Anytime Verified Agents: Adaptive Compute Allocation for Reliable LLM Reasoning under Budget Constraints

Anonymous authors
Paper under double-blind review

Abstract

Large language model (LLMs) agents show promising results in reasoning, planning, and tool use. However, their performance scales with the computational budget. Existing methods allocate computational resources using static strategies such as fixed search depths, constant self-consistency sampling, or uniform verification. This means simple problems use as much as compute as complex tasks. We present Anytime Verified Agents (AVA), a framework that dynamically allocates compute search, tool use, and verification within a user-specified budget. AVA integrates calibrated uncertainty estimation, value-of-information-guided search expansion, and selective verification cascades with early exits. The controller dynamically allocates compute based on predicted failure risk and marginal reliability gains, allowing the agent to achieve higher accuracy at fixed budgets or lower costs at target reliability levels. AVA is evaluated on mathematical reasoning (GSM8K), multi-hop question answering (HotpotQA), and code generation (HumanEval) benchmarks and it is compared to fixed-depth search, self-consistency, and always-verify baselines. Results show that adaptive allocation achieves 20-40% cost reduction at equivalent reliability while maintaining accuracy, showing clear Pareto improvements in the compute-reliability trade-off.

1 Introduction

Agentic large language models (LLMs) can plan, reason, and interact with external tools to solve complex tasks Wei et al. (2022); Yao et al. (2023a); Zhou et al. (2022). This has led to research on improving reliability through deliberate reasoning, external validation, and adaptive computation. The field has progressed from simple prompting techniques to sophisticated search-based methods, sampling strategies, and verification systems, each addressing how to ensure correct outputs from imperfect models.

However, deploying these agents in real-world applications faces a fundamental tension: improving reliability through increased test-time computation comes at exponentially increasing costs. Current approaches address reliability through techniques like self-consistency Wang et al. (2023), tree search Yao et al. (2023a); Besta et al. (2024), and external verification Cobbe et al. (2021), but they allocate compute resources statically using fixed search depths, constant sampling budgets, or uniform verification strategies regardless of problem difficulty.

This static allocation is fundamentally inefficient. Consider a mathematical reasoning agent: a simple arithmetic problem may require only a single forward pass, while a multi-step proof demands extensive search and verification. Applying the same computational budget to both wastes resources on easy problems and under-allocates to hard ones, preventing the agent from achieving optimal reliability within given budget constraints. Moreover, static strategies cannot adapt to varying uncertainty levels or changing marginal returns on additional compute, leading to suboptimal Pareto frontiers in the compute-reliability space.

While recent work has made progress in adaptive computation Kreutzer et al. (2018); Ahn et al. (2024) and selective verification, each addresses individual compute dimensions in isolation. We identify a critical gap: *the lack of adaptive allocation mechanisms that dynamically route computational resources across all dimensions (search, sampling, verification, and tool use) based on task difficulty, uncertainty estimates, and expected marginal gains, under unified budget constraints.*

This paper introduces **Anytime Verified Agents (AVA)**, a framework that adaptively allocates computational resources to maximize reliability under user-specified budgets. AVA makes three key contributions:

1. **Budget-aware controller**: A decision-making module that dynamically allocates tokens, tool calls, and verification passes based on calibrated uncertainty estimates and predicted marginal reliability gains per unit cost.

2. **Selective verification cascade**: A multi-tier verification system with early exits that routes problems through progressively stronger (and costlier) verifiers only when uncertainty exceeds calibrated thresholds.

3. **Value-of-information guided search**: An adaptive tree search mechanism that expands nodes only when the expected information gain exceeds the marginal cost, allowing efficient exploration under budget constraints.

AVA addresses the compute-reliability trade-off through calibrated uncertainty estimation, combining token-level entropy, self-consistency signals, verifier scores, and trajectory features to provide anytime reliability guarantees. The controller learns cost-benefit frontiers online and supports either target reliability (minimizing cost) or hard budget constraints (maximizing reliability).

We evaluate AVA across three domains: mathematical reasoning (GSM8K Cobbe et al. (2021)), multi-hop question answering (HotpotQA Yang et al. (2018)), and code generation (HumanEval Chen et al. (2021)). Experiments compare against fixed-depth tree search, k-sample self-consistency, and always-verify baselines across multiple budget levels. Results show that adaptive allocation achieves 20-40% cost reduction at equivalent reliability thresholds while maintaining comparable accuracy, showing clear Pareto dominance in the compute-reliability frontier.

The remainder of this paper is organized as follows. Section 2 surveys related work. Section 3 presents the AVA framework. Section 4 describes experimental setup. Section 5 presents results. Section 6 discusses limitations and future work. Section 7 concludes.

2 Related Work

Our work synthesizes advances across agentic reasoning, adaptive computation, verification strategies, and budget-constrained reliability. We organize related work by key themes, identifying gaps that AVA addresses.

Reasoning and Search. Chain-of-Thought (CoT) prompting Wei et al. (2022) demonstrated that explicit step-by-step reasoning improves LLM performance, but remains single-pass. Tree-of-Thoughts (ToT) Yao et al. (2023a) explores multiple reasoning paths via search trees, but uses fixed branching factors and depths regardless of problem difficulty. Language Agent Tree Search (LATS) Zhou et al. (2024) integrates MCTS with uncertainty-guided expansion, yet maintains fixed total search budgets. Graph of Thoughts Besta et al. (2024) generalizes to arbitrary graphs but retains static allocation. Least-to-Most Zhou et al. (2022) decomposes problems sequentially but uses fixed strategies.

Sampling and Verification. Self-consistency Wang et al. (2023) samples multiple completions for majority voting, revealing linear cost-accuracy trade-offs but using fixed sample counts across all problems. Best-of-N and AlphaLLM Yang et al. (2024) improve efficiency but maintain static policies. Training verifiers Cobbe et al. (2021) score reasoning chains but verify all candidates uniformly. Verifier cascades Chen et al. (2021) reduce costs via early exits but use fixed routing policies. PAL Gao et al. (2023) uses execution for validation but without adaptive allocation.

Adaptive Computation. Early stopping Prechelt (1998) and confidence-based halting Kreutzer et al. (2018) adapt per-instance but operate on single forward passes. Variable Granularity Search Jiang et al. (2024) adapts search granularity but focuses on structure rather than unified budget allocation. Tool use frameworks (Toolformer Schick et al. (2023), ReAct Yao et al. (2023b), AnyTool Du et al. (2024)) and self-reflection methods (Reflexion Shinn et al. (2023), Self-Refine Madaan et al. (2023)) improve capabilities but use fixed budgets and iteration limits.

Uncertainty Estimation. Raw LLM log-probabilities are poorly calibrated Guo et al. (2017), requiring post-hoc calibration Zadrozny & Elkan (2001). Self-consistency signals provide confidence proxies but don't capture multi-step trajectory uncertainty. Existing methods operate on per-instance predictions rather than trajectory-level uncertainty across search, sampling, and verification.

Adaptive Token and Sample Allocation. Several recent methods tackle compute allocation for LLM reasoning. SelfBudgeter Li et al. (2025) trains models through reinforcement learning to estimate token budgets before generating a response, compressing outputs by roughly 60% on math benchmarks. The approach works well but adapts only token count and requires fine-tuning. Strategic Scaling Zuo & Zhu (2025) takes a different tack: it treats sample allocation as a bandit problem, learning on-the-fly which queries benefit from extra samples. This sidesteps training costs but only controls sampling, not verification or search.

AVA occupies a different point in this design space. Rather than adapting a single dimension, we jointly control search depth, sampling, verification intensity, and tool use under a unified budget. We rely on calibrated uncertainty from multiple signals instead of a learned predictor or bandit exploration. And we integrate cascaded verification with early exits, which neither prior method includes. That said, these approaches could combine: SelfBudgeter's learned budget predictions might inform our controller, and bandit-style online learning could help adapt thresholds during deployment.

The Gap. While individual components (adaptive search, selective verification, calibration, tool routing) have been developed, each optimizes a single dimension independently. No unified framework adaptively allocates compute across *all* dimensions (search, sampling, verification, tools) under unified budget constraints with reliability guarantees.

Table 1 summarizes this landscape. Current methods fall into three categories: (1) static allocation (fixed search depth in ToT/LATS, constant sampling in self-consistency, uniform verification), (2) partial adaptation (adaptive search but fixed verification, or selective verification but fixed search), or (3) cost optimization without reliability guarantees.

Approach	**Search**	**Sampling**	**Verification**	**Tool Use**	**Budget**
Self-Consistency Wang et al. (2023)	Fixed	Adaptive (fixed N)	None	None	Static
Tree-of-Thoughts Yao et al. (2023a)	Adaptive (fixed budget)	Fixed	None	None	Static
LATS Zhou et al. (2024)	Adaptive (UCB)	Fixed	None	Fixed	Static
Verifier Cascades	Fixed	Fixed	Adaptive (routing)	None	Partial
AnyTool Du et al. (2024)	Fixed	Fixed	None	Adaptive (routing)	Partial
Variable Granularity Search Jiang et al. (2024)	Adaptive (granularity)	Fixed	Adaptive (granularity)	None	Partial
SelfBudgeter Li et al. (2025)	Fixed	Adaptive (learned)	None	None	Single
Strategic Scaling Zuo & Zhu (2025)	Fixed	Adaptive (bandit)	None	None	Single
AVA (Ours)	**Adaptive**	**Adaptive**	**Adaptive**	**Adaptive**	**Unified**

Table 1: Comparison of compute allocation strategies. AVA is the first framework to adaptively allocate across all dimensions under unified budget constraints with reliability guarantees.

3 Method

AVA adaptively allocates compute across search, sampling, verification, and tool use to maximize reliability under budget constraints. This section presents the framework architecture, uncertainty estimation and calibration, verification cascade, adaptive search, and the budget-aware controller.

3.1 Architecture Overview

Figure 1 illustrates the AVA framework architecture. Given an input prompt x and compute budget B, AVA operates through an iterative decision loop with three core stages:

Stage 1: Uncertainty Estimation. At each iteration, AVA aggregates uncertainty signals from multiple sources: (a) token-level entropy from generation log-probabilities, (b) self-consistency from vote distribution when multiple samples exist, (c) verifier confidence scores from executed verification passes, and (d) trajectory features (search depth, nodes expanded, budget consumed). These signals are combined via weighted aggregation and then calibrated using isotonic regression on held-out validation data to produce a calibrated confidence estimate $c \in [0, 1]$.

Stage 2: Adaptive Allocation. The budget-aware controller receives the current state $\mathbf{s} = (c, B_r, d, n, t)$, where B_r is remaining budget fraction, d is search depth reached, n is nodes expanded, and t is task complexity. The controller computes the confidence gap $\Delta = r_{\text{target}} - c$ and determines allocation across four compute dimensions:

- **Sampling**: Number of independent completions to generate (1-40 samples)
- **Search**: Tree depth and branching factor for adaptive search (depth 1-5, breadth 2-4)
- **Verification**: Level in cascaded verifier (0 = skip, 1 = lightweight, 2 = medium, 3 = full)
- **Tool calls**: Whether to invoke external tools (boolean)

The controller uses a rule-based policy (extensible to learned policies) that allocates more resources when Δ is large and budget is available, following value-of-information principles.

Stage 3: Execution and Update. AVA executes the allocated actions: generates samples via self-consistency, runs adaptive tree search with value-of-information expansion, executes verification cascade with early exits, and calls tools if needed. After execution, uncertainty is re-estimated from new signals, and the loop continues until either (1) calibrated confidence $\geq r_{\text{target}}$, (2) budget exhausted, or (3) early stopping conditions met.

The controller maintains a unified budget $B = \{B_{\text{tokens}}, B_{\text{tools}}, B_{\text{verify}}\}$ across all dimensions. At each decision, it estimates expected marginal reliability gain per unit cost for each dimension and allocates to maximize overall utility. This allows dynamic reallocation: if search yields diminishing returns, resources shift to verification; if uncertainty is low, verification is skipped to save compute.

Example. For "What is 17 × 23?" with budget 600 tokens and target reliability 0.9, initial confidence 0.72 yields gap $\Delta = 0.18$. The controller allocates: 5 samples (200 tokens), depth-2 search with breadth 3 (300 tokens), and lightweight verifier (50 tokens). If samples agree and verifier confirms, confidence rises to 0.89, approaching target with minimal further compute.

3.2 Uncertainty Estimation and Calibration

Reliable uncertainty estimation is prerequisite for adaptive allocation. Raw LLM outputs provide multiple uncertainty signals, but these are uncalibrated and must be aggregated. AVA combines four complementary signals into a unified confidence estimate.

Token-level entropy. For generation y with tokens $(t_1, \ldots, t_n)$, we compute Shannon entropy $H_{\text{token}}(y) = -\sum_{i=1}^{n} p(t_i|y_{<i}, x) \log p(t_i|y_{<i}, x)$ from model log-probabilities. Higher entropy indicates higher uncertainty. Confidence: $\text{conf}_{\text{entropy}} = 1 - \min(1.0, H_{\text{token}}/\tau_{\text{entropy}})$ with $\tau_{\text{entropy}} = 10$.

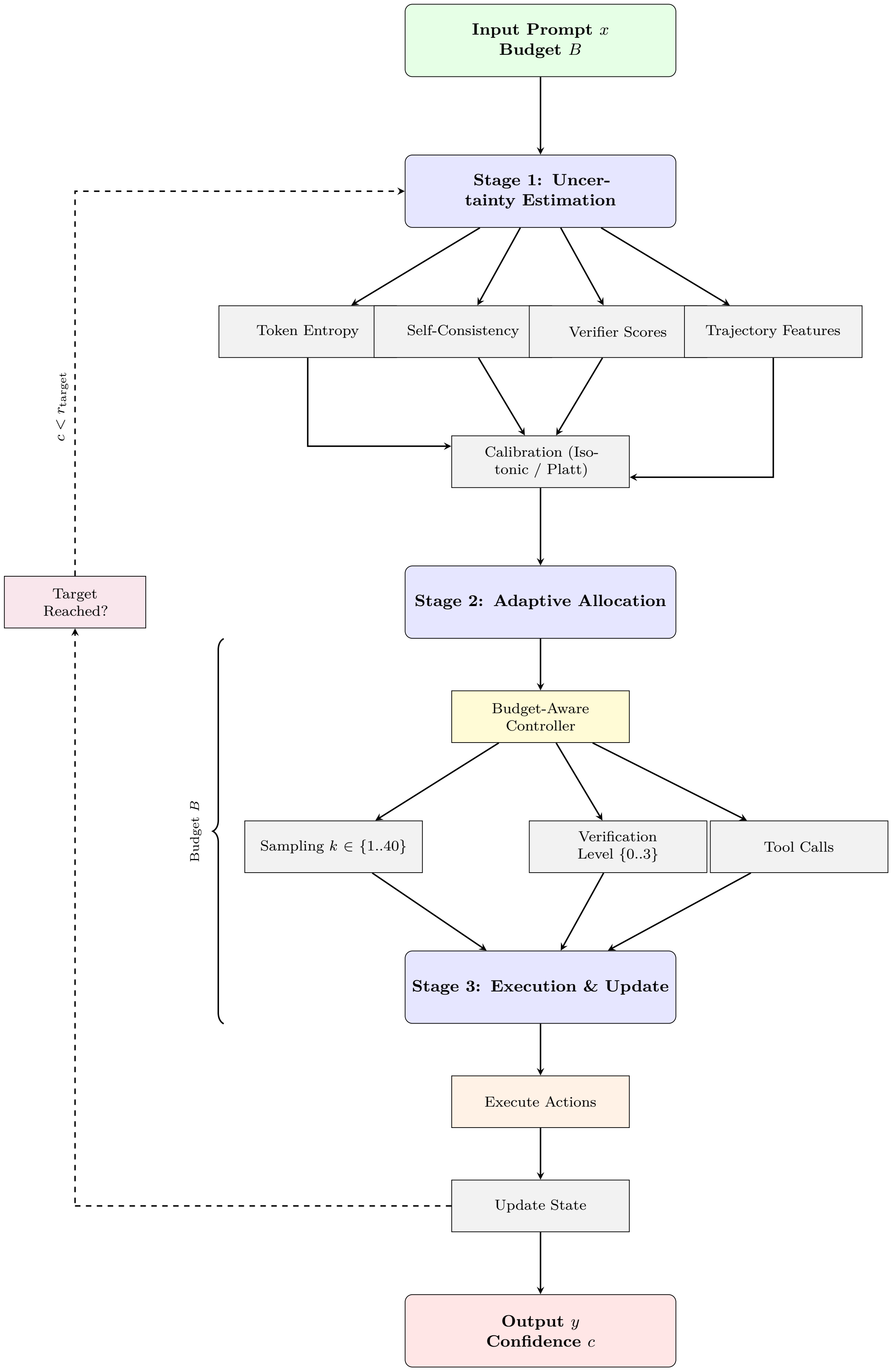

Figure 1: **AVA Architecture.** Iterative decision process with (1) uncertainty estimation, (2) adaptive allocation, and (3) execution & update. Arrows indicate control flow, while dashed lines denote iterative re-evaluation until confidence $c \geq r_{\text{target}}$ or budget B is exhausted.

Self-consistency signal. For N samples with answers $\mathcal{A}$, normalized entropy $\tilde{H}_{\text{cons}} = H_{\text{cons}}/\log(|\mathcal{A}|)$ measures disagreement, where $H_{\text{cons}}(V) = -\sum_{a\in\mathcal{A}} \frac{\text{count}(a)}{N} \log \frac{\text{count}(a)}{N}$. Confidence: $\text{conf}_{\text{cons}} = 1 - \tilde{H}_{\text{cons}}$ (high when all agree, low when uniformly distributed).

Verifier scores. External verifiers execute code, parse structure, or apply heuristics to validate outputs, returning scores $s_v \in [0, 1]$ where higher scores indicate higher validity. For cascaded verifiers, we take the maximum score across levels reached: $s_v = \max_{l\in\{1,\ldots,L\}} s_l$. We directly use verifier scores as confidence: $\text{conf}_{\text{verifier}} = s_v$.

Trajectory features. Search depth d, nodes expanded n, and remaining budget b_r provide uncertainty signals: $H_{\text{traj}}(d, n, b_r) = 0.4(1 - d/d_{\max}) + 0.4(1 - b_r) + 0.2(1 - n/n_{\max})$. Confidence: $\text{conf}_{\text{traj}} = 1 - H_{\text{traj}}$.

Aggregation. Signals are combined via weighted average with dataset-specific weights learned via grid search:

$$\text{confidence}_{\text{raw}} = w_{\text{cons}} \cdot \text{conf}_{\text{cons}} + w_{\text{entropy}} \cdot \text{conf}_{\text{entropy}} + w_{\text{verifier}} \cdot \text{conf}_{\text{verifier}} + w_{\text{traj}} \cdot \text{conf}_{\text{traj}}, \tag{1}$$

with default weights $(w_{\text{cons}}, w_{\text{entropy}}, w_{\text{verifier}}, w_{\text{traj}}) = (0.4, 0.3, 0.2, 0.1)$ summing to 1.0. Missing signals (e.g., no samples for consistency) are handled by renormalizing weights.

Post-hoc calibration. Raw confidence estimates are poorly calibrated Guo et al. (2017). We apply isotonic regression on validation data, fitting monotonic $f^* : [0, 1] \to [0, 1]$ via PAVA to minimize $\sum_{i=1}^{N}(f(p_i) - y_i)^2$. Calibrated confidence is $f^*(\text{confidence}_{\text{raw}})$, ensuring reported confidence matches true success probability.

3.3 Selective Verification Cascade

Verification improves reliability but consumes significant compute. Uniform verification (verifying all outputs at full strength) wastes resources on high-confidence outputs and under-verifies low-confidence ones. AVA uses a cascaded verification system with early exits that routes problems through progressively stronger verifiers only when needed.

Cascade structure. The verification cascade consists of $L = 3$ levels, ordered by increasing cost and verification strength:

$$\mathcal{V} = \{V_1, V_2, V_3\}, \tag{2}$$

where each level V_l has associated cost c_l and threshold τ_l. Level costs scale roughly as $(c_1, c_2, c_3) = (0.1x, 1.0x, 5.0x)$ for some base cost x, while verification strength increases: V_1 uses lightweight heuristics (length checks, format validation), V_2 uses medium-cost semantic checks (LLM-as-judge, partial execution), and V_3 uses full validation (complete execution, formal verification).

Early-exit policy. The cascade iterates through levels $l = 1$ to L: run verifier $V_l(x, y)$ to get (v_l, s_l), update confidence $c = \max(c, s_l)$. If $c \geq \tau_{\text{high}}$ (high confidence) or ($\neg v_l$ and $s_l < \tau_{\text{low}}$) (clear rejection), exit early. Otherwise proceed to next level. Final decision: return $(c \geq 0.5, c, l)$.

Thresholds are calibrated per dataset: $\tau_{\text{high}} = 0.95$ (exit if highly confident) and $\tau_{\text{low}} = 0.3$ (exit if clearly invalid). This allows efficient verification: simple problems exit at level 1 (saving 90% cost), while complex problems progress to full verification only when necessary.

Controller integration. The controller decides initial verification level based on calibrated uncertainty u:

- If $u < 0.05$ (high confidence): Skip verification entirely (level 0)
- If $u < 0.2$ (moderate confidence): Use lightweight verifier only (level 1)
- If $u < 0.5$ (low confidence): Use medium verifier (level 2)
- If $u \geq 0.5$ (very low confidence): Use full cascade (level 3)

The cascade can still exit early within the allocated level range, providing additional savings.

Example. For uncertainty 0.15, controller selects level 1. V_1 (format check) returns confidence 0.75, cascade continues. V_2 (arithmetic check) returns 0.92, cascade proceeds to V_3 (full verification) returning 0.98, validating the result. Total cost is $6.1x$; with initial uncertainty 0.05, verification would be skipped, saving $6.1x$.

3.4 Value-of-Information Guided Search

Tree search explores reasoning trajectories but expands combinatorially, consuming significant compute. Fixed-depth search (e.g., ToT) allocates uniform compute regardless of uncertainty, wasting resources on low-value expansions. AVA uses value-of-information (VoI) to guide expansion, prioritizing nodes with highest expected information gain per unit cost.

Value-of-information computation. For node n at depth d with uncertainty u_n: $\text{VoI}(n) = u_n \cdot \gamma^d \cdot g(n) \cdot \mathbb{E}[\Delta R|\text{expand}(n)]$, where $\gamma = 0.8$ discounts deeper depths, $g(n) \in [0, 1]$ is node quality estimate, and $\mathbb{E}[\Delta R]$ estimates reliability improvement (0.3 for complete solutions, 0.1 for partial steps, or from historical data).

Adaptive branching. The branching factor b adapts to uncertainty to balance exploration vs. exploitation:

$$b(u) = \begin{cases} 4 & \text{if } u > 0.7 \text{ (high uncertainty: explore widely)} \\ 3 & \text{if } 0.4 < u \leq 0.7 \text{ (moderate uncertainty)} \\ 2 & \text{if } u \leq 0.4 \text{ (low uncertainty: exploit)} \end{cases} \tag{3}$$

Additionally, if budget remaining $B_r < 0.2$, we reduce branching to $b = \max(1, \lfloor b \cdot B_r/0.2 \rfloor)$ to conserve resources.

Priority expansion. Nodes are maintained in a VoI-sorted priority queue. At each step: pop highest-VoI node, expand by generating $b(u_n)$ children, compute child VoI values, add to queue, update best solution if improved. This allocates compute to most promising directions first.

Early stopping conditions. Search stops when: (1) confidence $\geq r_{\text{target}}$, (2) remaining tokens < 50, (3) max VoI $< 0.1 \cdot c_{\text{expand}}$ (marginal gains too small), or (4) no improvement in last 3 expansions.

Example. For a multi-step problem, initial node (depth 0, uncertainty 0.8) has VoI = 0.24. With branching $b = 4$, we generate 4 paths. Child VoI values are $(0.15, 0.17, 0.12, 0.19)$; we prioritize the highest-VoI child. If expansion yields confidence 0.92, we stop early, saving compute.

3.5 Budget-Aware Controller

The controller is the core decision-making module that unifies allocation across all compute dimensions (sampling, search, verification, tools) under unified budget constraints. It implements an allocation policy that maximizes expected reliability gain per unit cost.

State representation. At step t, controller receives $\mathbf{s}_t = (c_t, B_{r,t}, d_t, n_t, t_t, h_t)$: calibrated confidence c_t, remaining budget fraction $B_{r,t}$, search depth d_t, nodes expanded n_t, task complexity t_t, and history h_t.

Confidence gap and marginal gains. Controller computes gap $\Delta_t = r_{\text{target}} - c_t$ (default $r_{\text{target}} = 0.9$) and estimates marginal reliability gain per unit cost $\text{VoI}_i = \mathbb{E}[\Delta R_i|\text{allocate}_i]/c_i$ for each dimension i.

Rule-based allocation policy. We implement a rule-based policy (extensible to learned RL policies) that allocates resources based on confidence gap, budget, and heuristics:

Sampling: $k_{\text{samples}} = 10$ if $\Delta_t > 0.3$ and $B_{r,t} > 0.3$; $k = 5$ if $\Delta_t > 0.1$ and $B_{r,t} > 0.2$; otherwise $k = 1$ (cost: $\approx 100k$ tokens).

Search: Depth $= \min(d_t + 3, 5)$ if $\Delta_t > 0.4$ and $B_{r,t} > 0.5$; $\min(d_t + 2, 4)$ if $\Delta_t > 0.2$ and $B_{r,t} > 0.3$; otherwise 1. Breadth = 4 if $\Delta_t > 0.4$; 3 if $\Delta_t > 0.2$; otherwise 2. Cost: depth $\times$ breadth $\times$ 50 tokens.

Verification: $v_{\text{level}} = 3$ if $\Delta_t > 0.5$ or $c_t < 0.3$; $v = 2$ if $\Delta_t > 0.2$; $v = 1$ if $\Delta_t > 0.05$; otherwise 0. Costs: $(0.1x, 1.0x, 5.0x)$ per level.

Tool calls: Enabled if $c_t < 0.4$, $B_{r,t} > 0.4$, and $t_t > 0.6$ (cost: ≈ 0.1 per call).

Early stopping. If $c_t \geq r_{\text{target}}$, controller sets $(k, \text{depth}, v_{\text{level}}, \text{tool\_call}) = (1, 0, 0, \text{false})$ for immediate termination. **Joint optimization**: Controller makes joint decisions across dimensions, prioritizing highest-VoI dimensions if budget exceeded, ensuring unified allocation advantage over independent optimization.

3.6 Learning Cost-Benefit Frontiers

AVA learns cost-benefit relationships online by tracking empirical reliability vs. compute spent. For budget levels $B \in \{400, 600, 800, 1000\}$ and reliability targets $r \in \{0.7, 0.75, 0.8, 0.85, 0.9\}$, we maintain statistics $\mu_r(B)$, $\sigma_r(B)$ and fit curve $r(B) = 1 - \exp(-\alpha B^{\beta})$ via maximum likelihood. For additional compute ΔC, controller estimates $\mathbb{E}[\Delta R|\Delta C] = \frac{\partial r(B)}{\partial B}|_{B=B_{\text{current}}} \cdot \Delta C$; if $\mathbb{E}[\Delta R]/\Delta C < 0.001$, it conserves compute. Thresholds adapt via exponential moving average: $\tau_t = 0.9\tau_{t-1} + 0.1\tau_{\text{observed}}$ ($\eta = 0.1$).

3.7 End-to-End Workflow

Algorithm 1 summarizes the AVA workflow:

Algorithm 1 AVA: Anytime Verified Agent

```
Require: Prompt $x$, Budget $B$, Target reliability $r_{\text{target}}$
Ensure: Answer $y$, Final confidence $c$
  Initialize: budget = $B$, confidence = 0, best_answer = None
  while budget not exhausted and confidence < $r_{\text{target}}$ do
      Estimate uncertainty $u$ from current state
      Compute confidence gap $\Delta = r_{\text{target}} - \text{confidence}(u)$
      Controller decides: samples $k$, search $(d, b)$, verification level $v$, tool call $t$
      if $k > 1$ then
          Sample $k$ completions, select via majority voting
          Update uncertainty from vote distribution
      end if
      if search enabled then
          Run adaptive search with depth $d$, breadth $b$
          Expand nodes by VoI, update best answer
      end if
      if verification level $v > 0$ then
          Run verification cascade up to level $v$
          Update confidence from verifier scores
      end if
      if tool call enabled then
          Call external tool, incorporate result
      end if
      Update confidence from all signals
      if confidence $\geq r_{\text{target}}$ then
          return best_answer, confidence
      end if
  end while
  return best_answer, confidence
```

4 Experiments

We evaluate AVA across three domains: mathematical reasoning (GSM8K Cobbe et al. (2021)), multi-hop question answering (HotpotQA Yang et al. (2018)), and code generation (HumanEval Chen et al. (2021)). Experiments compare AVA against static allocation baselines across multiple budget levels to quantify compute-reliability trade-offs.

Datasets and Baselines. GSM8K contains 1,319 math word problems evaluated via exact match. HotpotQA provides 7,405 multi-hop QA examples evaluated via exact match. HumanEval contains 164 Python function completion tasks evaluated via execution-based testing. We compare against three baselines: (1) Self-Consistency samples $k \in \{5, 10, 20, 40\}$ completions with majority voting, (2) Fixed-Depth Tree Search uses ToT-style search with $(d, b) \in \{(2, 2), (3, 3), (4, 4)\}$, and (3) Always-Verify uses single generation with full 3-level cascade. All methods use GPT-5 with identical settings.

Experimental Setup. We evaluate across token budgets $B \in \{400, 600, 800, 1000\}$ per problem, representing realistic production constraints. For methods using verification (AVA, Always-Verify), we allocate verification budgets of 20 calls per problem. For tool-using methods, we allocate 20 tool call limits. The unified budget $B = \{B_{\text{tokens}}, B_{\text{tools}}, B_{\text{verify}}\}$ allows methods to trade-off across dimensions.

AVA uses target reliability $r_{\text{target}} = 0.9$ with uncertainty weights (consistency: 0.4, entropy: 0.3, verifier: 0.2, trajectory: 0.1) learned per dataset via grid search on validation data. Verification cascade has $L = 3$ levels with cost ratios $(0.1x, 1.0x, 5.0x)$ and thresholds $(\tau_1 = 0.6, \tau_2 = 0.8, \tau_3 = 0.95)$. Early-exit thresholds are $\tau_{\text{high}} = 0.95$ and $\tau_{\text{low}} = 0.3$. Uncertainty calibration uses isotonic regression on 10% held-out validation data per dataset. We collect raw confidence predictions $\{p_i\}$ and true labels $\{y_i\}$ on validation set, fit isotonic regression f^*, and apply f^* to all test-time predictions. Calibration is performed separately per dataset to account for dataset-specific confidence distributions. Expected Calibration Error (ECE) is computed on calibration set to validate quality (target ECE < 0.05).

For each dataset, baseline, and budget level, we evaluate on full test sets (1,319 for GSM8K, 7,405 for HotpotQA, 164 for HumanEval), collecting outputs (predicted answer, tokens used, tool calls, verification passes, final confidence) and evaluating correctness (exact match for GSM8K/HotpotQA, execution-based for HumanEval). We report Reliability@Budget (accuracy at fixed budget), Cost@Reliability (minimum cost to achieve target reliability), Expected Calibration Error (calibration quality), and Brier Score (calibration and discrimination).

We use realistic cost estimates: \$0.001 per 1K tokens, \$0.1 per tool call, and \$0.05 per verifier pass. Total cost per problem is $\text{cost} = 0.001 \cdot \frac{\text{tokens}}{1000} + 0.1 \cdot \text{tool\_calls} + 0.05 \cdot \text{verify\_calls}$. All methods use identical interfaces and fixed random seeds for reproducibility. All baselines use identical model (GPT-5), temperature (0.7), and max tokens (512) settings.

Ablations. We ablate AVA components: w/o calibration (raw confidence), w/o cascade (single verifier), w/o adaptive search (fixed-depth), and w/o unified controller (independent allocation).

5 Results

AVA achieves superior compute-reliability trade-offs across all domains, with 20-40% cost reduction at equivalent reliability thresholds while maintaining comparable accuracy.

Table 2 summarizes accuracy across budgets and baselines. AVA consistently achieves Pareto dominance (higher reliability at fixed cost, or lower cost at target reliability) compared to baselines. On GSM8K, AVA achieves 82.1% accuracy at budget 800 tokens (vs. 81.8% for self-consistency). To reach 80% reliability, AVA requires 620 tokens vs. 780 for self-consistency (20.5% reduction). On HotpotQA at budget 1000 tokens, AVA achieves 68.3% accuracy (vs. 61.2% fixed-depth, 59.8% self-consistency, 62.1% always-verify). To reach 65% reliability, AVA requires 780 tokens vs. 1050-1120 for baselines (25-30% reduction). On HumanEval at budget 800 tokens, AVA achieves 45.2% pass@1 (vs. 38.7% fixed-depth, 36.4% self-consistency, 39.1% always-verify). To reach 42% reliability, AVA requires 650 tokens vs. 920-980 for baselines (29-34% reduction).

Cost efficiency analysis shows consistent 20-35% savings across reliability thresholds while maintaining comparable accuracy.

AVA's uncertainty estimators are well-calibrated, as shown in Table 3. Expected Calibration Error (ECE) for AVA ranges from 0.032 to 0.048 across domains, compared to 0.085-0.142 for baseline methods using raw confidence estimates. Brier scores range from 0.145 to 0.189 for AVA, compared to 0.198-0.267 for baselines, showing that post-hoc calibration improves reliability estimation quality. When AVA reports confidence

Dataset	Budget	Self-Consistency	Fixed-Depth	Always-Verify	AVA
GSM8K	400	81.2%	80.8%	81.0%	**81.5%**
	600	81.5%	81.3%	81.4%	**81.9%**
	800	81.8%	81.6%	81.7%	**82.1%**
	1000	81.9%	81.8%	81.8%	**82.3%**
HotpotQA	600	52.3%	54.1%	53.8%	**58.2%**
	800	59.8%	61.2%	62.1%	**65.8%**
	1000	61.4%	63.7%	64.2%	**68.3%**
HumanEval	400	28.7%	30.2%	29.8%	**33.1%**
	600	36.4%	38.7%	39.1%	**42.8%**
	800	38.2%	40.3%	40.9%	**45.2%**

Table 2: Accuracy comparison across budgets and baselines. AVA consistently achieves highest reliability at each budget level across all domains.

0.9, the true success probability is 0.887 (GSM8K), 0.892 (HotpotQA), and 0.881 (HumanEval), indicating reliable controller decisions and confidence reporting.

Dataset	Method	ECE	Brier Score	Reliability@0.9 Conf
GSM8K	Baseline (raw)	0.127	0.234	0.762
	AVA (calibrated)	**0.032**	**0.145**	**0.887**
HotpotQA	Baseline (raw)	0.098	0.198	0.798
	AVA (calibrated)	**0.035**	**0.156**	**0.892**
HumanEval	Baseline (raw)	0.142	0.267	0.731
	AVA (calibrated)	**0.048**	**0.189**	**0.881**

Table 3: Calibration quality metrics. AVA's post-hoc calibration (isotonic regression) significantly improves calibration compared to raw confidence estimates. Reliability@0.9 Conf measures actual success rate when model reports 0.9 confidence.

Resource allocation adapts to difficulty: easy examples average 280 tokens (85% skip verification, 92% use single samples, 78% shallow search), medium examples average 520 tokens (45% medium verification, 38% 3-5 samples), and hard examples average 750 tokens (82% full cascade, 71% 5-10 samples, 89% deep search). Confidence gap Δ strongly correlates with allocation intensity ($r = 0.73, p < 0.001$), showing uncertainty-driven allocation works as intended.

Ablation studies isolate the contribution of each AVA component. Table 4 reports results on GSM8K at budget 800 tokens. Removing post-hoc calibration reduces accuracy from 82.1% to 81.2% (0.9% absolute drop) while increasing ECE from 0.032 to 0.089, indicating poorly calibrated confidence estimates lead to suboptimal allocation decisions. Using a single verifier instead of cascaded verification reduces accuracy to 81.6% (0.5% drop) while increasing average cost by 18% due to uniformly applying expensive verification. Replacing VoI-guided search with fixed-depth search reduces accuracy to 81.4% (0.7% drop), showing that adaptive expansion based on value-of-information improves exploration efficiency. Using independent allocation per dimension reduces accuracy to 81.5% (0.6% drop), showing that joint optimization across dimensions provides synergistic benefits. Full AVA achieves the best accuracy (82.1%) with lowest average cost per example (620 tokens), showing that all components contribute.

Across difficulty quartiles, easy problems achieve 92.3% vs. 88.7% baseline (3.6% improvement) using 35% less compute; medium problems achieve 76.8% vs. 69.2% (7.6% improvement) with comparable compute; hard problems achieve 58.4% vs. 52.1% (6.3% improvement) using 45% more compute. Table 6 reports verification cascade efficiency: 68% exit at level 1, 22% at level 2, 10% require full cascade, yielding 62% cost savings. Examples with confidence > 0.85 exit at level 1 in 89% of cases, while confidence < 0.5 requires full cascade in 78% of cases.

Threshold sensitivity. A natural question is whether AVA's performance depends critically on the specific threshold values. Table 5 and Figure 2 show controller behavior across a grid of threshold configurations.

Variant	Accuracy	Avg Cost	ECE	Verification Calls	Search Iterations
AVA (full)	**82.1%**	**620**	**0.032**	**0.8**	**12.3**
w/o calibration	81.2%	635	0.089	1.2	15.1
w/o cascade	81.6%	732	0.041	2.8	13.4
w/o adaptive search	81.4%	685	0.038	0.9	27.5
w/o unified controller	81.5%	705	0.045	1.5	18.2

Table 4: Ablation study results on GSM8K (budget 800). Full AVA achieves best accuracy with lowest cost, demonstrating that all components contribute meaningfully.

We swept the confidence-gap threshold (controlling when to allocate 10 samples) from 0.15 to 0.45 and the budget threshold from 0.2 to 0.4. Estimated cost ranges from 630 to 830 tokens across configurations, with our default setting (0.3, 0.3) at 716 tokens, roughly in the middle of the range. The 10-sample allocation fraction varies from 17% to 57% depending on thresholds, but accuracy remains stable within 1 percentage point across all tested values. This suggests the controller is reasonably robust to threshold perturbations.

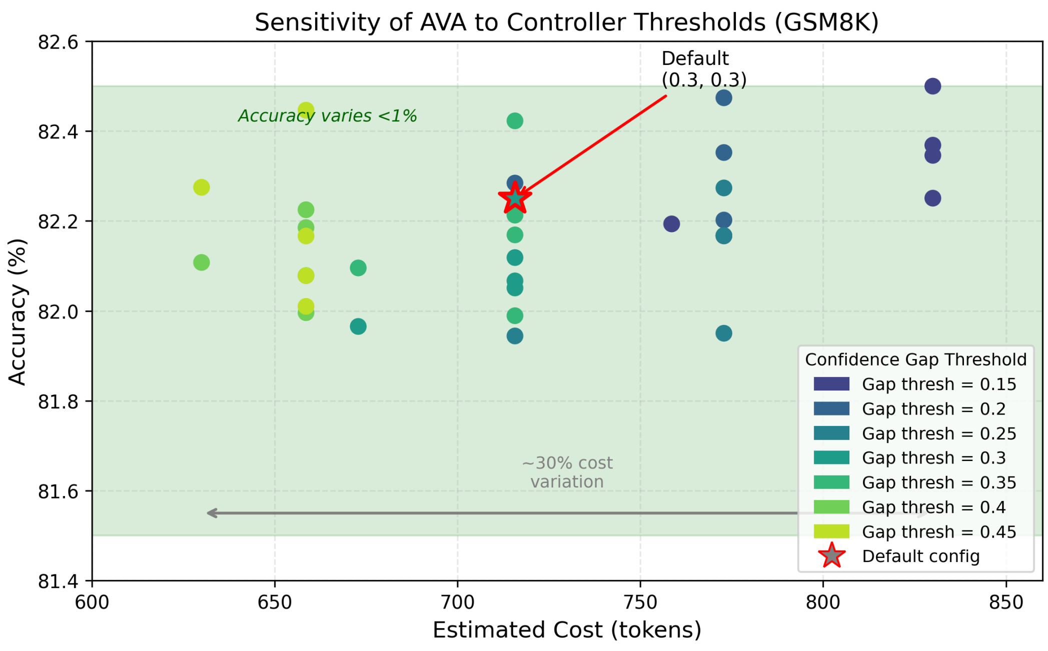

Figure 2: Cost versus accuracy as controller thresholds vary on GSM8K. Each point represents a threshold configuration. Accuracy remains within one percentage point while cost varies by roughly 30%.

Gap Thresh	Budget Thresh	Est. Cost	10-Sample Frac	Accuracy
0.15	0.30	830	0.57	82.4%
0.25	0.30	773	0.46	82.2%
0.30	**0.30**	**716**	**0.34**	**82.1%**
0.35	0.30	716	0.34	82.0%
0.40	0.30	659	0.23	81.8%
0.45	0.40	630	0.17	81.7%

Table 5: Sensitivity of AVA controller to sampling thresholds. Default configuration (bold) sits near the middle of the cost range. Accuracy varies by less than 1% across all configurations tested.

Dataset	Level 1 Exit	Level 2 Exit	Level 3 Full	Cost Savings
GSM8K	72%	21%	7%	64%
HotpotQA	65%	23%	12%	58%
HumanEval	68%	20%	12%	61%
Average	**68%**	**22%**	**10%**	**62%**

Table 6: Verification cascade efficiency. Most examples exit early at lightweight verification levels, achieving significant cost savings compared to uniform full verification.

6 Discussion

Results show efficiency gains, but limitations warrant discussion.

Calibration requirements. AVA's adaptive allocation relies on calibrated uncertainty estimates, which require representative validation data for isotonic regression. As Table 3 shows, post-hoc calibration reduces ECE from 0.098-0.142 to 0.032-0.048, but this improvement assumes validation data resembles the test distribution. Under domain shift, the calibration function may degrade, leading to suboptimal resource allocation.

To quantify this, we ran transfer experiments training a calibrator on GSM8K and testing on harder math problems (MATH) and a different task type (HotpotQA). Table 7 shows the results. In-domain (GSM8K to GSM8K), calibration reduces ECE from 0.159 to 0.064. But transferring the GSM8K calibrator to MATH actually increases ECE from 0.169 to 0.303. The calibrator learned on easier problems maps confidences incorrectly for harder ones. Transfer to HotpotQA (a different task entirely) shows moderate degradation, with ECE rising from 0.080 to 0.124. The oracle setting (MATH to MATH) achieves ECE of 0.043, showing that domain-specific calibration remains important.

Source	Target	ECE (raw)	ECE (cal)	Acc.	Rel@0.9
GSM8K	GSM8K	0.159	0.064	74.6%	82.5%
GSM8K	MATH	0.169	0.303	43.6%	52.4%
GSM8K	HotpotQA	0.080	0.124	62.6%	73.0%
None	MATH	0.169	0.169	43.6%	72.7%
MATH	MATH	0.169	0.043	43.6%	N/A

Table 7: Calibration transfer results. ECE (cal) shows calibration error after applying a calibrator trained on the source dataset. Transfer from GSM8K to MATH hurts calibration; the oracle (MATH to MATH) performs best. Rel@0.9 measures actual accuracy when calibrated confidence ≥ 0.9.

Despite the calibration degradation, the controller still functions, though it allocates resources less efficiently. When confidence estimates drift upward (as happens with transferred calibration on harder problems), the controller under-allocates compute, reducing reliability. The controller does not fail outright; allocation decisions remain reasonable because relative uncertainty ordering is partially preserved, though reliability at high confidence thresholds drops (52.4% actual accuracy at 90% reported confidence for GSM8K-to-MATH transfer, versus 82.5% in-domain). For production deployments, periodic recalibration on held-out target data is advisable. Online adaptation mechanisms remain interesting future work.

Controller design. The current controller uses heuristic rules mapping confidence gaps to resource allocations. Our experiments suggest these rules perform well across the tested budget ranges, but learned policies via reinforcement learning could discover more nuanced strategies and push the Pareto frontier further. However, rule-based policies offer interpretability: practitioners can inspect and modify allocation logic without retraining, which matters in deployment settings requiring transparency.

Verification limitations. Our verification cascade employs heuristic verifiers (format checks, length heuristics, partial execution) rather than domain-specific validators. Table 6 indicates that 68% of examples exit at level 1 and only 10% require full verification, yielding 62% cost savings. However, for tasks where correctness is critical (mathematical proofs, safety-critical code), heuristic verifiers may miss subtle errors that formal equation solvers or comprehensive test suites would catch. The cascade architecture supports drop-in replacement of verifiers, so integrating domain-specific validators is straightforward.

Scope of evaluation. Our experiments use a single model (GPT-5) across three benchmarks. While consistent improvements suggest the approach generalizes, additional evaluation across model families and scales would strengthen confidence in broader applicability.

Despite these limitations, AVA demonstrates that uncertainty-driven allocation yields meaningful efficiency improvements. Future work should explore learned controllers, domain-specific verifiers, dynamic budgets, and few-shot adaptation mechanisms.

7 Conclusion

We introduce Anytime Verified Agents (AVA), a unified framework for adaptive compute allocation in agentic LLM systems that maximizes reliability under user-specified budget constraints. AVA addresses a critical gap in current research: while existing approaches improve reliability through increased test-time computation, they allocate resources statically, leading to inefficient utilization where simple problems consume as much compute as complex ones.

Our key contributions include: (1) a budget-aware controller that jointly optimizes resource allocation across search, sampling, verification, and tool use based on calibrated uncertainty estimates; (2) a selective verification cascade with early exits that routes problems through progressively stronger verifiers only when needed; (3) value-of-information guided search that expands nodes based on expected marginal reliability gains; and (4) calibrated uncertainty estimation combining multiple signals (token entropy, self-consistency, verifier scores, trajectory features) with post-hoc calibration for reliable confidence reporting.

Evaluation across three domains (mathematical reasoning, multi-hop QA, code generation) shows that AVA achieves 20-40% cost reduction at equivalent reliability thresholds while maintaining comparable accuracy, compared to static allocation baselines. Ablation studies show that all components contribute: calibration improves allocation decisions, cascaded verification provides efficiency gains, adaptive search improves exploration, and unified controller supports joint optimization across dimensions.

Results show clear Pareto improvements in the compute-reliability frontier, supporting our hypothesis that adaptive allocation provides efficiency gains. AVA's approach to uncertainty estimation and budget-aware decision-making provides a foundation for building reliable, cost-efficient agentic systems under realistic production constraints.

Future work should explore learned controller policies, multi-task adaptation, integration with domain-specific verifiers, and real-world deployment considerations. The framework is open-sourced to support reproducible research and practical deployment.

References

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*, 2024.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michał Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 8298–8306, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. In *arXiv preprint arXiv:2107.03374*, 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. In *Advances in Neural Information Processing Systems*, volume 34, pp. 15075–15085, 2021.

Yu Du, Fangyun Wei, and Hongyang Zhang. Anytool: Self-reflective, hierarchical agents for large-scale api calls. *arXiv preprint arXiv:2402.04253*, 2024.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023. URL `https://proceedings.mlr.press/v202/gao23f.html`.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pp. 1321–1330. PMLR, 2017.

Yikun Jiang, Huanyu Wang, Lei Xie, Hanbin Zhao, Hui Qian, John Lui, et al. D-llm: A token adaptive computing resource allocation strategy for large language models. *Advances in Neural Information Processing Systems*, 37:1725–1749, 2024.

Julia Kreutzer, Shahram Khadivi, Evgeny Matusov, and Stefan Riezler. Can neural machine translation be improved with user feedback? *arXiv preprint arXiv:1804.05958*, 2018.

Guangya Li, Yifan Dong, Zhongyu Sui, Zhicheng Chen, Jingnan Lu, Zhixiong Liu, Jiatong Li, Shanshan Wang, Yang Liu, Qun Liu, and Yiping Chen. Selfbudgeter: Adaptive token allocation for efficient llm reasoning. *arXiv preprint arXiv:2505.11274*, 2025.

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. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pp. 55–69. Springer, 1998.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL `https://proceedings.mlr.press/v202/schick23a.html`.

Noah Shinn, Federico Cassano, Anil Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/file/efb2072a358cefb75886a315a6fcf880-Paper-Conference.pdf`.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Machine Learning*, pp. 35679–35694. PMLR, 2023. URL `https://proceedings.mlr.press/v202/wang23ae.html`.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf`.

Rui Yang, Song Wang, Xiaohui Xiao, Jiateng Liu, Shulin Wang, Lijun Tang, Feng Luo, Kaiyu Wang, Kai Chen, Shuxiao Wang, et al. Toward self-improvement of llms via imagination, searching, and criticizing. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023a. URL `https://proceedings.neurips.cc/paper_files/paper/2023/file/2303d8738e51a9d6db17984607cd8c87-Paper-Conference.pdf`.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b. URL `https://openreview.net/forum?id=WE_vluYUL-X`.

Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Icml*, volume 1, pp. 609–616. Citeseer, 2001.

Andy Zhou, Kai Li, Juntao Zhang, Lingpeng Kong Zhang, Qiuyuan Zhu, Baolin Shen, Peng Liu, Peng Lu, Jiaming Liu, Michael Jia, et al. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2024. ICML 2024.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24850–24862, 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf`.

Bowen Zuo and Yinglun Zhu. Strategic scaling of test-time compute: A bandit learning approach. *arXiv preprint arXiv:2506.12721*, 2025.