Search Into Weights: Turning Test-Time Search into Capability per Parameter

Anonymous authors
Paper under double-blind review

Abstract

Test-time search produces both candidate solutions and feedback that can train a small model during inference. We propose a memory-conditioned draft model that serves as a speculative proposer and a within-problem learner. An append-only memory stores representations of previously sampled, target-verified rollouts and their outcome scores; gated cross-attention exposes these records to the draft. Verification distributions provide a distillation objective, while outcome-weighted replay provides an auxiliary learning signal. We distinguish two mechanisms that are often conflated: under exact speculative sampling, adapting the draft changes sampling cost but not the frozen target's conditional output distribution; direct changes to search quality require an explicit draft-dependent selection rule. We give sufficient conditions for exact sampling with adaptive memory, describe replay bookkeeping, and derive a compute ledger and a conditional break-even criterion. We specify a matched-budget evaluation on mathematical reasoning, including a decoder-only control given the same rollout information. This paper presents a method and analysis with an evaluation protocol; empirical improvements in accuracy, acceptance length, and architectural efficiency remain untested.

1 Introduction

Search exposes a language model to completed continuations and feedback that were unavailable before those computations. We ask whether a small draft can compress this search history into its parameters and a memory interface during a single problem. Here, grounded futures means previously sampled continuations relative to an earlier search node, not access to future observations or to information outside the computation history. A causal decoder can also receive this history as context; our architectural hypothesis concerns the cost and usefulness of its representation.

Scaling test-time compute is now a primary lever for reasoning in large language models (Snell et al., 2024). Most of that compute is spent with frozen weights: self-consistency draws more samples, and tree search draws deeper and wider ones. Separately, speculative decoding makes each sampled token cheaper by letting a small draft model propose tokens that a large target verifies in parallel, with no change to the target's output distribution (Leviathan et al., 2023; Chen et al., 2023). And test-time training (TTT) shows that a few gradient steps on the test instance itself can buy capability that a frozen model lacks (Sun et al., 2020; Akyürek et al., 2024).

Existing work combines speculative proposals with search and adapts draft models online. SpecSearch (Wang et al., 2025) studies speculative search, while OnlineSpec (Qian et al., 2026) studies online draft adaptation. We explore a per-problem adaptation design with rollout memory and two training signals, without claiming that combining these ingredients alone establishes novelty or improvement.

A crucial distinction is whether target verification is exact token-level rejection sampling or a task-level selection procedure. Exact rejection sampling removes the draft's effect on the emitted conditional token distribution. In that regime, adaptation can improve reasoning

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.

accuracy at a fixed budget only indirectly, through computational savings that permit more search, or through an explicitly draft-dependent search controller. We therefore do not infer better reasoning proposals from a higher token acceptance rate. Direct proposal-quality effects must be evaluated in a separately identified, non-exact search variant.

The proposed memory stores summaries of target hidden states over completed rollouts together with scores and tree statistics. Its zero-initialized gate recovers the base draft at initialization. The comparison with an in-context decoder using the same records is essential: memory supplies a different interface to available information, not an information-theoretic advantage over every causal decoder.

Contributions.

- Method. A draft model that is simultaneously the speculative proposer, an MCTS proposer and a per-instance test-time learner, trained with a dual objective combining a verification-distillation loss (using logits the target already computes) and an outcome-weighted replay loss (§3).
- Grounded-future memory. A position-free, append-only memory of sampled-and-verified futures read through gated cross-attention, with a cold start that recovers the base draft exactly (§3.3).
- Correctness. Two propositions: burst-frozen memory is sufficient for exact speculative sampling, and per-segment versioning supports replay under partial rollouts that cross update rounds (§4).
- Cost analysis. A FLOP ledger for memory-conditioned speculative search and a break-even inequality for test-time training whose two savings terms partition cleanly, usable as an adaptive trigger (§5).
- Protocol. A matched-FLOP evaluation contract, including the ablation that separates the architectural channel from the search it consumes (§6).

2 Related work

Speculative decoding. Speculative sampling (Leviathan et al., 2023; Chen et al., 2023) guarantees exact samples from the target given any fixed proposal distribution. Medusa (Cai et al., 2024) and EAGLE (Li et al., 2024; 2025) improve acceptance by reusing target features in the drafter; we use target features as a memory representation, inspired by feature-based drafting. These methods motivate learning better proposals for inference acceleration.

Online draft adaptation. OnlineSpec (Qian et al., 2026) is our nearest neighbour. It adapts the draft from verification feedback, framed as dynamic-regret minimization with optimistic gradient reuse and an online ensemble. We differ in three ways. Objective: OnlineSpec optimizes acceleration; our proposed draft is conditioned on search history and also trained on scored rollouts; possible acceleration and task-level gains must be measured separately. Regime: OnlineSpec accumulates across a request stream; we train per instance and reset. Information: OnlineSpec has no search and no lookahead channel; our memory injects verified futures. We use verification feedback as one component of the objective.

Speculative reasoning and search. SpecSearch (Wang et al., 2025) inserts speculative drafting into beam search and MCTS as node expansion, adapting only a scalar rejection threshold; its theory bounds quality degradation relative to the target. SpecReason (Pan et al., 2025) delegates easy reasoning steps to a small model, also with frozen weights. SpecSearch is a related baseline; our exact token-verification variant must not be equated with its step-level search procedure.

Search-trained reasoning. rStar-Math, ReST-MCTS*, AlphaMath and MCTS-DPO (Guan et al., 2025; Zhang et al., 2024; Chen et al., 2024; Xie et al., 2024) distil search outcomes into the main policy across problems. We train the draft, within one problem, and discard the update.

Table 1: Positioning. P1: per-instance test-time parameter updates. P2: a draft verified by a larger target. P3: model used in a search procedure.

	P1 per-instance TTT	P2 draft + verify	P3 search use
OnlineSpec	– (stream-accumulated)	✓	–
SpecSearch / SpecReason	– (frozen)	✓	✓
rStar-Math / ReST-MCTS* family	– (offline, main policy)	–	✓
TTT / ARC-TTT	✓ (main model)	–	–
Ours	✓ (draft, reset)	✓	✓ (dual objective)

Test-time training. TTT adapts parameters on the test input through a self-supervised or task loss (Sun et al., 2020), and has proved effective for few-shot reasoning (Akyürek et al., 2024). TTT layers (Sun et al., 2024) make the hidden state itself a learner. These works provide complementary approaches to inference-time adaptation.

3 Method

3.1 Setup

Let π_T be a frozen target reasoner, which acts as the speculative verifier and as the terminal judge, and never reads the memory. Let q_ϕ be a small draft model that shares the target's tokenizer (required for token-level acceptance). The draft body is frozen; the trainable state ϕ consists of low-rank adapters and a gated cross-attention encoder (§3.3). For each problem x we run MCTS over reasoning steps, with rollouts indexed by j and TTT rounds indexed by r. At the start of every problem, $\phi \leftarrow \phi_0$.

A burst is one uninterrupted drafting episode: one node expansion or one rollout continuation. Within a burst the adapter and the memory snapshot are held fixed; Proposition 1 gives a sufficient condition for correctness.

3.2 Three roles, one parameter set

Proposer. The draft proposes γ tokens; the target verifies them in one parallel pass under the standard rejection rule, with acceptance rate α. Search policy. Raw candidates are proposed by q_ϕ. After exact rejection sampling, emitted continuations follow the target. Search selection uses a fixed controller and logged target-derived scores in the exact variant. Learner. Between rounds, ϕ is updated from signals the search already produced (§3.4).

3.3 Grounded-future memory

When rollout j completes we append an entry $e_j = (\bar{h}_j, u_j, z_j)$ to the memory M, where $\bar{h}_j$ is the pooled target hidden state over the rollout's verified tokens, projected from the target width to the draft width (these states are already computed by verification, as in EAGLE); u_j embeds the outcome or verifier verdict; and z_j embeds tree statistics (visit count, value estimate, relative depth).

M carries no positional encoding. Futures from different branches share no time axis, so M is semantically a set; insertion order is kept only for storage and versioning. Keys and values are therefore relocatable and can be extended incrementally, and a snapshot is a prefix slice $M_{\le \ell}$. We define a memory version as $v = (r, \ell)$: the round r fixes the key and value projections and ℓ fixes the content.

Every k-th draft layer gains a gated cross-attention block (Alayrac et al., 2022):

$$h \leftarrow h + \tanh(g) \cdot \mathrm{Attn}\big(hW_Q,\ K(M_{\le \ell}),\ V(M_{\le \ell})\big), \qquad g \text{ initialized to } 0. \tag{1}$$

With $M = \emptyset$ or $g = 0$ the draft is exactly the base draft, so the initial distribution is a baseline and the growth of g is itself a measurement.

Grounding constraint. Entries come only from futures that were actually sampled and verified by the target, never from the draft's own predictions. These entries are part of the available search history. They may also be supplied to a causal decoder as context. Target verification establishes sampling consistency, not semantic correctness; outcome scores are separate and may be noisy.

3.4 Dual test-time objective

We minimize $\mathcal{L}(\phi) = \mathcal{L}_{\text{acc}} + \lambda\,\mathcal{L}_{\text{qual}}$, with gradients through all trainable components of ϕ, including the cross-attention projections and gates; the base-model weights remain fixed.

Acceptance loss. Verifying γ drafted tokens yields the target's next-token distribution at each drafted position as a by-product, at no extra target FLOPs. With s_t the state at drafted position t,

$$\mathcal{L}_{\text{acc}} = \sum_{t \in \text{drafted}} \mathrm{KL}_k\big(\pi_T(\cdot \mid s_t) \,\big\|\, q_\phi(\cdot \mid s_t, M_v)\big), \tag{2}$$

where KL_k groups all tokens outside the target's top-k set into one bucket, with each distribution assigning that bucket its total remaining probability mass. Shipping full rows is infeasible at modern vocabulary sizes (about 0.5 MB per position for a 248k vocabulary), whereas serving engines expose top-k log-probabilities natively. Rejected positions can provide useful feedback and are included: their states condition on the drafted prefix, but the target's distributions there were computed under exactly that prefix, so the term is well-defined off-policy distillation.

Outcome-weighted replay. Let y_j be a completed, target-distributed rollout, with a score R_j from a fixed inference-time judge. We use a weighted maximum-likelihood auxiliary objective,

$$\mathcal{L}_{\text{qual}} = -\sum_j w_j \sum_t \log q_\phi(y_{jt} \mid y_{j,<t}, x, M_{v_{jt}}), \qquad w_j = \frac{\exp(R_j/\tau)}{\sum_i \exp(R_i/\tau)}, \quad \tau > 0. \tag{3}$$

The memory snapshot for each segment contains only records available before that segment was sampled. This is supervised replay on scored target samples, not an unbiased policy-gradient estimator for the draft. In particular, corrected target tokens must not be assigned draft behaviour probabilities. The auxiliary loss can conflict with acceptance distillation; λ is an ablation parameter, and neither objective guarantees improved acceptance or accuracy.

Benchmark reference answers are used only for final evaluation. Online scores must come from the frozen judge, executable checks available at deployment, or another explicitly reported inference-time verifier. A judge's approval is not evidence of mathematical correctness.

3.5 Algorithm

The method is piecewise stationary: search runs in rounds under a frozen adapter, and updates happen between rounds (Algorithm 1). Entries appended within a round are visible to later bursts in the same round, because ℓ grows while earlier keys and values stay valid (they are position-free and the adapter is fixed). Round size trades adaptation frequency against batching throughput and is an evaluation variable.

4 Correctness

Proposition 1 (Exactness under memory conditioning). If ϕ and $M_{\le \ell}$ are fixed within a drafting burst, speculative sampling with proposal $q_\phi(\cdot \mid \cdot, M_{\le \ell})$ emits exact samples from π_T.

Proof. The acceptance argument of Leviathan et al. (2023) requires, at each position, a fixed and valid proposal distribution that is known at verification time; it makes no assumption about how that distribution is computed. Accept a drafted token y with probability

Algorithm 1 Test-time-trained draft for speculative search (one problem)

$\phi \leftarrow \phi_0$; $M \leftarrow \emptyset$; $r \leftarrow 0$
while FLOP budget remains do
  for each expansion or rollout burst in round r do
    $\ell \leftarrow |M|$ ▷ burst snapshot
    speculative decoding with $q_{\phi_r}(\cdot \mid \cdot, M_{\le \ell})$; target verifies
    log (segment, top-k target log-probs, reward, version $v = (r, \ell)$)
    if a rollout completes then append e_j to M
  end for
  $\phi_{r+1} \leftarrow \phi_r - \eta \nabla_\phi \mathcal{L}(\phi)$ ▷ k inner steps on round-r logs
  re-project $K(M), V(M)$ under ϕ_{r+1}; $r \leftarrow r + 1$
end while
return answer selected by the fixed judge; reset $\phi \leftarrow \phi_0$

$\min(1, \pi_T(y)/q(y))$ and on rejection resample from the normalized residual $\max(0, \pi_T - q)$. For any fixed q the emitted token is distributed as π_T. With ϕ and $M_{\le \ell}$ fixed, $q_\phi(\cdot \mid s_t, M_{\le \ell})$ is such a distribution at every position t of the burst, so the argument applies position by position. □

Corollary 1 (No direct distributional quality gain). For a fixed prefix and fixed target, the conditional distribution of tokens emitted by exact speculative sampling is independent of the draft adapter and its memory. Thus changes in draft proposal probabilities alone cannot change that emitted distribution.

Proof. Apply Proposition 1 separately to each proposal distribution: each produces the same target conditional distribution. This statement concerns the sampling kernel, not the distribution of answers after budget-dependent stopping or search selection. □

Freezing a burst is sufficient, not necessary. An adaptive implementation can also be exact if it retains the actual proposal probabilities used to sample each token and applies the corresponding acceptance and residual correction. Recomputing a different proposal during verification is invalid.

Proposition 2 (Sufficient replay bookkeeping). An immutable append-only memory, logged snapshot lengths, and stored behaviour adapter versions suffice to reconstruct the draft distribution used for a raw proposal segment, subject to deterministic replay. Alternatively, logged proposal probabilities suffice for its behaviour likelihood on observed tokens.

Proof. The snapshot length identifies the entries read by the segment, and the stored adapter determines its projections and conditional distribution. Replaying those inputs recovers that distribution. If only a likelihood ratio on observed tokens is required, storing the relevant behaviour likelihoods directly avoids reconstruction. Support of the behaviour policy on those tokens is additionally required. □

A rollout spanning bursts must retain each segment's memory version. These records support the supervised objective above, but do not justify a draft importance ratio on target-corrected tokens. A separate raw-proposal policy-gradient variant would need its own sampling law, reward assignment and support analysis.

5 Cost analysis

We count FLOPs at $2N$ per forward token and take backward as about twice forward. Let c_T and c_q be the target and draft forward costs per token, $c_{\mathrm{x}}(\ell)$ the cross-attention cost per draft token per adapter layer (linear in ℓ), and L_{ad} the number of adapter layers. The method's draft pays for reading memory on every proposed token: $c'_q = c_q + L_{\mathrm{ad}}\, c_{\mathrm{x}}(\ell)$. Baseline drafts pay c_q.

Speculative ledger. Let $C_{\text{verify}}(\gamma)$ include every target forward operation needed for a verification cycle, including any pending prefix position and bonus-token computation. Under an i.i.d. acceptance approximation,

$$\mathbb{E}[L] = \sum_{i=0}^{\gamma} \alpha^i, \qquad f(\alpha, \ell) = \frac{\gamma c'_q(\ell) + C_{\text{verify}}(\gamma)}{\mathbb{E}[L]}. \tag{4}$$

For a conventional implementation that evaluates $\gamma + 1$ target positions per cycle, a simple approximation is $C_{\text{verify}} = (\gamma{+}1)c_T$. Then even at full acceptance, $f = c_T{+}\gamma c'_q/(\gamma{+}1) > c_T$: parallel verification can reduce latency without reducing arithmetic work relative to target-only decoding. Cache conventions and measured execution determine the actual ledger. Acceptance improvements reduce cost relative to a poorer speculative draft, but do not by themselves establish a FLOP advantage over plain target decoding.

Break-even. Let $C_{\text{step}} \approx 3c'_q n$ be the cost of one TTT step on n tokens (adapter-only gradients still require a backward pass through the body), and C_{reproj} the cost of re-projecting memory keys and values after an update. Test-time training has positive estimated compute savings under this simplified accounting when

$$\underbrace{\big[f(\alpha_0, \ell_0) - f(\alpha_r, \ell_r)\big]\, T_{\text{rem}}}_{\text{cheaper tokens}} + \underbrace{\Delta R \cdot C_{\text{rollout}}}_{\text{rollouts avoided}} > \sum_r \big(k\, C_{\text{step}} + C_{\text{reproj}} + C_{\text{entry}}\big), \tag{5}$$

where T_{rem} counts tokens of rollouts that still occur, and ΔR counts rollouts that no longer occur. The two savings terms partition the savings: a token saved by not rolling out is never also counted as cheaper. The measured break-even point, in tokens or search nodes per problem, is the threshold below which TTT should not trigger, which could inform an adaptive policy after calibration. Here C_{entry} includes memory construction costs. For the exact variant with a fixed controller and fixed work count, there is no established quality-based reduction in rollouts: set $\Delta R = 0$ unless a separately measured controller effect justifies it. The inequality is retrospective accounting, not a proof that those savings occur.

Mixture-of-experts verification. For sparse targets, arithmetic accounting should use the experts actually evaluated for each token, while latency also depends on batching, expert routing, communication and memory movement. The number of distinct experts touched by a batch is not a substitute for per-token FLOP counting. Report both measured arithmetic estimates and elapsed time, including training and memory overhead.

6 Evaluation protocol

We commit in advance to the following contract, so that the method's claims can be tested rather than tuned.

Matched compute. Every method receives the same per-problem FLOP budget B, enforced by a hard governor. Baselines convert B into more samples or deeper search. The primary metric is accuracy at B, swept over B to trace a Pareto curve; the secondary metric is wall-clock time at fixed accuracy.

Baselines. (1) Target chain-of-thought; (2) self-consistency with N samples; (3) target-only MCTS; (4) speculative MCTS without TTT (our exact sampling configuration, with SpecSearch evaluated separately); (5) TTT without search; (6) the full method.

Benchmarks. Competition mathematics (MATH-500 and AIME), with held-out reference answers for evaluation and a separately specified inference-time scoring procedure for training and answer selection.

Mechanism measurements. (a) Acceptance length as a function of round within a problem, isolating the speed channel. (b) With the target frozen, the selection rate and value of adapted-draft proposals versus base-draft proposals at the same nodes, testing raw-proposal quality in a separately labelled non-exact selection variant. This measurement cannot establish a quality effect for the exact sampling kernel. (c) The trajectory of the cross-attention gate g.

Accounting commitments. (i) The architecture is never credited with search's gains: the decisive ablation gives a decoder-only draft with the same parameter count the same rollouts in context, at matched compute. The difference isolates the cross-attention channel. (ii) Cross-attention, re-projection and entry-projection FLOPs are charged to the method. (iii) Target costs use measured activated experts.

Ablations. Memory content (target-hidden summaries versus outcome and tree statistics only); cross-attention adapter versus soft-token memory at matched parameters and compute; the mixing weight λ; round size; γ; and per-instance reset versus accumulation across problems (to measure, not assume, the value of reset).

7 Discussion and limitations

The main limitation of this submission is that it does not yet contain empirical results; it presents the method, its correctness properties, its cost analysis and a pre-committed protocol. Beyond that, three risks are specific to the method. First, per-instance TTT may not pay back on easy problems; the break-even inequality makes this measurable and turns it into a trigger policy. Second, reviewers may ask whether the method is simply a faster MCTS; measurement (b) and the in-context decoder-only ablation are designed to answer exactly that question. Third, the method requires a draft that shares the target's tokenizer, which may require distilling or truncating a draft from the target family.

The architecture is not specific to reasoning. Any setting that produces verified futures, such as resolved events in live forecasting benchmarks or observed outcomes in agentic environments, can populate the memory, and the same zero-initialized cross-attention channel applies. Testing whether gains along this forward dimension transfer across such settings is the natural next step.

The proposed system raises security questions we intend to study: adversarial inputs that inflate test-time budgets, reward hacking of the verifier, and poisoning of the test-time training signal. Because the draft updates on signals derived from the input, these are properties of any system that learns at inference time, not of this method alone.

8 Conclusion

We proposed one architectural addition, a future-state encoder over grounded futures, that gives a language model a compressed interface to its search history. In the reasoning instantiation, this is a way to spend test-time compute that neither freezes the model nor retrains the large one: train the small draft, within the problem, on the signals search already produces, and let it read verified futures. Whether updates improve acceptance or final reasoning performance is an empirical question; exact verification preserves the target distribution regardless of the draft. We gave conditions under which this preserves exact sampling and consistent replay bookkeeping, a ledger that says when it pays, and a protocol to test it.

AI use statement

We used generative AI tools to assist with literature search, verification of citation metadata, and drafting and editing of the manuscript text from the authors' notes. Generative AI also assisted with method formalization, mathematical analysis, and revisions to the proposed objectives and correctness arguments. The manuscript reports no empirical results. The author is responsible for checking the final text, analysis, citations, and any subsequently released artifacts.

Reproducibility statement

The proposed method is described in §3 and Algorithm 1; assumptions and proofs for both propositions are in §4; the cost model and all FLOP terms are in §5; and the evaluation

contract, including baselines, budgets and ablations, is in §6. An executable implementation, hyperparameters, and empirical results are not included in this submission.

References

Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, et al. The surprising effectiveness of test-time training for few-shot learning. arXiv preprint arXiv:2411.07279, 2024.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, et al. Flamingo: a visual language model for few-shot learning. In Advances in Neural Information Processing Systems, 2022.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, et al. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. arXiv preprint arXiv:2401.10774, 2024.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. arXiv preprint arXiv:2302.01318, 2023.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. AlphaMath almost zero: Process supervision without process. arXiv preprint arXiv:2405.03553, 2024.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, et al. rStar-Math: Small LLMs can master math reasoning with self-evolved deep thinking. arXiv preprint arXiv:2501.04519, 2025.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In International Conference on Machine Learning, 2023.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. arXiv preprint arXiv:2401.15077, 2024.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. arXiv preprint arXiv:2503.01840, 2025.

Rui Pan, Yinwei Dai, Zhihao Zhang, Gabriele Oliaro, Zhihao Jia, and Ravi Netravali. SpecReason: Fast and accurate inference-time compute via speculative reasoning. arXiv preprint arXiv:2504.07891, 2025.

Yu-Yang Qian, Hao-Cong Wu, Yichao Fu, Hao Zhang, and Peng Zhao. When drafts evolve: Speculative decoding meets online learning. arXiv preprint arXiv:2603.12617, 2026.

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.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In International Conference on Machine Learning, 2020.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, et al. Learning to (learn at test time): RNNs with expressive hidden states. arXiv preprint arXiv:2407.04620, 2024.

Zhihai Wang, Jie Wang, Jilai Pan, Xilin Xia, Huiling Zhen, Mingxuan Yuan, et al. Accelerating large language model reasoning via speculative search. arXiv preprint arXiv:2505.02865, 2025.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P. Lillicrap, Kenji Kawaguchi, et al. Monte carlo tree search boosts reasoning via iterative preference learning. arXiv preprint arXiv:2405.00451, 2024.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. ReST-MCTS*: LLM self-training via process reward guided tree search. arXiv preprint arXiv:2406.03816, 2024.