能にする [21]。

ノード評価値 各ノード x の評価値 $Q(x)$ として、解法プロセスを評価できる報酬器が出力する値（スカラー値）を用いる。報酬器への入力は、根ノードの問題文とノード x で生成されたテキストであり、逐次生成ではステップ断片、完全生成では最終解答までを含む生成全体に対応する。

ベースライン 候補生成・選別型として Repeated Sampling [1]（幅方向）と Sequential Refinement [21]（深さ方向）を採用する。探索型デコーディングとしては、MCTS [5]、AB-MCTS-M [3]、LiteSearch [12] を採用する。MCTS 以外の基本設定は原論文に従う（付録 B）．なお公平なトークン消費比較のため LiteSearch の Greedy 事前生成は無効化する。ノードは原則として逐次生成で構成し、完全生成で評価するのは Repeated Sampling と（原論文同様）AB-MCTS-M に限る。

表 1: 正解率（Acc.，3 回試行平均）を示す。Greedy は探索なしの生成結果である。Full と添字がある手法では各ノードを完全生成で構成している。太字は予算 B ごとの最良スコア、下線は次点を表す。

	Math500 (Lv.5)			AIME24/25		
Methods \ Budget B	10K	20K	30K	10K	20K	30K
Llama-3.1-8B-Instruct						
- Greedy	.224	.224	.224	.033	.033	.033
- Refinement$_{\text{Full}}$	.249	.246	.241	.033	.028	.028
- Repeated$_{\text{Full}}$	**.393**	.438	**.450**	.039	.050	.056
- AB-MCTS-M$_{\text{Full}}$	.311	.323	.343	.050	.050	.050
- AB-MCTS-M	.124	.179	.209	.000	.011	.011
- MCTS	.333	.406	.430	.039	.072	.089
- LiteSearch-Incre.	.249	.291	.304	.033	.039	.056
- LiteSearch-batch	.236	.271	.281	.033	.044	.039
- BG-MCTS (ours)	**.393**	**.465**	.445	**.072**	**.083**	**.101**
Qwen-2.5-7B-Instruct						
- Greedy	.493	.493	.493	.100	.100	.100
- Refinement$_{\text{Full}}$	.498	.493	.490	.094	.083	.089
- Repeated$_{\text{Full}}$	.632	.664	.702	**.156**	.161	**.183**
- AB-MCTS-M$_{\text{Full}}$	.629	.657	.659	.144	.150	.156
- AB-MCTS-M	.433	.542	.600	.072	.128	.139
- MCTS	.619	.657	.659	**.156**	.167	.167
- LiteSearch-Incre.	.488	.510	.510	.089	.083	.083
- LiteSearch-batch	.527	.557	.557	.078	.083	.083
- BG-MCTS (ours)	**.662**	**.699**	**.711**	**.156**	**.189**	**.183**

BG-MCTS の基本設定 葉での新規ノード展開数を 2、探索定数を $c = \sqrt{2}$ とする。また、補正係数（式 5）および分散項係数（式 6）は $\kappa = \lambda = 1$ とする。

モデルとデータ 生成モデルには Llama-3.1-8B-Instruct [13] と Qwen2.5-7B-Instruct [14] を、報酬器には GenPRM-7B [22, 23] を用いる。評価には数学タスクの MATH500 [15] と AIME24/25 [16, 17] を用いる。

5.2 実験結果

主な結果 表 1 の通り、BG-MCTS は全ての予算制約下において、予算を明示的に考慮しないベースライン手法を上回る正解率を達成した。これは、計算予算の枯渇による打ち切りではなく、残予算を探索方策に組み込み、停止条件と優先度を整合させることが、固定予算下で有効であることを示唆する。

予算に応じた探索挙動 図 3 は、Qwen-2.5-7B-Instruct を MATH500 (Lv.5) で評価した際の解答到達率を示す。BG-MCTS は探索序盤では試行を広げ、残予算が減る局面で解答到達率が大きく伸びる傾向を示すことから、探索から解到達への遷移という設計意図通りの挙動が伺える。同様に図 2 では、消費出力トークン数が予算 B に近づくにつれて解答正解

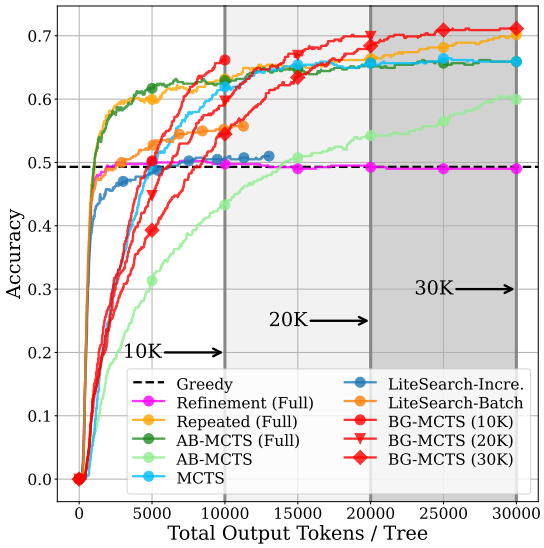


図 2: 消費出力トークンに対する解答正解率 (Acc.) の変化。使用モデルは Qwen2.5-7B-Instruct、ベンチマークは Math500 (Lv.5) である。

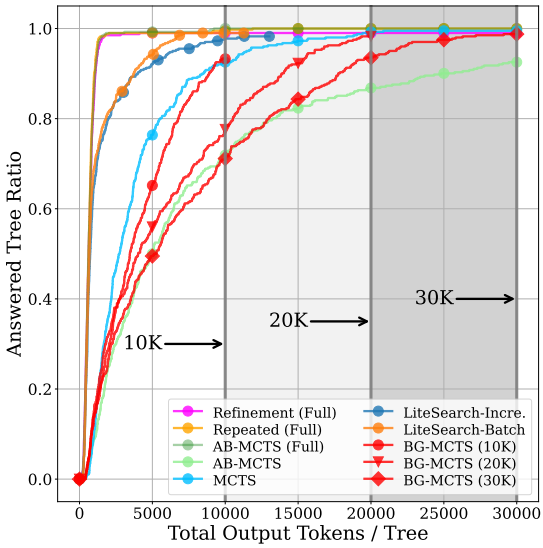


図 3: 各アルゴリズムの解答済みノードを持つ探索木の割合。使用モデルは Qwen2.5-7B-Instruct、ベンチマークは Math500 (Lv.5) である。

表 2: BG-MCTS の各要素を無効化した際の正解率 (Acc.)．2 回試行の平均値を示す。赤字は無効化前からの低下を、下線は MCTS を下回ることを示す。

	Math500 (Lv.5)		
Methods \ Budget B	10K	20K	30K
Llama-3.1-8B-instruct			
- MCTS	.333	.406	.430
- BG-MCTS (ours)	.393	.465	.445
- w/o 活用項の補正項 (式 5)	.433	.425	.485
- w/o 探索項の減衰 (式 4)	.373	.403	.429
- w/o 探索幅の動的制御 (式 6)	.407	.470	.433
Qwen-2.5-7B-instruct			
- MCTS	.619	.657	.659
- BG-MCTS (ours)	.662	.699	.711
- w/o 活用項の補正項 (式 5)	.631	.690	.720
- w/o 探索項の減衰 (式 4)	.605	.660	.679
- w/o 探索幅の動的制御 (式 6)	.642	.675	.657

率が伸び、枯渇のタイミングに合わせて正解率が高まる挙動が見て取れる。表 1 の結果と併せると、予算制約に整合した探索配分が固定予算下の精度向上に寄与していることが分かる。また B が大きいほど探索初期に幅方向へ割ける余裕が増え、深掘りに有利な候補を早期に確保しやすいことが示唆される。

ベースラインの傾向 (AB-MCTS-M)　表 1 より、AB-MCTS-M は完全生成で Repeated Sampling を下回り、逐次生成でも MCTS を下回る場合がある。原論文では明示的フィードバックに基づく修正誘導が可能な設定で有効性が示されている一方 [3]、本研究では報酬器のスカラー値のみを手掛かりとするため修正信号が弱く、同様の改善が得られにくいと考えられる。実際に解答到達が遅い傾向にある（図 3）。

ベースラインの傾向 (LiteSearch)　LiteSearch は探索トークン数を抑制できる一方で、正解率は相対的に低い（表 1、図 2）。また、解答ノード到達率が早期に増加する（図 3）ことから、コスト削減を優先して探索が早期に収束し、局所解に陥って予算を十分に活用できていない可能性が示唆される。

分析　表 2 に BG-MCTS の各要素の寄与を示す。活用項の補正（式 5）、探索項の残予算依存性（式 4）、展開幅制御（式 6）はいずれも有効で、特に探索項の残予算依存性を無効化すると多くの設定に負の影響があった。これは終盤でも探索が抑制されず、予算を解到達に分配できないためだと考えられる。

6　おわりに

LLM の探索型デコーディングにおいて所与の予算内で探索配分を制御する **Budget-Guided MCTS** を提案した。数学推論タスクでの実験により、残予算の枯渇に伴い i) 探索の重点を幅優先から深さ優先へ遷移させ、併せて ii) 収束に近いノードに新規子ノードを追加することの有効性を示した。

今後は、出力だけでなく入力トークンや評価器計算も含む予算定義へ拡張する。また、複数評価器併用時の配分設計および正解率 – 計算量トレードオフの体系的検証を進めたい。

謝辞

本研究は、文部科学省の補助事業「生成 AI モデルの透明性・信頼性の確保に向けた研究開発拠点形成」の支援を受けた。本研究は、東京科学大学のスーパーコンピュータ TSUBAME4.0 を利用して実施した。

参考文献

[1] Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large Language Monkeys: Scaling inference compute with repeated sampling. arXiv:2407.21787, 2024.

[2] Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Lei Han, Haitao Mi, and Dong Yu. Toward self-improvement of LLMs via imagination, searching, and criticizing. In **Advances in Neural Information Processing Systems**, pp. 52723–52748, 2024.

[3] Yuichi Inoue, Kou Misaki, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. Wider or deeper? scaling LLM inference-time compute with adaptive branching tree search. In **The Thirty-ninth Annual Conference on Neural Information Processing Systems**, 2025.

[4] Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte Carlo Tree Search for comprehensive exploration in LLM-based automatic Heuristic Design. In **Forty-second International Conference on Machine Learning**, 2025.

[5] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In **Proceedings of the 41st International Conference on Machine Learning**, 2024.

[6] Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Lillicrap Timothy P, Kenji Kawaguchi, and Michael Shieh. Monte Carlo Tree Search boosts reasoning via iterative preference learning. In **The First Workshop on System-2 Reasoning at Scale, NeurIPS'24**, 2024.

[7] Di Zhang, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. Accessing GPT-4 level mathematical olympiad solutions via Monte Carlo Tree Self-refine with LLaMa-3 8B. arXiv:2406.07394, 2024.

[8] Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus Mcaleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. In **International Conference on Machine Learning**, pp. 49890–49920, 2024.

[9] Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Alphamath almost zero: Process supervision without process. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, **Advances in Neural Information Processing Systems**, Vol. 37, pp. 27689–27724. Curran Associates, Inc., 2024.

[10] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate problem solving with large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, **Advances in Neural Information Processing Systems**, Vol. 36, pp. 11809–11822. Curran Associates, Inc., 2023.

[11] Chaojie Wang, Yanchen Deng, Zhiyi Lyu, Liang Zeng, Jujie He, Shuicheng Yan, and Bo An. Q*: Improving multi-step reasoning for LLMs with deliberative planning. arXiv:2406.14283, 2024.

[12] Ante Wang, Linfeng Song, Ye Tian, Baolin Peng, Dian Yu, Haitao Mi, Jinsong Su, and Dong Yu. LiteSearch: Efficacious tree search for LLM, 2024.

[13] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. arXiv:2407.21783, 2024.

[14] Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. arXiv:2412.15115, 2025.

[15] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. arXiv:2305.20050, 2023.

[16] Maxwell-Jia. Aime 2024 dataset. https://huggingface.co/datasets/Maxwell-Jia/AIME_2024, 2024. Accessed: 2026-01-02.

[17] OpenCompass. Aime 2025 dataset. https://huggingface.co/datasets/opencompass/AIME2025, 2025. Accessed: 2026-01-02.

[18] 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**, Vol. 36, pp. 46534–46594, 2023.

[19] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering Chess and Shogi by self-play with a general reinforcement learning algorithm. **Nature**, No. 529, pp. 484–489, 2016.

[20] Nathan Habib, Clémentine Fourrier, Hynek Kydlíček, Thomas Wolf, and Lewis Tunstall. LightEval: A lightweight framework for LLM evaluation, 2023. https://github.com/huggingface/lighteval.

[21] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candes, and Tatsunori Hashimoto. s1: Simple test-time scaling. In **Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing**, pp. 20275–20321, 2025.

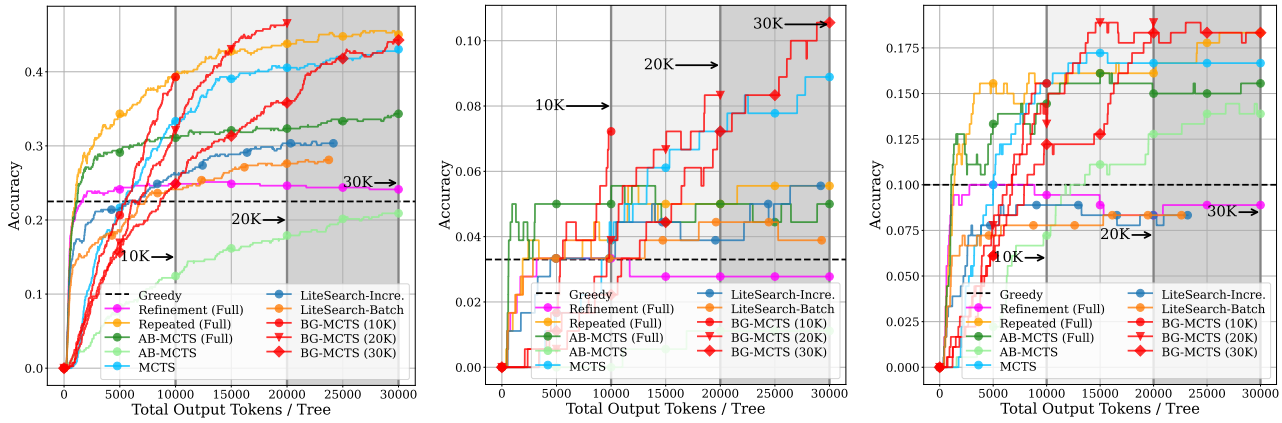
[22] Jian Zhao, Runze Liu, Kaiyan Zhang, Zhimu Zhou, Junqi Gao, Dong Li, Jiafei Lyu, Zhouyi Qian, Biqing Qi, Xiu Li, and Bowen Zhou. GenPRM: Scaling test-time compute of process reward models via generative reasoning. arXiv:2504.00891, 2025.

[23] Runze Liu, Jian Zhao, Kaiyan Zhang, Zhimu Zhou, Junqi Gao, Dong Li, Jiafei Lyu, Zhouyi Qian, Biqing Qi, Xiu Li, and Bowen Zhou. Awesome process reward models. https://github.com/RyanLiu112/Awesome-Process-Reward-Models, 2025. GitHub repository.

A　付録: 各種生成モデルを採用した際の実験結果

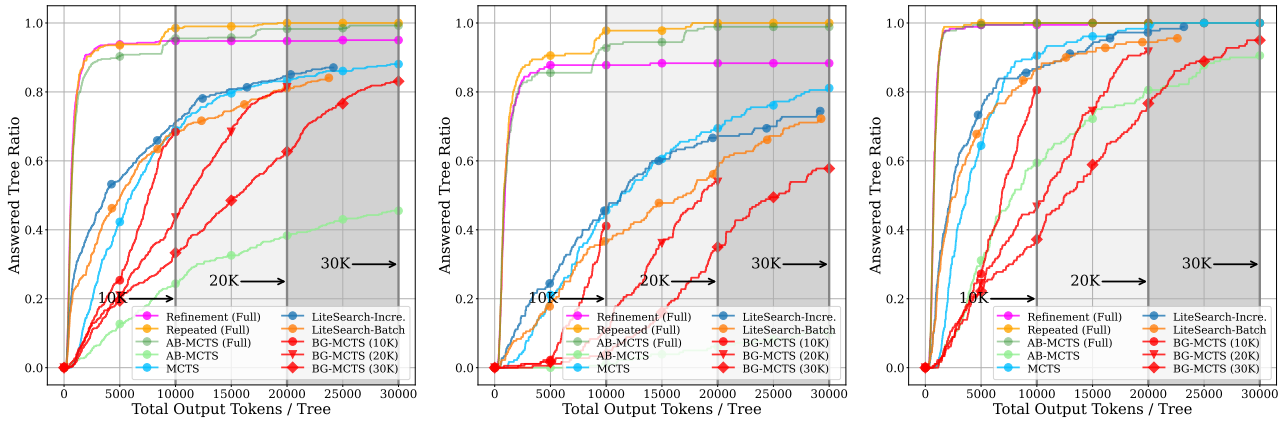
本文で導入した実験（表 1）以外のモデル・ベンチマーク設定について、消費出力トークン数に対する正解率（Acc.）の推移を図 4 に、解答済みノードを含む探索木の割合（解答到達率）図 5 に示す。BG-MCTS では、消費出力トークンが予算 B に近づくにつれて正解率と解答到達率の双方が増加しており、残予算に応じた探索挙動が機能していることが分かる。この傾向は予算設定を変えても（B = 10K/20K/30K）一貫して観察される。

さらに注目すべき点として、BG-MCTS の解答到達率はベースラインよりも低い傾向がある。これは、解の早期到達を優先して解答ノードを増やすよりも、探索初期に幅広く候補枝を探索して有望枝を同定し、限られた予算をより高品質な解の探索に配分することが、最終的な正解率の改善に寄与し得ることを示唆する。



(a) Llama-3.1-8B-Instruct / Math500 Lv.5　(b) Llama-3.1-8B-Instruct / AIME24/25　(c) Qwen2.5-7B-Instruct / AIME24/25

図 4: **消費出力トークンに対する解答正解率（Acc.）の変化**



(a) Llama-3.1-8B-Instruct / Math500 Lv.5　(b) Llama-3.1-8B-Instruct / AIME24/25　(c) Qwen2.5-7B-Instruct / AIME24/25

図 5: **消費出力トークンに対する解答済みノードを持つ探索木の割合**

B　付録: 木探索アルゴリズムのハイパーパラメータ探索

ベースラインである MCTS (§ 3) については，論文ごと設定が異なるため [2, 3, 4, 5, 6, 7, 8] ハイパーパラメータ探索を行った。具体的には、PUCT（式 1）における探索項の重み $c = \{\sqrt{0.1}, \sqrt{1.0}, \sqrt{2.0}\}$、および固定拡張ノード数 = $\{2, 3, 5\}$ に関して、Llama-3.1-8B-Instruct を用いてパラメータ探索を行った。ベンチマークは Math500 の level-5 を用い、予算は $B = 15K$ の範囲とした。検証の結果、もっとも性能の良かった $c = \sqrt{2}$、固定拡張ノード数 = 2 を採用した。