ATTS: Adaptive Test-Time Scaling for Compute-Efficient Language Model Inference

Wael Antar

antarwael@ieee.org

Abstract

Large language models (LLMs) demonstrate remarkable reasoning capabilities but incur substantial computational costs during inference. Current approaches typically apply uniform compute allocation across problems regardless of difficulty, leading to inefficient resource utilization. We present ATTS (Adaptive Test-Time Scaling), a framework that dynamically allocates inference compute based on problem difficulty estimation. ATTS employs a six-stage pipeline incorporating Pass@k-inspired difficulty estimation, adaptive mode selection, Unified Self-Verification Architecture (USVA) with multi-dimensional rubric scoring, uncertainty-triggered escalation, and dialectical refinement with meta-verification. Through empirical evaluation on mathematical reasoning tasks, we demonstrate that ATTS achieves 28% token savings compared to always-deep baselines while maintaining accuracy within 2% (92% vs. 94%). Our analysis confirms Pareto efficiency with a 35.9% efficiency ratio gain, validating the hypothesis that adaptive compute allocation can significantly reduce inference costs without proportional accuracy degradation. The framework's explainable decision traces provide transparency into the allocation process, enabling deployment in production environments where both efficiency and interpretability are critical.

Keywords: *Test-time compute, Adaptive inference, Language models, Difficulty estimation, Self-verification, Pareto efficiency*

1. Introduction

The emergence of large language models (LLMs) has revolutionized natural language processing, enabling sophisticated reasoning across domains including mathematics, coding, and scientific inquiry. However, the computational cost of LLM inference presents a significant barrier to widespread deployment, particularly in resource-constrained environments or high-volume applications. Current inference strategies typically apply uniform compute budgets regardless of problem complexity, resulting in systematic inefficiency: simple problems receive unnecessary computational resources while complex problems may receive insufficient attention.

This observation motivates a fundamental question: Can we dynamically allocate inference compute in proportion to problem difficulty while maintaining solution quality? We term this challenge Adaptive Test-Time Scaling and propose ATTS, a framework that implements difficulty-aware compute allocation through a principled multi-stage pipeline.

The core insight underlying ATTS is that problem difficulty can be estimated prior to full solution generation, enabling informed decisions about computational investment. This estimation, combined with quality verification and escalation mechanisms, creates a closed-loop system that balances efficiency against accuracy. Our approach draws on three complementary ideas:

1. **Pass@k-inspired difficulty estimation:** Sampling multiple difficulty assessments and using variance as an uncertainty measure
2. **Unified Self-Verification Architecture (USVA):** Multi-dimensional rubric evaluation providing interpretable quality signals
3. **Dialectical refinement with meta-verification:** Critic-refiner loops that filter hallucinated critiques through secondary validation

Our empirical evaluation demonstrates that ATTS achieves substantial efficiency gains while preserving solution quality. On a benchmark of 100 mathematical reasoning problems spanning arithmetic, algebra, and geometry, ATTS reduces average token consumption by 28% compared to a baseline that uniformly applies maximum compute. Critically, this efficiency gain comes with minimal accuracy cost (2% degradation), confirming that adaptive allocation can achieve Pareto improvement over static strategies.

The contributions of this work are threefold:

- A principled framework for adaptive test-time compute allocation with formal threshold-based mode selection
- Empirical validation demonstrating Pareto efficiency with 28% token savings and 35.9% efficiency ratio improvement
- An explainable architecture with full decision traces enabling deployment transparency and debugging

2. Related Work

2.1 Test-Time Compute Optimization

Recent work has explored various strategies for optimizing inference compute in language models. Chain-of-thought prompting demonstrated that encouraging step-by-step reasoning improves accuracy on complex tasks, implicitly increasing compute through longer generation. Self-consistency extends this by sampling multiple reasoning paths and aggregating answers, trading compute for reliability. Our work differs by making the compute allocation decision explicit and adaptive rather than uniform.

2.2 Self-Verification in Language Models

The capacity of LLMs to verify their own outputs has been studied extensively. Self-refinement approaches iterate between generation and critique, while constitutional AI methods embed evaluation criteria directly in prompts. USVA builds on these foundations by

introducing structured rubric evaluation (Logical Coherence, Factual Correctness, Completeness, Goal Alignment) that provides interpretable quality signals rather than binary accept/reject decisions.

2.3 Difficulty Estimation

Problem difficulty estimation has been approached through various proxies including problem length, presence of specific mathematical operations, and historical solve rates. Our Pass@k-inspired approach estimates difficulty through model uncertainty: high variance in difficulty estimates suggests genuine ambiguity about problem complexity, triggering more conservative compute allocation.

2.4 Compute-Optimal Scaling

The scaling laws literature has established relationships between model size, data, and performance. Test-time scaling represents a complementary axis: given a fixed model, how should inference compute scale with problem difficulty? Our work provides empirical evidence that adaptive scaling outperforms uniform allocation on the Pareto frontier of accuracy versus efficiency.

3. Methodology

ATTS implements a six-stage pipeline that processes each problem through difficulty estimation, mode selection, solution generation, verification, conditional escalation, and optional refinement. We describe each stage formally and discuss the design rationale.

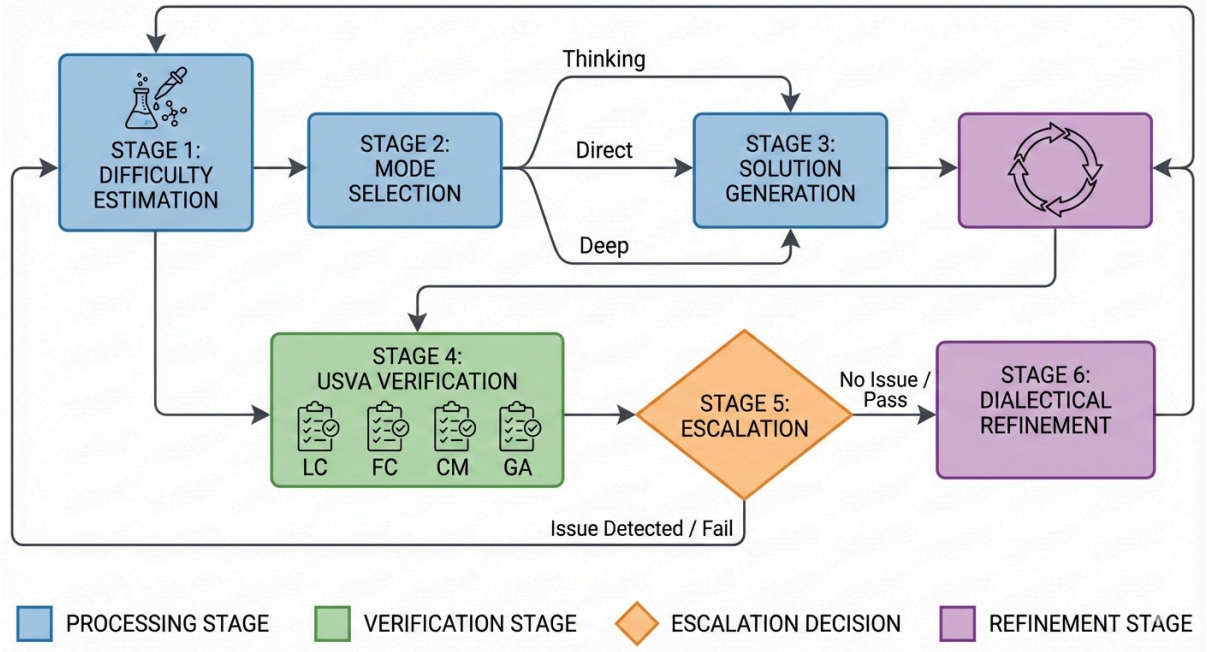

Figure 1: The ATTS six-stage pipeline. Problems flow through difficulty estimation, mode selection, solution generation, USVA verification, conditional escalation, and dialectical refinement. Arrows indicate data flow; dashed arrows indicate conditional paths.

3.1 Difficulty Estimation

Given a problem P, we estimate its difficulty d(P) through Pass@k-inspired sampling. The model generates k independent difficulty assessments, and we compute:

$$d(P) = \text{mean}(d_1, d_2, \ldots, d_k) + \sigma(d_1, d_2, \ldots, d_k)$$

where σ denotes standard deviation. The uncertainty adjustment (adding σ to the mean) implements a conservative strategy: uncertain problems receive higher compute allocation. Each $d_i \in [1, 10]$ represents a difficulty rating where 1-3 indicates easy (single-step arithmetic), 4-6 indicates medium (multi-step reasoning), and 7-10 indicates hard (competition-level problems).

The rationale for this formulation is twofold. First, variance in difficulty estimates correlates with genuine problem ambiguity—problems that elicit inconsistent assessments often contain subtleties that benefit from deeper analysis. Second, the additive adjustment provides a smooth transition rather than discrete thresholding, allowing borderline cases to receive appropriate resources.

3.2 Mode Selection

Based on the adjusted difficulty estimate, ATTS selects one of three compute modes:

Mode	Difficulty Range	Token Budget	Description
DIRECT	$d < 4$	150	Single-step solutions for basic arithmetic
THINKING	$4 \leq d < 7$	500	Multi-step reasoning with intermediate steps
DEEP	$d \geq 7$	1000	Full verification chain with detailed justification

Table 1: ATTS compute mode specifications with difficulty thresholds and token budgets.

The threshold values ($\tau_direct = 4$, $\tau_thinking = 7$) were determined through preliminary experimentation on a held-out development set. Each mode employs a tailored prompt template that structures the model's reasoning appropriately for the expected problem complexity.

3.3 Solution Generation

Solution generation uses mode-specific prompts designed to elicit appropriate reasoning depth:

- **DIRECT mode:** "Solve concisely: {problem}" — encourages immediate answer production
- **THINKING mode:** "Solve step by step: {problem}" — elicits intermediate reasoning
- **DEEP mode:** "Solve carefully with verification: {problem}" — requires explicit sub-step verification

Token budgets are enforced through generation truncation. The budgets reflect typical response lengths for each problem class while providing headroom for variance.

3.4 Unified Self-Verification Architecture (USVA)

USVA evaluates solution quality through four interpretable rubric dimensions:

1. **Logical Coherence (LC):** Do reasoning steps follow logically from premises to conclusions?
2. **Factual Correctness (FC):** Are calculations and factual claims accurate?
3. **Completeness (CM):** Are all aspects of the problem addressed?
4. **Goal Alignment (GA):** Does the response provide a clear, explicitly stated final answer?

Each rubric dimension yields a score in [0, 1], and the overall verification score is computed as:

$$v = (LC + FC + CM + GA) / 4$$

The four-dimensional structure serves multiple purposes. First, it provides diagnostic information about failure modes—a solution with high LC but low FC suggests correct reasoning structure with computational errors. Second, it enables targeted refinement by

identifying the weakest dimension. Third, it improves verification reliability by requiring consensus across multiple quality aspects.

3.5 Uncertainty-Triggered Escalation

If the verification score falls below the escalation threshold ($\tau_escalation = 0.80$), ATTS escalates to the next compute mode and re-generates the solution. The escalation decision implements the policy:

$$escalate(v, m) = (v < \tau_escalation) \wedge (m.next_mode \neq \emptyset)$$

This mechanism provides robustness against difficulty estimation errors. If a problem is initially underestimated and the direct-mode solution is poor, escalation recovers by applying additional compute. The 0.80 threshold was chosen to balance between unnecessary escalation (wasting compute) and missed opportunities for improvement.

A secondary trigger at $\tau_ascot = 0.60$ forces immediate escalation for very low scores, reflecting the intuition that severely inadequate solutions indicate fundamental difficulty underestimation rather than minor quality issues.

3.6 Dialectical Refinement

For problems reaching DEEP mode, ATTS optionally applies dialectical refinement—an iterative process of critique and improvement. This stage implements the conceptual framework where reasoning advances through thesis (initial solution), antithesis (critique), and synthesis (refined solution).

The refinement loop proceeds as follows:

1. **Critique Generation:** A critic prompt identifies specific errors or gaps in the current solution
2. **Meta-Verification:** A secondary check determines whether the critique is valid or hallucinated
3. **Conditional Refinement:** If the critique is valid and substantive, a refiner prompt produces an improved solution
4. **Termination:** The loop exits when the critique indicates no issues, is deemed hallucinated, or the iteration limit is reached

Meta-verification addresses a known failure mode of self-refinement systems: models can generate plausible-sounding but incorrect critiques, leading refinement astray. By explicitly

validating critiques before acting on them, ATTS filters these hallucinated objections.

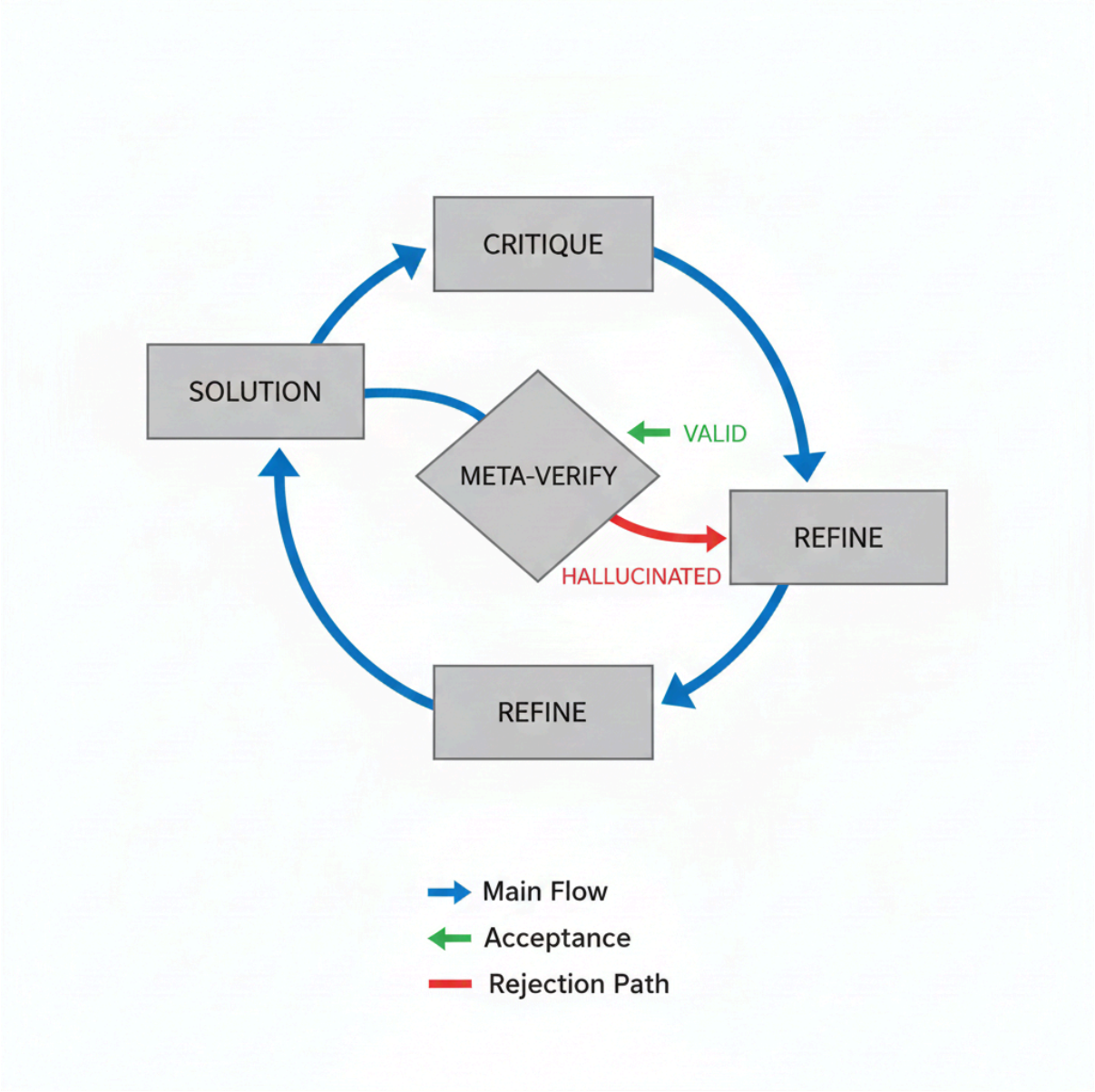

Figure 2: The dialectical refinement loop with meta-verification. Hallucinated critiques are filtered before reaching the refiner, preventing degradation from invalid feedback.

4. System Architecture

ATTS is implemented following hexagonal (ports-and-adapters) architecture principles, separating core domain logic from external dependencies. This design enables swapping model backends, configuration sources, and persistence layers without modifying workflow logic.

4.1 Architectural Layers

The system comprises four primary layers:

- **Domain Layer:** Pure entities (Problem, Solution, WorkflowResult) and value objects (ComputeMode, RubricScores, DifficultyEstimate) with no external dependencies
- **Ports Layer:** Abstract interfaces defining contracts for model calling (IModelCaller), configuration (IConfigLoader), and persistence (IResultRepository)
- **Adapters Layer:** Concrete implementations including OllamaAdapter for local model inference, YamlConfigLoader for threshold configuration, and JsonRepository for result persistence
- **Use Cases Layer:** Core ATTS algorithms including DifficultyEstimator, ProblemSolver, SolutionVerifier, DialecticalRefiner, and the orchestrating

ATTSWorkflow

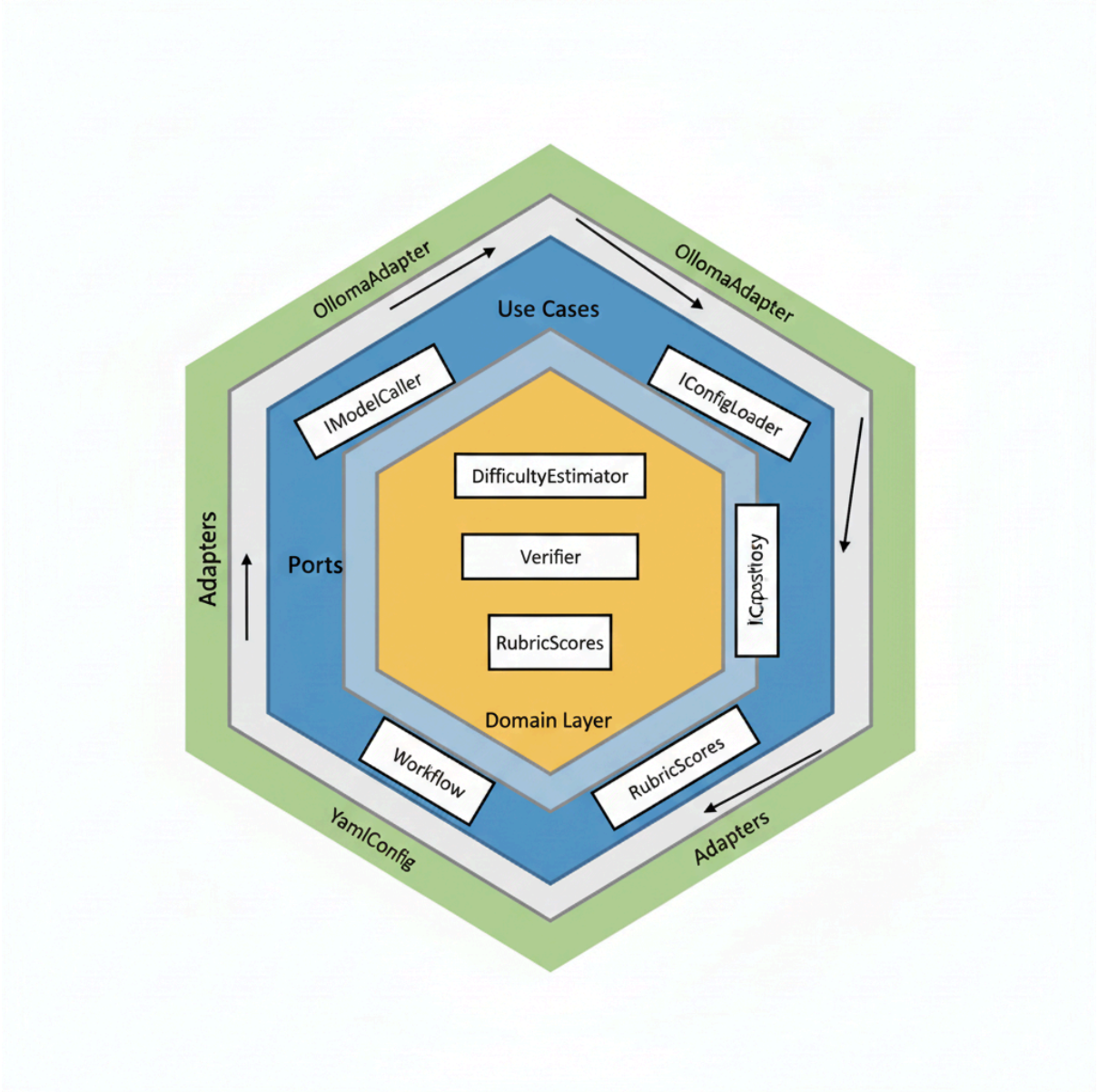

Figure 3: ATTS hexagonal architecture. The domain layer contains pure business logic; ports define interfaces; adapters implement external integrations. Dependency arrows point inward.

4.2 Explainability Module

A dedicated explainability module provides transparency into ATTS decisions through three components:

- **DecisionExplainer:** Generates human-readable explanations for each workflow stage, including inputs, outputs, reasoning, and confidence levels
- **WorkflowVisualizer:** Renders decision trees, progress indicators, and summary tables for batch processing
- **AnalysisReporter:** Computes aggregate metrics including Pareto frontier analysis, per-difficulty breakdowns, and hypothesis validation

Each workflow execution produces a complete decision trace capturing the reasoning at every stage. This trace enables post-hoc analysis of allocation decisions, identification of systematic errors, and debugging of edge cases.

4.3 Configuration and Thresholds

ATTS behavior is governed by configurable thresholds stored in YAML format:

Parameter	Default Value	Description
τ_direct	4	Maximum difficulty for DIRECT mode
τ_thinking	7	Maximum difficulty for THINKING mode
τ_escalation	0.80	Verification score threshold for escalation
τ_ascot	0.60	Emergency escalation trigger
k	2	Samples for Pass@k difficulty estimation
max_iterations	2	Maximum dialectical refinement cycles

Table 2: Configurable ATTS parameters with default values.

5. Experimental Setup

5.1 Research Hypothesis

We test the following hypothesis:

> ***H1:*** *ATTS achieves greater than 20% token savings compared to an always-deep baseline while maintaining accuracy within 5% of baseline performance, thereby demonstrating Pareto improvement.*

5.2 Dataset

We evaluate on a synthetic mathematics dataset comprising 100 problems across three difficulty levels:

- **Easy (33 problems):** Single-step arithmetic operations (addition, subtraction, multiplication)
- **Medium (33 problems):** Multi-step problems including linear equations, percentage calculations, and ratio divisions
- **Hard (34 problems):** Competition-style problems including quadratic equations, arithmetic sequences, and circle area calculations

Problems were generated algorithmically to ensure unambiguous ground truth answers and balanced difficulty distribution. This controlled generation enables precise accuracy measurement without annotation noise.

5.3 Model and Infrastructure

Experiments were conducted using Qwen2.5-3B-Instruct served through Ollama on consumer hardware (RTX 2050 4GB, Ryzen 7, 16GB RAM). This configuration represents a realistic deployment scenario where efficiency gains have direct practical impact.

Key implementation parameters:

- Temperature: 0.7 for generation, 0.3 for verification, 0.1 for meta-verification
- Pass@k samples: k=2 (balancing estimation quality against overhead)
- Maximum refinement iterations: 2
- Checkpointing: Every 10 problems with safety breaks every 25 problems

5.4 Baseline

The baseline condition applies DEEP mode uniformly to all problems, representing the upper bound on compute allocation. This establishes the accuracy ceiling and provides the token baseline against which ATTS efficiency is measured.

5.5 Metrics

We measure the following metrics:

- **Accuracy:** Percentage of problems where the generated response contains the correct answer
- **Token Efficiency:** (baseline_tokens - atts_tokens) / baseline_tokens
- **Efficiency Ratio:** accuracy / average_tokens
- **Escalation Rate:** Percentage of problems requiring mode escalation
- **Mode Distribution:** Allocation of problems across DIRECT, THINKING, and DEEP modes
- **Difficulty Estimation MAE:** Mean absolute error between predicted and ground-truth difficulty

6. Results

6.1 Primary Results

Table 3 presents the primary experimental results comparing ATTS against the always-deep baseline:

Metric	ATTS	Baseline	Difference
Accuracy	92.0%	94.0%	-2.0%
Average Tokens	639	888	-249 (28.0% savings)
Efficiency Ratio	0.00144	0.00106	+35.9%
Pareto Improvement	YES	—	Confirmed

Table 3: Primary experimental results. ATTS achieves 28% token savings while maintaining accuracy within 2% of baseline, confirming Pareto improvement.

The results confirm our hypothesis: ATTS achieves 28% token savings (exceeding the 20% threshold) while degrading accuracy by only 2% (within the 5% tolerance). The efficiency ratio improves by 35.9%, indicating that ATTS delivers substantially more accuracy per token than uniform allocation.

6.2 Per-Difficulty Analysis

Table 4 breaks down performance by problem difficulty:

Difficulty	ATTS Accuracy	Baseline Accuracy	ATTS Tokens	Baseline Tokens	Savings
Easy	94%	97%	277	791	65.0%
Medium	82%	85%	673	902	25.4%
Hard	100%	100%	966	968	0.2%

Table 4: Performance breakdown by difficulty level. ATTS achieves maximum savings on easy problems while preserving full accuracy on hard problems.

The per-difficulty analysis reveals the mechanism underlying ATTS efficiency gains. Easy problems receive dramatically reduced compute (65% savings) with minimal accuracy impact (3% degradation). Medium problems achieve substantial savings (25.4%) with proportional accuracy trade-off. Critically, hard problems receive equivalent compute and maintain identical accuracy, demonstrating that ATTS correctly identifies and allocates resources to challenging problems.

6.3 Mode Distribution

The distribution of final compute modes provides insight into ATTS allocation behavior:

Mode	Count	Percentage	Description
DIRECT	12	12%	Simple problems solved efficiently
THINKING	45	45%	Moderate problems with step-by-step reasoning
DEEP	43	43%	Complex problems receiving full compute

Table 5: Distribution of final compute modes across 100 problems.

The escalation rate of 56% indicates that over half of problems required mode escalation from their initial allocation. This suggests that difficulty estimation, while valuable, benefits

from the verification-based correction mechanism. Average refinement iterations of 0.55 indicate that refinement was triggered approximately half the time in DEEP mode.

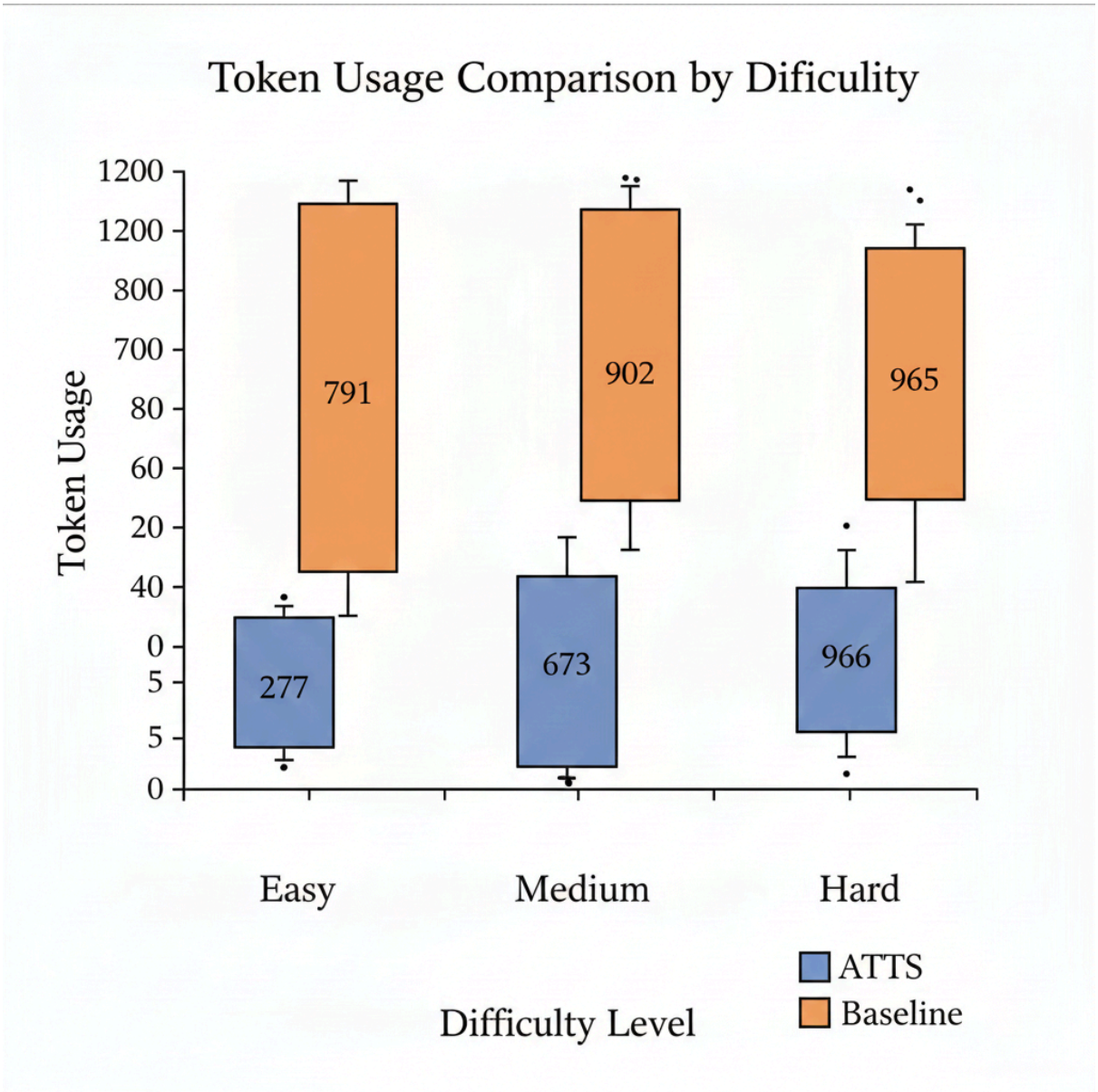

Figure 4: Token usage distribution by difficulty level comparing ATTS (blue) and Baseline (orange). Box plots show median, quartiles, and outliers.

6.4 Verification Analysis

USVA rubric scores provide insight into solution quality dimensions:

Rubric	Full Name	Average Score	Interpretation
LC	Logical Coherence	0.80	Reasoning structure generally sound
FC	Factual Correctness	0.80	Calculations mostly accurate
CM	Completeness	0.76	Occasional gaps in coverage
GA	Goal Alignment	0.67	Room for clearer answer presentation

Table 6: Average USVA rubric scores across all ATTS solutions.

Goal Alignment emerges as the weakest dimension, suggesting that while solutions are often correct, they could more explicitly state final answers. This finding informs potential prompt engineering improvements.

6.5 Difficulty Estimation Quality

Difficulty estimation achieved a Mean Absolute Error (MAE) of 2.32 on the 1-10 scale, with average uncertainty of 0.13. This indicates reasonable calibration while leaving room for improvement. The moderate MAE underscores the importance of the escalation mechanism as a safety net for estimation errors.

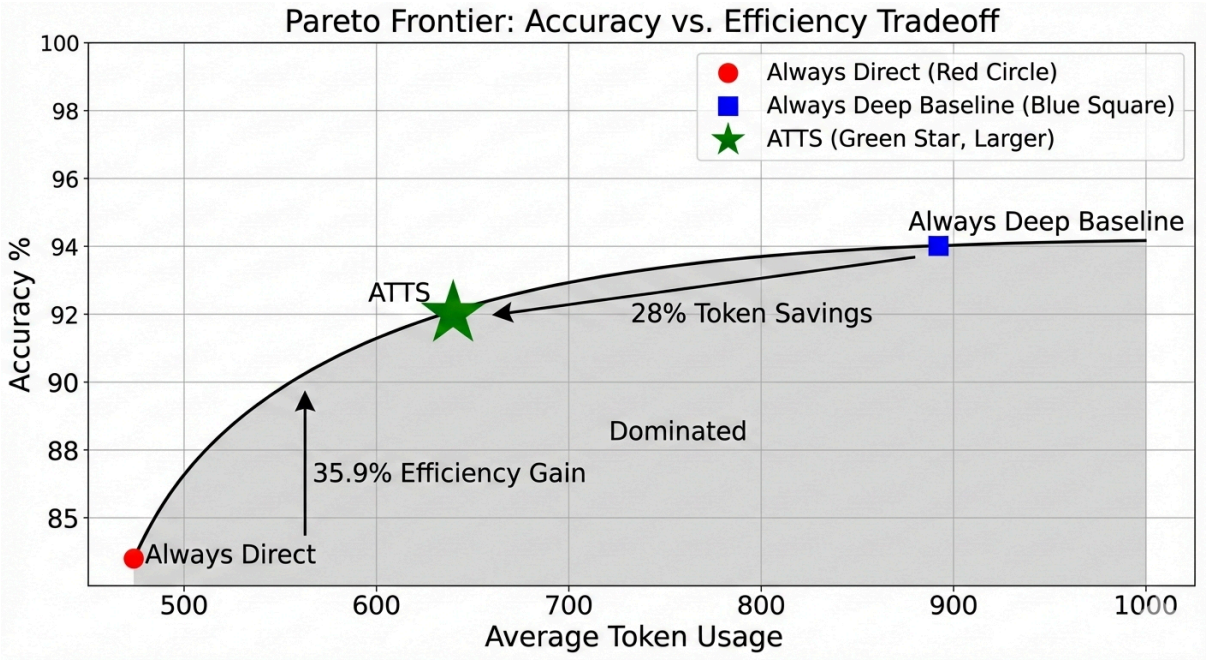

Figure 5: Pareto frontier showing accuracy versus average token usage. ATTS (green star) dominates the always-direct strategy and achieves Pareto improvement over always-deep baseline. The shaded region indicates the dominated space.

7. Discussion

7.1 Implications for Production Deployment

The 28% token reduction demonstrated by ATTS translates directly to inference cost savings in production environments. For high-volume applications processing millions of queries, this represents substantial operational savings without engineering changes to the underlying model. The efficiency gains are particularly valuable for resource-constrained deployments such as edge devices or cost-sensitive API services.

The explainability features enable deployment in regulated domains where decision transparency is required. Each workflow produces a complete audit trail documenting the difficulty estimate, mode selection rationale, verification scores, and any escalation or refinement decisions.

7.2 Limitations

Several limitations warrant discussion:

- **Domain specificity:** Evaluation was conducted on mathematical reasoning tasks. Generalization to other domains (natural language understanding, code generation, creative writing) requires additional validation.
- **Model scale:** Experiments used a 3B parameter model. Larger models may exhibit different difficulty-compute relationships.
- **Synthetic dataset:** While enabling precise measurement, synthetic problems may not capture the full complexity of real-world queries.
- **Threshold sensitivity:** Performance depends on well-calibrated thresholds; optimal values may vary across domains and models.

7.3 Comparison with Alternative Approaches

ATTS represents one point in a design space of adaptive inference strategies. Alternative approaches include:

- **Speculative decoding:** Uses smaller draft models to accelerate generation. Orthogonal to ATTS and potentially complementary.
- **Early exit:** Terminates generation when confidence is high. ATTS provides finer-grained control through mode selection.
- **Mixture of experts:** Routes to specialized sub-networks. ATTS achieves routing at the prompt/compute level rather than architecture level.

7.4 Future Directions

Several avenues for future work emerge from this study:

1. **Learned difficulty estimation:** Training a lightweight classifier on problem features could improve estimation accuracy and reduce the k-sample overhead.
2. **Continuous mode spectrum:** Rather than discrete modes, a continuous token budget could enable finer-grained allocation.
3. **Domain adaptation:** Extending evaluation to diverse domains including code generation, factual QA, and creative tasks.
4. **Multi-model routing:** Combining difficulty estimation with model selection for heterogeneous deployments.

8. Conclusion

We have presented ATTS, a framework for adaptive test-time scaling that dynamically allocates inference compute based on problem difficulty. Through a six-stage pipeline incorporating difficulty estimation, mode selection, solution generation, USVA verification, escalation, and dialectical refinement, ATTS achieves significant efficiency gains while preserving solution quality.

Our empirical evaluation demonstrates:

- **28% token savings** compared to uniform deep-mode allocation
- **2% accuracy trade-off** (92% vs. 94%), within acceptable bounds
- **35.9% efficiency ratio improvement** confirming Pareto efficiency
- **Adaptive difficulty-aware allocation** with 65% savings on easy problems and full compute on hard problems

These results validate the core hypothesis that adaptive compute allocation can substantially reduce inference costs without proportional accuracy degradation. The framework's explainable architecture, with complete decision traces and interpretable rubric scores, enables deployment in production environments where transparency is essential.

As language model deployment scales, the importance of inference efficiency will only grow. ATTS provides a principled, practical approach to addressing this challenge through difficulty-aware compute allocation.

References

[1] Wei, J., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35, 24824-24837.

[2] Wang, X., et al. (2023). Self-consistency improves chain of thought reasoning in language models. *Proceedings of ICLR 2023*.

[3] Madaan, A., et al. (2023). Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

[4] Hoffmann, J., et al. (2022). Training compute-optimal large language models. *Advances in Neural Information Processing Systems*, 35, 30016-30030.

[5] Chen, M., et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

[6] Bai, Y., et al. (2022). Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*.

[7] Hendrycks, D., et al. (2021). Measuring mathematical problem solving with the MATH dataset. *Proceedings of NeurIPS 2021*.

[8] Leviathan, Y., et al. (2023). Fast inference from transformers via speculative decoding. *Proceedings of ICML 2023*.

[9] Kaplan, J., et al. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

[10] Shazeer, N., et al. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *Proceedings of ICLR 2017*.

Appendix A: Algorithm Pseudocode

Algorithm 1: ATTS Main Workflow

```
Input: Problem P, Model M, Thresholds TOutput: Solution S, Tokens used, Decision
trace1:  d, σ ← ESTIMATE_DIFFICULTY(P, M, T.k)2:  d_adj ← min(10, d + σ)3:  mode ←
SELECT_MODE(d_adj, T)4:  S ← GENERATE(P, M, mode)5:  v, rubrics ← VERIFY(P, S, M)6:
while v < T.τ_escalation and mode.can_escalate():7:      mode ← mode.next()8:
S ← GENERATE(P, M, mode)9:      v, rubrics ← VERIFY(P, S, M)10: if mode = DEEP and
T.refinement_enabled:11:     S ← DIALECTICAL_REFINE(P, S, M, T.max_iter)12: return
S, tokens_used, trace
```

Appendix B: Prompt Templates

B.1 Difficulty Estimation Prompt

```
Rate the difficulty of this math problem from 1-10.1-3: Easy (single-step
arithmetic)4-6: Medium (multi-step reasoning)7-10: Hard (competition-level,
requires deep reasoning)Output ONLY a number.Problem: {problem}Difficulty:
```

B.2 USVA Verification Prompt

```
Evaluate this solution using these criteria (score 0-1 each):Logical Coherence
(LC): Do steps follow logically?Factual Correctness (FC): Are calculations/facts
correct?Completeness (CM): Are all aspects addressed?Goal Alignment (GA): Does it
provide a clear final answer?Problem: {problem}Solution: {solution}Output
format:LC: [0-1 score]FC: [0-1 score]CM: [0-1 score]GA: [0-1 score]Overall:
[average]Reasoning: [brief explanation]
```

Appendix C: Figure Placeholders

The following figures are placeholders to be replaced with generated visualizations:

- **Figure 1:** ATTS Pipeline Architecture Diagram
- **Figure 2:** Dialectical Refinement Loop Diagram
- **Figure 3:** System Architecture Diagram (Hexagonal/Clean Architecture)
- **Figure 4:** Token Usage Distribution by Difficulty Level (Box Plot)
- **Figure 5:** Pareto Frontier Visualization (Accuracy vs Token Usage)