ADAPTIVE UTILITY-RATIONAL ALLOCATION: THINKING ONLY WHEN IT MATTERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reasoning models can continue generating after an answer has stabilized, increasing inference cost without improving task performance. We introduce Adaptive Utility-Rational Allocation (AURA), a controller framework that compares a proxy for further improvement with the price of additional reasoning tokens. AURA combines a development-tuned certainty score, stopping safeguards, bounded auxiliary checks, and batch-level price feedback while leaving model weights fixed. On MATH-500, AURA achieves 93.2% accuracy with 646 tokens, versus 748 for Plan-and-Budget and 751 for an RL halter at the same accuracy point estimate. Paired intervals support these 13.6–14.0% token reductions. Score ablations on MATH-500 and TeleQuAD identify confidence as a consistent contributor to token efficiency, while a confidence-only score nearly reproduces the full MATH result and uses fewer tokens on TeleQuAD. In cumulative ablations across both tasks, safeguards and auxiliary checks are associated with higher observed answer quality. Batch-level price feedback reduces the remaining budget overshoot. We relate utility-based stopping to certainty thresholds and show when estimation errors leave the local stopping decision unchanged. The results support a modular view of adaptive stopping: score composition, eligibility safeguards, and budget feedback govern different aspects of the accuracy–cost tradeoff.

1 INTRODUCTION

Inference-time reasoning improves language model performance by allocating computational resources to intermediate problem-solving steps. Its cost, however, depends on when the generation stops. A fixed token allowance can truncate useful reasoning on one problem and prolong an already sufficient solution on another. Therefore, budget-aware evaluations consider both answer quality and the computational cost of obtaining the answer (Snell et al., 2024; Wang et al., 2024; Muennighoff et al., 2025). Effective stopping requires a decision about the expected benefit of another reasoning block, rather than a length target alone.

The central challenge is deciding how much reasoning an instance should receive. If a model reasons for too long, it may produce redundant traces, increase latency, and waste tokens. If it stops too early, it may miss constraints or return an unstable answer. Recent work shows that inference-time computation can improve reasoning, but also that accuracy alone can obscure large differences in token cost and compute efficiency (Snell et al., 2024; Wang et al., 2024). Methods such as budget forcing (Muennighoff et al., 2025), search-based reasoning (Yao et al., 2023a), and compute-aware inference search (Wu et al., 2025) improve test-time control. Recent methods learn budget assignments via constrained optimization (Zhai et al., 2026), allocate sampling budgets based on predictive uncertainty (Nguyen et al., 2026), and determine when additional responses may improve an answer (Wang et al., 2026). These developments motivate an examination of how certainty signals, stopping safeguards, and budget feedback contribute to efficient control within a reasoning trace.

We propose Adaptive Utility-Rational Allocation (AURA), a training-free framework for test-time reasoning control. At each decoding checkpoint, AURA compares a certainty-based proxy for the benefit of further reasoning with the priced cost of another reasoning block. Early stopping requires the utility condition and all stopping safeguards to hold for consecutive checkpoints. AURA combines development-tuned certainty estimation, risk and completeness checks, bounded auxiliary

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.

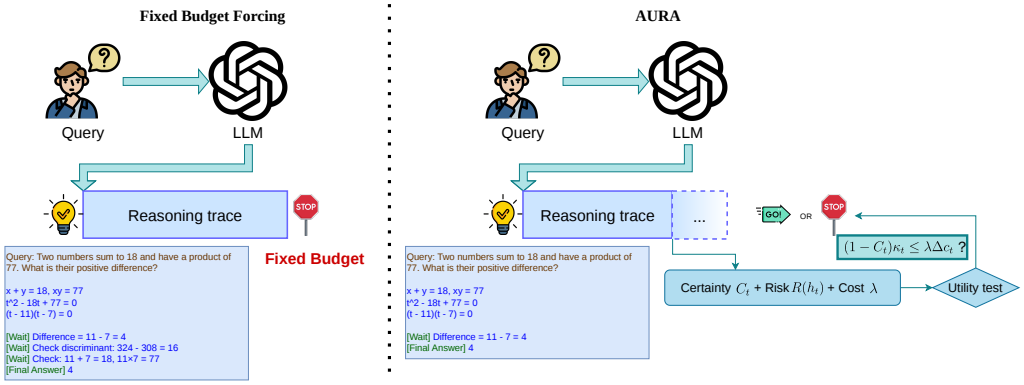


Figure 1: Comparison between fixed-budget reasoning and AURA. Fixed budget forcing allocates a static reasoning budget to each instance, which can lead to redundant continuations once the answer has stabilized. AURA permits early stopping when the estimated benefit of another reasoning block does not exceed its priced token cost and all safeguards hold for the required consecutive checkpoints.

verification, and batch-level token-price feedback toward a target average budget. Figure 1 illustrates the difference between fixed-budget reasoning and AURA's adaptive halting rule.

The experiments separate the contribution of these mechanisms from claims of improved answer quality. On MATH-500 (Lightman et al., 2023), AURA, Plan-and-Budget, and an RL halter, each obtains 93.2% accuracy at their primary operating points. AURA reduces the mean number of generated tokens by 13.6% and 14.0% relative to the latter two methods. Paired comparisons confirm significant token reductions, while showing no statistically established difference in accuracy. At matched token budgets, AURA and the baselines achieve similar observed accuracy. This suggests that AURA's primary advantage is reducing token use while maintaining comparable accuracy.

The ablation studies further clarify which components drive AURA's performance. Replacing the four-term certainty score with confidence alone leaves MATH-500 accuracy and token use essentially unchanged when the safeguards, auxiliary checks, and price update remain fixed. This result separates the choice of score from the surrounding stopping policy. It supports a simpler score in this setting, without establishing that the entropy- and stability-dependent safeguards can also be removed.

Our contributions are as follows:

- We introduce AURA, a development-tuned inference controller combining utility-based stopping, completeness and risk safeguards, bounded auxiliary checks, and batch-level token-price adaptation for frozen reasoning models.
- We characterize the relationship between utility-based stopping and fixed certainty thresholds, and establish a conditional local decision-stability bound under continuation-value and token-cost estimation errors.
- We demonstrate 13.6–14.0% token savings over Plan-and-Budget and an RL halter at the same observed MATH-500 (Lightman et al., 2023) accuracy of 93.2%. Evaluations on GPQA-Diamond (Rein et al., 2023) and TeleQuAD-Extractive (Gebre et al., 2025) (hereafter, TeleQuAD) extend the analysis through matched-token comparisons and ablations of score composition and controller stages.

The rest of the paper is organized as follows. Section 2 reviews adaptive reasoning and test-time scaling. Section 3 presents AURA and its theoretical grounding. Section 4 reports empirical results. Section 5 summarizes the findings and limitations.

2 Related Work

Adaptive reasoning and test-time control. Reasoning-oriented prompting shows that LLM performance depends strongly on how test-time computation is organized. Chain-of-thought prompting

elicits intermediate reasoning steps (Wei et al., 2022). Tree-of-Thoughts expands multiple reasoning branches with intermediate evaluation (Yao et al., 2023a), while ReAct interleaves reasoning with external actions (Yao et al., 2023b). Process supervision improves reasoning by providing feedback on intermediate steps rather than only final answers (Lightman et al., 2023). Recent methods further adapt the structure of reasoning through meta-level controllers, evolving meta-thoughts, and more structured reasoning units (Xiang et al., 2025; Liu et al., 2025; Teng et al., 2025; Li et al., 2025a).

Budget-aware inference and test-time scaling. A growing body of work treats inference compute as a resource rather than a fixed decoding parameter. Budget-aware evaluation shows that accuracy alone can hide large differences in reasoning cost (Wang et al., 2024). Test-time scaling studies further show that effective use of inference compute can complement increasing model size (Snell et al., 2024; Wu et al., 2025). Budget forcing controls reasoning length by extending or truncating generations under fixed budgets (Muennighoff et al., 2025). Other methods allocate longer generations to harder instances through token-budget controllers, planning-based decomposition, budget-control tokens, reinforcement learning, or lightweight budget predictors (Han et al., 2025; Lin et al., 2026; Wen et al., 2025; Aggarwal & Welleck, 2025; Li et al., 2025b).

Recent adaptive allocation methods. Recent allocation methods operate at different levels. Zhai et al. (2026) use Lagrangian optimization to construct budget labels and train a classifier to assign budgets to new questions. Uncertainty-Aware Budget Allocation distributes additional samples across questions using uncertainty from an initial response (Nguyen et al., 2026). AERA learns a stopping gate over successive blocks of sampled responses, using offline supervision to identify opportunities to improve answers (Wang et al., 2026). AURA controls continuation within a reasoning trace through a development-tuned certainty proxy, safeguards, and batch-level token-price updates.

Adaptive computation, routing, and domain reasoning resources. Another related direction allocates computation across models, internal computation paths, or external resources. Dynamic-depth and adaptive-pruning methods vary internal computation during inference (Chen et al., 2025; Lin et al., 2025). Entropy-adaptive decoding and routing methods select models or sampling budgets based on uncertainty, difficulty, and cost-quality trade-offs (Simonds, 2025; Ding et al., 2025). AURA controls reasoning length within a fixed model. The primary experiments evaluate adaptive stopping while holding the generator fixed, separating stopping behavior from model selection.

3 METHODOLOGY

3.1 PROBLEM SETUP

Reasoning tasks vary widely in difficulty. A fixed inference budget is therefore inefficient: easy examples may receive redundant tokens, while hard examples may stop before convergence. AURA treats halting as a decision problem. The goal is to stop when the expected benefit of further reasoning no longer justifies its compute cost.

For input x, let h_t contain x, the current reasoning prefix $\mathcal{R}_t$, and all observations available at decoding checkpoint t. Each checkpoint spans a block of main-generation tokens. Let $\hat{y}_t$ be the answer obtained from h_t using a fixed readout, and let c_t denote the cumulative charged token count, including main reasoning, probes, and verification but excluding input prompts. For a halting policy π, let $\tau_\pi \leq T_{\max}$ be its stopping checkpoint, with forced termination at the horizon $T_{\max}$. We formulate adaptive stopping through the quality–cost objective

$$J(\pi) = \mathbb{E}[\mathcal{L}(\hat{y}_{\tau_\pi}, y^\star) + \lambda c_{\tau_\pi}], \tag{1}$$

where $\mathcal{L}$ is task loss, $y^\star$ is the target answer, and $\lambda \geq 0$ is the per-token compute price. Longer reasoning is useful when its expected reduction in loss exceeds its cost.

For a specified one-checkpoint continuation procedure at $t < T_{\max}$, define the expected loss reduction and incremental charged cost as

$$\Delta_t = \mathbb{E}[\mathcal{L}(\hat{y}_t, y^\star) - \mathcal{L}(\hat{y}_{t+1}, y^\star) \mid h_t, \texttt{continue}], \tag{2}$$

$$d_t = \mathbb{E}[c_{t+1} - c_t \mid h_t, \texttt{continue}]. \tag{3}$$

The one-step comparison favors continuation when $\Delta_t > \lambda d_t$ and stopping otherwise. This compares stopping now with taking one additional checkpoint and then stopping. Since Δ_t is not directly observable during decoding, AURA uses a certainty-based proxy, supplemented by risk checks and practical safeguards.

3.2 AURA Stopping Rule

AURA computes a certainty score $C_t \in [0, 1]$ from signals available during generation:

$$C_t = \alpha \, \mathrm{Conf}_t + \beta \, (1 - \widetilde{\mathrm{Ent}}_t) + \gamma \, \mathrm{Coh}_t + \zeta \, \mathrm{Stab}_t. \tag{4}$$

The terms denote token confidence, normalized entropy, semantic coherence, and recent answer stability. Each feature lies in $[0, 1]$, and the nonnegative weights sum to one. The default weights are $(0.45, 0.25, 0.20, 0.10)$. Confidence and entropy reuse generation probabilities; coherence compares segment embeddings; stability measures agreement among extracted answers. Appendix A.1 gives their definitions. The score is a development-tuned stopping statistic rather than a calibrated probability of correctness. A confidence-only score sets $C_t = \mathrm{Conf}_t$ while retaining the same risk checks, stopping safeguards, auxiliary calls, and budget update.

The stopping rule uses certainty as a proxy for remaining value. The utility scale $\kappa_t > 0$ maps residual uncertainty into an estimated loss reduction, $\widehat{\Delta}_t = (1 - C_t)\kappa_t$. AURA accepts the utility condition when

$$(1 - C_t)\kappa_t \le \lambda \widehat{d}_t, \qquad \text{equivalently} \qquad C_t \ge \vartheta_t = 1 - \frac{\lambda \widehat{d}_t}{\kappa_t}, \tag{5}$$

where $\widehat{d}_t$ estimates the next checkpoint's cost. In the MATH-500 configuration, $\kappa_t = 1$ and $\widehat{d}_t = 64$, the nominal main-generation block size. Variable future auxiliary costs are omitted from this local estimate but included in the charged totals used for evaluation and budget feedback.

For fixed token price, utility scale, and block cost, this condition is equivalent to a fixed certainty threshold. AURA adapts the threshold between batches through the token-price update. The utility condition and all stopping safeguards must hold for m consecutive checkpoints before an early halt, reducing sensitivity to transient confidence spikes.

3.3 Risk-Aware and Adaptive Budget Control

Certainty alone can be misleading when the model is confidently wrong. AURA therefore uses risk scores to restrict early stopping and trigger auxiliary checks. Pre-probe risk depends on entropy dynamics and answer instability; post-probe risk additionally uses disagreement among candidate answers. High risk can trigger bounded answer-only probes and, when a complete candidate is available, a short verification. The resulting observations inform the final risk gate $R_t \le r_{\max}$. Appendix A.1 describes the risk interfaces, trigger thresholds, prompts, and call limits. A verification response supplies an additional model observation, not a correctness certificate.

Deployment often specifies an average token budget B. AURA responds to deviations from this target by updating the compute price after each batch:

$$\lambda_{k+1} = \left[\lambda_k + \eta(\bar{c}_k - B)\right]_+, \tag{6}$$

where k indexes batches, $\bar{c}_k$ is the mean charged token count including auxiliary generations, and η is the update step size. The MATH-500 configuration uses batches of 32 and $\eta = 10^{-6}$. The price remains fixed within each batch. If recent generations exceed the target budget, λ increases and stopping becomes more aggressive; if they underspend, λ decreases, and AURA allows longer reasoning. The update uses token counts without correctness labels and targets average cost without guaranteeing exact budget feasibility.

Algorithm 1 describes one input at a fixed price. It returns the answer and charged cost; Equation 6 is applied after the batch finishes. Observation and readout modules are fixed within an evaluation configuration.

Algorithm 1 AURA: Adaptive utility-rational allocation at a fixed token price

Require: Input x, model f, checkpoint horizon $T_{\max}$, token price λ
Require: Certainty weights $(\alpha, \beta, \gamma, \zeta)$, utility scale κ_t, cost estimate $\widehat{d}_t$
Require: Risk threshold $r_{\max}$, minimum checkpoints $t_{\min}$, consecutive count m
Require: Fixed observation and readout modules; auxiliary triggers and call limits
Initialize reasoning prefix $\mathcal{R}_0$, charged count $c_0 = 0$, and streak $q = 0$
for $t = 1$ to $T_{\max}$ **do**
    $c_t \leftarrow c_{t-1}$
    Generate the next block with f; update $\mathcal{R}_t$, h_t, and charged count c_t
    Observe checkpoint features and run permitted probes and verification
    Update h_t with auxiliary observations and charge their generated tokens to c_t
    Compute C_t, final risk R_t, and entropy-spike flag s_t
    $g_t \leftarrow [t \geq t_{\min}] \wedge \textsc{Complete}(h_t) \wedge [R_t \leq r_{\max}] \wedge \neg s_t$
    if g_t and $(1 - C_t)\kappa_t \leq \lambda \widehat{d}_t$ **then**
        $q \leftarrow q + 1$
    else
        $q \leftarrow 0$
    end if
    if $q \geq m$ or $t = T_{\max}$ **then**
        Obtain $\hat{y}_t$ from h_t using the fixed readout
        return $\hat{y}_t$ and c_t
    end if
end for

3.4 Practical Safeguards

AURA uses several safeguards to reduce premature stopping. A minimum-step guard permits early halting only when $t \geq t_{\min}$, and a completeness guard requires an extractable answer that is syntactically complete in the benchmark's output format. An entropy-spike override forces continuation when local uncertainty rises sharply. Uncertainty or elevated risk can also trigger bounded probing and verification. The utility condition and all eligibility checks must pass for m consecutive checkpoints; any failure resets the streak. The MATH-500 defaults are $t_{\min} = 4$, $m = 2$, and $r_{\max} = 0.6$. Under Algorithm 1, the earliest eligible early halt is therefore checkpoint 5. Reaching the generation horizon instead forces a readout even if the early-stopping conditions are not met. Each controller-selected halt is determined by the certainty score, compute price, risk gate, and guard outcomes.

3.5 Local Approximation Analysis

Let $\widehat{\Delta}_t = (1 - C_t)\kappa_t$ be AURA's estimated continuation value. At a fixed state h_t with $t < T_{\max}$ and fixed token price λ, assume

$$|\widehat{\Delta}_t - \Delta_t| \leq \varepsilon_\Delta, \qquad |\widehat{d}_t - d_t| \leq \varepsilon_c.$$

For the specified one-checkpoint continuation procedure, the proxy and exact utility decisions agree whenever

$$|\Delta_t - \lambda d_t| > \varepsilon_\Delta + \lambda \varepsilon_c.$$

The excess one-step action cost of the proxy decision is at most $\varepsilon_\Delta + \lambda\varepsilon_c$. This conditional result concerns the unguarded comparison at a fixed state and price; it does not establish a guarantee for the complete adaptive policy. Appendix A.2 provides the proof and a delayed-benefit counterexample explaining why the local bound does not imply global stopping optimality.

4 Experiments

We examine token savings at a selected operating point, accuracy at matched token use, and the contribution of score composition and controller stages. Figure 2 summarizes the primary and

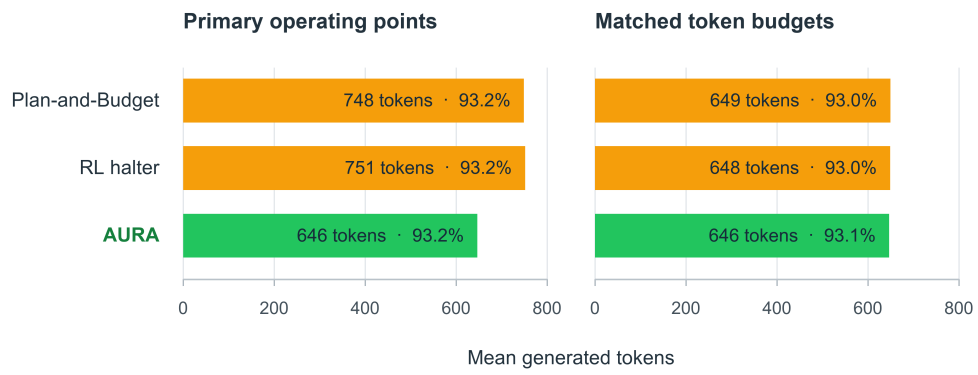


Figure 2: Token use and accuracy on MATH-500 with s1-32B. AURA uses 13.6–14.0% fewer tokens at the same observed accuracy (left), while matched-token comparisons yield similar observed accuracy (right).

matched-token MATH-500 comparisons. MATH-500 provides the primary comparison; GPQA-Diamond tests matched-token allocation at a lower accuracy level; and TeleQuAD tests component behavior on shorter, domain-specific answers.

4.1 Experimental setup

Model and configuration. We evaluate the frozen s1-32B model (Muennighoff et al., 2025) on MATH-500 using five decoding seeds on the same 500 test problems. s1-32B provides an openly available reasoning backbone with an established budget-forcing procedure, enabling controlled comparisons of stopping policies while holding the generator fixed. AURA uses token probabilities, generated text, and bounded auxiliary checks without changing model weights. Its interface can be implemented for other autoregressive models that expose token probabilities and permit control over generation at checkpoints; deployment on another backbone requires validation of its answer extraction and controller settings. AURA does not train a halting policy; controller settings for this comparison are selected on a disjoint 500-problem development set. Answer extraction, completeness checks, minimum reasoning length, and utility scale are task-specific; Appendix A.1 distinguishes these settings from the shared defaults. Main generation uses temperature 0.6, top-p 0.95, 64-token checkpoints, and a 2,048-token generation cap. The four score weights are $(0.45, 0.25, 0.20, 0.10)$ for confidence, entropy, coherence, and stability, respectively. The initial token price is 0.00150 in the primary comparison and 0.00152 in the matched-token comparison; the price is updated after batches of 32 instances. Appendix A.1 specifies the signals, safeguards, auxiliary prompts, and development search ranges.

Comparators. We compare AURA with static and s1-style budget forcing, answer-convergence stopping, certainty-threshold stopping, Plan-and-Budget (Lin et al., 2026), a classifier halter, and a reinforcement-learning (RL) halter. Static budget forcing assigns a fixed allowance; the s1-style comparator uses wait/terminate control. Answer convergence monitors extracted answers, and the certainty comparator applies an internal confidence/entropy heuristic, distinct from an implementation of certainty-guided reasoning or CoDE-Stop. The learned halters use a shallow stopping predictor or a length-penalized policy, respectively. The MATH-500 comparisons hold the backbone, input prompts, tokenizer, and maximum generation limit fixed across methods. Budgets and thresholds are selected on development data. Matching targets *mean generated tokens* rather than tokens-per-correct.

Additional tasks. GPQA-Diamond (Rein et al., 2023) extends the matched comparison to scientific question answering. TeleQuAD-Extractive (Gebre et al., 2025) evaluates domain-specific extraction with exact match (EM), providing a different answer format and shorter trajectories. These experiments support comparisons within each task, rather than isolating cross-backbone transfer or the effect of task-specific tuning. Appendix A.3.3 gives GPQA cost decomposition and paired intervals; Appendix A.3.4 gives the complete TeleQuAD score ablation.

Table 1: Primary MATH-500 comparison with s1-32B. Time is batch-amortized milliseconds per completed instance. Table 2 gives paired uncertainty for the comparisons with Plan-and-Budget and the RL halter.

Method	Accuracy (%)	Tokens	TPC	Time (ms)
Static budget forcing	92.6	880	950	314
s1-style budget forcing	93.0	824	886	296
Answer-convergence stopping	92.8	735	792	263
Certainty-threshold stopping	92.6	704	760	252
Plan-and-Budget	93.2	748	803	270
Classifier halter	93.0	764	822	276
RL halter	93.2	751	806	272
AURA	93.2	646	693	232

Table 2: Paired MATH-500 differences: AURA minus the named comparator, with 95% bootstrap intervals in brackets. Negative token differences favor AURA.

Setting	Comparator	Accuracy difference (pp)	Token difference
Primary	Plan-and-Budget	$0.00\ [-0.56, 0.56]$	$-102\ [-118.30, -85.15]$
Primary	RL halter	$0.00\ [-0.64, 0.64]$	$-105\ [-122.20, -87.76]$
Matched	Plan-and-Budget	$0.08\ [-0.68, 0.84]$	$-3\ [-18.11, 11.53]$
Matched	RL halter	$0.08\ [-0.60, 0.76]$	$-2\ [-16.27, 11.71]$

Metrics and cost accounting. Accuracy is the fraction of correct final answers. Token cost is the mean generated reasoning tokens over all instances, including incorrect answers and auxiliary probe and verification outputs. Input prompts are excluded. Tokens-per-correct (TPC) normalizes this cost by the number of correct answers:

$$\mathrm{TPC} = \frac{\sum_i c_i}{\sum_i \mathbf{1}\{\hat{y}_i = y_i^\star\}} = \frac{\text{mean tokens}}{\text{accuracy as a fraction}}. \tag{7}$$

When accuracies are equal, TPC ranks methods in the same order as token count. Generated tokens do not measure prefill or encoder computation. We therefore also report batch-amortized time, defined as batch wall time divided by completed instances. MATH-500 timing measurements use eight H200 GPUs, vLLM 0.7.3, tensor parallelism eight, concurrency 32, ten warm-up batches, and CUDA synchronization. They include prefill, extraction, and auxiliary generation, and exclude queueing, model loading, tokenization, and metric aggregation. This metric measures throughput-related cost rather than individual request latency.

Statistical analysis. For MATH-500, paired 95% bootstrap intervals use 10,000 replicates, resampling the 500 problems as clusters and retaining all five seeds for each sampled problem. This preserves method pairing and repeated evaluations. The intervals summarize variation across problems in the evaluated runs, not sensitivity to batch ordering or controller selection. Accuracy and EM differences are percentage points (pp). Contrasts use unrounded values and can differ from the subtraction of rounded entries. We report paired intervals to quantify uncertainty. Similar observed accuracy refers to close point estimates rather than statistical equivalence.

4.2 Token savings at the primary operating point

Table 1 compares the primary configurations. AURA, Plan-and-Budget, and the RL halter each achieve 93.2% accuracy. AURA uses 646 tokens per problem, compared with 748 and 751 for the latter two methods: reductions of 13.6% and 14.0%, respectively. Relative to static budget forcing, the reduction is 26.6%.

The paired token differences from Plan-and-Budget and the RL halter are -102 and -105, with intervals that exclude zero (Table 2). Both accuracy intervals contain zero. Thus, the experiment supports lower token use at the same observed accuracy for these configurations. It does not establish equal population accuracy or characterize a complete accuracy–cost frontier.

Table 3: Matched-token GPQA-Diamond comparison. Tokens include auxiliary generation.

Method	Accuracy (%)	Tokens	TPC
Plan-and-Budget	60.4	574	950
RL halter	60.2	568	943
AURA	60.8	571	939

Table 4: Certainty-score ablations on MATH-500 and TeleQuAD. The rest of each task's controller is fixed, including risk features, auxiliary checks, and price updates.

	MATH-500		TeleQuAD	
Certainty score	Accuracy (%)	Tokens	EM (%)	Tokens
Full	93.20	646.0	75.80	228.0
Confidence only	93.20	646.1	75.16	217.5
Drop confidence	93.12	720.9	75.50	251.6
Drop entropy term	93.20	648.5	75.68	230.0
Drop coherence	93.20	646.0	75.49	223.1
Uniform weights	93.20	691.6	75.74	243.1
Stability only	93.20	687.8	74.94	255.9

4.3 Comparison at matched token use

At the matched operating point, AURA achieves 93.1% accuracy with 646 tokens, Plan-and-Budget 93.0% with 649, and the RL halter 93.0% with 648 (Appendix A.3.1). Each comparator's token mean is within 0.5% of AURA's. The methods achieved similar observed accuracy at approximately 646 tokens, with paired comparisons reported in Table 2. The primary token savings therefore establish an advantage over the selected higher-cost configurations, while the matched comparison provides no evidence of an accuracy advantage at approximately 646 tokens. Matching generated tokens controls decoding volume without equalizing prefill, encoder work, or total FLOPs.

Beyond near-ceiling accuracy. On GPQA-Diamond, AURA achieves 60.8% accuracy with 571 tokens (Table 3). Plan-and-Budget and the RL halter use 574 and 568 tokens, respectively, a mismatch of at most 0.53%. The paired accuracy differences are $+0.4$ pp $[-2.2, 3.0]$ against Plan-and-Budget and $+0.6$ pp $[-2.1, 3.4]$ against RL; the token differences are -3 $[-18.1, 12.0]$ and $+3$ $[-13.2, 19.1]$. The methods also achieved similar observed accuracy on GPQA-Diamond at matched token budgets, extending the comparison beyond the high-accuracy MATH-500 setting.

4.4 Certainty-score ablations

Table 4 changes the certainty score while retaining the risk checks, stopping guards, auxiliary calls, and price update. For a dropped term, its weight is set to zero and the remaining weights are renormalized. Uniform weighting assigns $1/4$ to each term.

On MATH-500, confidence alone nearly reproduces the full score: the full-minus-confidence-only difference is 0.00 pp $[-0.12, 0.12]$ in accuracy and -0.1 $[-0.3, 0.2]$ tokens. Only three of 2,500 problem–seed evaluations have different halt locations. By contrast, removing confidence increases token use by 74.9 $[66.7, 83.2]$, and uniform weighting increases it by 45.6 $[38.2, 53.1]$.

On TeleQuAD, removing confidence adds 23.6 tokens $[20.5, 26.7]$, and uniform weighting adds 15.1 $[12.6, 17.6]$ relative to the full score. Confidence, therefore, helps avoid longer generations under these score variants on both tasks. The full score uses 10.5 more tokens than the confidence-only score, a 4.8% increase, with observed EM higher by 0.6 percentage points. The four-term score has an established token cost in this contrast but no established EM benefit.

The ablations support the separation of the certainty estimator from the stopping policy. Confidence-only is an effective, simpler score in the evaluated configurations, while entropy and stability still enter the risk checks. Its performance does not justify removing those checks, probes, or budget feedback. Eight half/double-weight perturbations yield MATH-500 token means of 646.0–686.7

Table 5: Cumulative controller ablations. Targets are $B = 646$ tokens for MATH-500 and $B = 228$ for TeleQuAD; gap is $|\bar{c} - B|$. Stage differences are descriptive, without paired uncertainty estimates.

	MATH-500			TeleQuAD		
Stage	Acc. (%)	Tokens	Gap	EM (%)	Tokens	Gap
Fixed certainty, no guards	91.9	588.8	57.2	74.4	207.8	20.2
Utility, fixed price, no guards	91.9	588.8	57.2	74.4	207.8	20.2
+ stopping guards	92.7	622.7	23.3	75.2	219.8	8.2
+ probes and verification	93.1	652.0	6.0	75.7	231.0	3.0
+ batch price update (full)	93.2	646.0	0.0	75.8	228.0	0.0

and TeleQuAD means of 224.7–239.2 (Appendix A.3.5). These bounded perturbations describe score-weight sensitivity, not robustness to changes in risk thresholds or the price-update step size.

4.5 Controller stages and auxiliary cost

Table 5 adds controller stages cumulatively on both tasks. With fixed κ, token price, and block length, the utility rule is algebraically a fixed certainty threshold; the first two rows coincide. Minimum-length, completeness, and consecutive-checkpoint guards increase the observed score by 0.8 pp on each task, with additional decoding. Probes and verification raise the point estimates by a further 0.4 pp on MATH-500 and 0.5 pp on TeleQuAD. The batch update then reduces token use from 652 to 646 and from 231 to 228, respectively, while the score estimates increase by 0.1 pp.

The same directional pattern appears on both tasks: safeguards and checks accompany higher observed quality and more decoding, while budget feedback removes the final overshoot. This distinguishes the operational role of budget feedback from the choice of certainty score. The cumulative design does not isolate individual guards or their interactions, and agreement with these two aggregate targets does not establish tracking across arbitrary budgets or input orders.

At the primary operating point, 646 tokens comprise 622.72 main-generation, 18.10 probe, and 5.18 verification tokens. Auxiliary decoding is 3.60% of the total, comparable to 8.2/228 tokens on TeleQuAD and 20.6/571 on GPQA. MATH-500 mean call counts are 1.1768 probes, 0.1052 verifications, and 10.20 MiniLM evaluations per instance. Confidence and entropy reuse decoding probabilities; stability and completeness use cached text. Batch-amortized time is 232 ms versus 270/272 ms for Plan-and-Budget/RL, reductions of 14.1%/14.7%. The timed region includes auxiliary execution. At matched token budgets, the methods showed similar observed batch-amortized times (Appendix A.3.6).

At the primary MATH-500 operating point, AURA reduces token use while retaining the same observed accuracy. The ablations distinguish the roles of certainty scoring, stopping safeguards, auxiliary checks, and batch-level budget feedback.

5 Conclusion

We introduced Adaptive Utility-Rational Allocation (AURA), a controller framework that combines utility-based stopping, risk and completeness checks, and batch-level token-price feedback without changing model weights. At the primary MATH-500 operating point, AURA uses 13.6–14.0% fewer tokens than Plan-and-Budget and an RL halter, with the same observed accuracy of 93.2%. Matched-token comparisons on MATH-500 and GPQA-Diamond yield similar observed accuracy. Confidence-only scoring nearly reproduces the full MATH-500 result, while the additional score terms increase token use on TeleQuAD. In cumulative ablations, safeguards and auxiliary checks accompany higher observed quality, and price feedback reduces budget overshoot. These findings support adaptive stopping with explicit token accounting and budget control.

The limitations of this work are its dependence on task-specific certainty and risk settings and a myopic continuation proxy, whose local analysis does not guarantee optimal stopping or budget feasibility.

AI USE STATEMENT

In this work, we used generative AI tools to draft and edit portions of the manuscript. We have not used generative AI tools for other tasks with the required disclosure. All authors checked and revised the AI-assisted writing. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Pranjal Aggarwal and Sean Welleck. L1: Controlling How Long A Reasoning Model Thinks With Reinforcement Learning, October 2025. URL http://arxiv.org/abs/2503.04697. arXiv:2503.04697 [cs].

Yilong Chen, Junyuan Shang, Zhenyu Zhang, Yanxi Xie, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. Inner thinking transformer: Leveraging dynamic depth scaling to foster adaptive internal thinking. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28241–28259, 2025. URL https://aclanthology.org/2025.acl-long.1369/.

Dujian Ding, Ankur Mallick, Shaokun Zhang, Chi Wang, Daniel Madrigal, Mirian Del Carmen Hipolito Garcia, Menglin Xia, Laks V. S. Lakshmanan, Qingyun Wu, and Victor Rühle. BEST-Route: Adaptive LLM Routing with Test-Time Optimal Compute, June 2025. URL http://arxiv.org/abs/2506.22716. arXiv:2506.22716 [cs].

F. Gebre, H. Holm, M. Gunnarsson, D. Nimara, J. Wei, V. Huang, A. Sharma, and H. G. Ranjani. TeleQuAD: A suite of question answering datasets for the telecom domain, 2025.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24842–24855, 2025. URL https://aclanthology.org/2025.findings-acl.1274/.

Cheryl Li, Tianyuan Xu, and Yiwen Guo. Reasoning-as-Logic-Units: Scaling Test-Time Reasoning in Large Language Models Through Logic Unit Alignment, February 2025a. URL http://arxiv.org/abs/2502.07803. arXiv:2502.07803 [cs].

Junyan Li, Wenshuo Zhao, Yang Zhang, and Chuang Gan. Steering LLM Thinking with Budget Guidance, June 2025b. URL http://arxiv.org/abs/2506.13752. arXiv:2506.13752 [cs].

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023. URL https://openreview.net/forum?id=v8L0pN6EOi.

Junhong Lin, Xinyue Zeng, Jie Zhu, Song Wang, Julian Shun, Jun Wu, and Dawei Zhou. Plan and Budget: Effective and Efficient Test-Time Scaling on Reasoning Large Language Models, March 2026. URL http://arxiv.org/abs/2505.16122. arXiv:2505.16122 [cs].

Zhixuan Lin, Johan Obando-Ceron, Xu Owen He, and Aaron Courville. Adaptive Computation Pruning for the Forgetting Transformer, August 2025. URL http://arxiv.org/abs/2504.06949. arXiv:2504.06949 [cs].

Qin Liu, Wenxuan Zhou, Nan Xu, James Y. Huang, Fei Wang, Sheng Zhang, Hoifung Poon, and Muhao Chen. MetaScale: Test-Time Scaling with Evolving Meta-Thoughts, March 2025. URL http://arxiv.org/abs/2503.13447. arXiv:2503.13447 [cs].

Ali Maatouk, Fadhel Ayed, Nicola Piovesan, Antonio De Domenico, Merouane Debbah, and Zhi-Quan Luo. Teleqna: A benchmark dataset to assess large language models telecommunications knowledge. *IEEE Network*, 2025. URL https://ieeexplore.ieee.org/abstract/document/11021474/.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B. Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025. URL https://aclanthology.org/2025.emnlp-main.1025/.

Manh Nguyen, Sunil Gupta, and Hung Le. Uncertainty-Aware Budget Allocation for Adaptive Test-Time Reasoning, May 2026. URL http://arxiv.org/abs/2605.26849. arXiv:2605.26849 [cs.CL].

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A Graduate-Level Google-Proof Q&A Benchmark, November 2023. URL http://arxiv.org/abs/2311.12022. arXiv:2311.12022 [cs.AI].

Toby Simonds. Entropy Adaptive Decoding: Dynamic Model Switching for Efficient Inference, February 2025. URL http://arxiv.org/abs/2502.06833. arXiv:2502.06833 [cs].

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters, August 2024. URL http://arxiv.org/abs/2408.03314. arXiv:2408.03314 [cs].

Fengwei Teng, Quan Shi, Zhaoyang Yu, Jiayi Zhang, Yuyu Luo, Chenglin Wu, and Zhijiang Guo. Atom of Thoughts for Markov LLM Test-Time Scaling, December 2025. URL http://arxiv.org/abs/2502.12018. arXiv:2502.12018 [cs].

Junlin Wang, Siddhartha Jain, Dejiao Zhang, Baishakhi Ray, Varun Kumar, and Ben Athiwaratkun. Reasoning in token economies: budget-aware evaluation of LLM reasoning strategies. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 19916–19939, 2024. URL https://aclanthology.org/2024.emnlp-main.1112/.

Ziming Wang, Ivor Tsang, and Hangwei Qian. AERA: Adaptive Evidence Residual Allocation for Efficient Test-Time Reasoning, August 2026. URL http://arxiv.org/abs/2608.27964. arXiv:2608.27964 [cs.AI].

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022. URL https://proceedings.neurips.cc/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Hao Wen, Xinrui Wu, Yi Sun, Feifei Zhang, Liye Chen, Jie Wang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. BudgetThinker: Empowering Budget-aware LLM Reasoning with Control Tokens, August 2025. URL http://arxiv.org/abs/2508.17196. arXiv:2508.17196 [cs].

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference Scaling Laws: An Empirical Analysis of Compute-Optimal Inference for Problem-Solving with Language Models, March 2025. URL http://arxiv.org/abs/2408.00724. arXiv:2408.00724 [cs].

Violet Xiang, Charlie Snell, Kanishk Gandhi, Alon Albalak, Anikait Singh, Chase Blagden, Duy Phung, Rafael Rafailov, Nathan Lile, Dakota Mahan, Louis Castricato, Jan-Philipp Franken, Nick Haber, and Chelsea Finn. Towards System 2 Reasoning in LLMs: Learning How to Think With Meta Chain-of-Thought, January 2025. URL http://arxiv.org/abs/2501.04682. arXiv:2501.04682 [cs].

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023a. URL https://proceedings.neurips.cc/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models, March 2023b. URL http://arxiv.org/abs/2210.03629. arXiv:2210.03629 [cs].

Zhiyuan Zhai, Bingcong Li, Bingnan Xiao, Ming Li, and Xin Wang. Adaptive Test-Time Compute Allocation for Reasoning LLMs via Constrained Policy Optimization, April 2026. URL http://arxiv.org/abs/2604.14853. arXiv:2604.14853 [cs].

A Additional details and analyses

A.1 Implementation Details

Training infrastructure. The reported infrastructure comprises eight NVIDIA H200 SXM GPUs, each with 141 GB of VRAM, with bf16 precision for fine-tuning and inference when supported by the backend. The compute estimates are approximately 40 node-hours (320 H200 GPU-hours) for the reported experiments and 120 additional H200 GPU-hours for pilot runs; these estimates are separate from the batch-amortized timing measurements in Appendix A.3.6.

Evaluation protocol. Within each controlled comparison, methods use the same base model, input prompt template, decoding configuration, tokenizer, and maximum generation limit. The primary MATH-500 comparison uses s1-32B with five decoding seeds on the same 500 test problems and controller selection on a disjoint 500-problem development set. Main generation uses temperature 0.6, top-$p = 0.95$, 64-token checkpoints, and a 2,048-token generation cap. Generated-token counts include main reasoning and auxiliary probe and verification outputs but exclude input prompts; encoder computation is accounted for in timing rather than token counts.

The MATH-500 paired 95% intervals for accuracy, tokens, and TPC use 10,000 bootstrap replicates, resampling problems as clusters and retaining all five seeds for each sampled problem. They preserve pairing between methods and summarize variation across the evaluated problems, conditional on the runs and controller configurations. Contrasts use unrounded measurements. Secondary tables show point estimates unless their captions specify uncertainty; neither these point estimates nor intervals containing zero establish accuracy equivalence. Appendix A.3.2 summarizes the supplementary comparisons and their documentation.

Tokens-per-correct. Tokens-per-correct (TPC) is computed as

$$\mathrm{TPC} = \frac{\text{Avg. generated tokens}}{\text{Accuracy}},$$

where accuracy is expressed as a fraction rather than a percentage. For example, in the primary MATH-500 comparison, AURA obtains 93.2% accuracy with 646 average generated tokens, giving

$$646/0.932 \approx 693.$$

TPC divides the total generated-token cost of all evaluated attempts, including incorrect answers, by the number of correct answers; lower values indicate lower cost per correct answer.

Budget matching. For token-matched comparisons, each baseline's stopping threshold or budget parameter is selected on a held-out development split to match AURA's average generated-token count within approximately $\pm 1\%$. The matched configuration is then evaluated on the test split without further retuning. This controls average decoding volume, including auxiliary outputs, but does not equalize encoder work, prefill, total FLOPs, or per-instance token use. The MATH-500 configurations appear in Table 13.

TCORE construction and validation. TCORE is a separate telecom trace-construction study based on TeleQnA (Maatouk et al., 2025). Candidate questions are selected from TeleQnA's 10,000 questions for multi-step constraints, a teacher model generates structured rationales, and automatic filters check answer consistency, completeness, and redundancy. The resulting collection contains 1,000 items spanning system architecture, interface specifications, QoS trade-offs, and protocol operations. Convergence and efficiency metadata describe answer stabilization and redundant continuation in these traces.

The rationales and metadata are teacher-generated and automatically filtered; they are not independently human-validated stopping targets. Answer-consistency and completeness filters can reject incorrect or unfinished traces, but do not establish that every retained intermediate step is valid or that an annotated stopping point is optimal. No TCORE data or labels enter the primary MATH-500 experiments. This diagnostic construction is separate from the frozen-generator comparisons and supplies no evidence that fine-tuning is necessary for AURA.

Training configurations. The separate TCORE diagnostic fine-tuning study uses the following configuration:

- **Base model:** Qwen/Qwen2.5-32B-Instruct
- **Optimizer:** AdamW with (β_1=0.9, β_2=0.95)
- **Learning rate:** 1×10^{-5} with cosine decay and 5% warmup
- **Epochs:** 5
- **Block size:** 20,000 tokens
- **Batch size:** micro-batch size of 1 with no gradient accumulation
- **Precision:** bf16 mixed precision
- **Curriculum strategy:** 3-stage progression based on trajectory length

Curriculum learning. For TCORE fine-tuning, we used a 3-stage curriculum based on difficulty annotations. Data difficulty was estimated from baseline error rate, reasoning trajectory length, and entropy profile. Easier examples were presented first, followed by longer and more uncertain trajectories. This fine-tuning is used only in the diagnostic TCORE studies; the main AURA halting results do not require telecom-specific fine-tuning.

AURA inference control. AURA evaluates each decoding checkpoint with a certainty score:

$$C_t = \alpha\,\mathrm{Conf}_t + \beta\,(1 - \widetilde{\mathrm{Ent}}_t) + \gamma\,\mathrm{Coh}_t + \zeta\,\mathrm{Stab}_t.$$

Unless otherwise stated, the weights were selected on a held-out development split and then kept fixed for test evaluation. The default development-selected weights are

$$\alpha = 0.45, \qquad \beta = 0.25, \qquad \gamma = 0.20, \qquad \zeta = 0.10.$$

The stopping threshold is induced by the utility rule

$$(1 - C_t)\kappa_t \le \lambda \widehat{d}_t,$$

where the MATH-500 local test uses $\widehat{d}_t = 64$, the nominal main-generation block cost. This estimate omits future auxiliary generation; realized probe and verification costs enter the charged totals and batch update. The MATH-500 defaults are $\kappa_t = 1$, $t_{\min} = 4$ checkpoints, $m = 2$, and $r_{\max} = 0.6$. AURA also uses a minimum-step guard, an answer-completeness check, a consecutive-halt requirement, and a short verification trigger when the risk proxy exceeds its threshold. These safeguards are applied before finalizing an answer. The main implementation follows Algorithm 1.

Confidence and entropy. Let $\mathcal{I}_t$ index the token positions in checkpoint interval t, and let $p_j(v)$ be the next-token distribution at position j. Confidence averages the maximum full-vocabulary probability:

$$\mathrm{Conf}_t = \frac{1}{|\mathcal{I}_t|} \sum_{j \in \mathcal{I}_t} \max_v p_j(v).$$

For the top-20 token set K_j, define $q_j(v) = p_j(v) / \sum_{u \in K_j} p_j(u)$ for $v \in K_j$. The normalized entropy feature is

$$\widetilde{\mathrm{Ent}}_t = \frac{-1}{|\mathcal{I}_t| \log 20} \sum_{j \in \mathcal{I}_t} \sum_{v \in K_j} q_j(v) \log q_j(v).$$

Renormalization discards the tail mass, so this feature measures uncertainty within the top-20 distribution rather than full-vocabulary entropy. Both features reuse generation logits and require no additional LLM call.

Coherence and stability. Coherence applies a fixed clipping and rescaling map to the cosine similarity between mean-pooled `all-MiniLM-L6-v2` embeddings of the current 64-token segment and the preceding 256 tokens. This feature measures local semantic consistency, not logical validity. It requires a separate encoder evaluation at each checkpoint. Stability measures the modal frequency of normalized extracted answers over the current and two preceding checkpoints. When all three checkpoints yield valid extracted answers a_j, the statistic for their window W_t is

$$\mathrm{Stab}_t = \max_a \frac{1}{3} \sum_{j \in W_t} \mathbf{1}\{a_j = a\}.$$

The equality comparison follows answer normalization. Stability uses the current trajectory and requires no auxiliary LLM generation. Repeated agreement can occur for an incorrect answer, so stability alone does not establish correctness.

Answer extraction and completeness. For MATH-500, extraction prioritizes the last balanced `\boxed{}` expression, then a `Final Answer` field, then the last standalone expression after an equality. LaTeX normalization and parsing follow extraction. Completeness requires an extracted answer in the benchmark format, balanced delimiters, no unfinished terminal operator or fraction, successful checker parsing, and no continuation marker. These requirements test syntax and answer format. The risk and utility conditions remain separate.

Risk modules and entropy override. Let $\boldsymbol{e}_t$ collect the entropy-dynamics inputs, $u_t = 1 - \mathrm{Stab}_t$ denote answer instability, and D_t denote normalized disagreement among candidate answers after probing and benchmark-specific normalization. The risk interfaces are

$$r_t^{\mathrm{pre}} = R_{\mathrm{pre}}(\boldsymbol{e}_t, u_t; \theta_{\mathrm{pre}}),$$
$$r_t^{\mathrm{post}} = R_{\mathrm{post}}(\boldsymbol{e}_t, u_t, D_t; \theta_{\mathrm{post}}),$$

where θ_{pre} and θ_{post} fix the aggregation and normalization of their inputs. These modules supply decision scores rather than calibrated error probabilities. Distinguishing their inputs from the certainty score makes their role explicit: entropy and stability can affect risk even when their certainty weights are zero.

Writing H_t for the entropy value supplied to the risk module and $\mathcal{J}_t$ for its configured recent reference window, the entropy spike is

$$E_t = \mathrm{clip}(H_t - \mathrm{median}_{j \in \mathcal{J}_t} H_j,\, 0,\, 1)\,, \qquad s_t = \mathbf{1}\{E_t \geq 0.55\}.$$

The override $s_t = 1$ forces continuation independently of the utility score. After auxiliary observations, the final risk R_t must be at most 0.60 for an early halt.

Probes and verification. A probe event is triggered when pre-probe risk is at least 0.45, or once the minimum-step requirement is met when stability is below 0.67. For an unrounded modal frequency over three valid answers, agreement at two checkpoints still meets the stability trigger because $2/3 < 0.67$. Each input permits at most two probe events, each generating two answer-only candidates with a 32-token cap per candidate. Probe decoding uses temperature 0.7 and top-$p = 0.95$, with the instruction:

> Independently solve only the unresolved final step of the problem. Return one candidate's final answer in `\boxed{}` and no explanation.

If post-probe risk exceeds 0.60 and a complete answer exists, the controller invokes verification at temperature zero, with a 64-token cap and at most one verification call per input:

> Check the candidate final answer against the original problem and the current reasoning. Return VERIFIED if it is correct; otherwise return one corrected answer in `\boxed{}`.

The verification response is an additional model observation, not a correctness certificate. All auxiliary decoded tokens enter the charged count. The call limits bound probe generation by 128 tokens and verification by 64 tokens per input. Exhausting either allowance prevents further calls of that type; it does not waive the risk, completeness, or entropy-spike conditions.

Consecutive eligibility. Algorithm 1 increments the halt streak only when the minimum-length, completeness, risk, entropy-spike, and utility conditions all pass, and resets it otherwise. With $t_{\min} = 4$ and $m = 2$, the earliest eligible MATH-500 early halt is checkpoint 5 (320 main tokens for full 64-token blocks). Auxiliary calls do not advance the streak; termination at the generation horizon is a separate condition.

Development selection. MATH-500 controller selection uses a disjoint development split of 500 examples. The candidate sets are $\kappa \in \{0.75, 1, 1.25\}$, $\rho \in \{0.25, 0.5, 0.75\}$, $r_{\max} \in \{0.5, 0.6, 0.7\}$, and $m \in \{1, 2, 3\}$. The selected tuple is $(1, 0.5, 0.6, 2)$. Selection considers error, generated tokens, and early-exit risk; ρ weights the development risk criterion and is absent from the runtime utility test. The candidate sets describe parameter selection rather than an empirical sensitivity result.

The score weights, top-20 entropy feature, risk inputs, three-checkpoint stability window, encoder and context length, consecutive count, and auxiliary prompts are shared. Answer extraction, completeness, minimum length, and utility scale are task-specific. Initial prices and baseline caps or thresholds depend on the evaluation setting; B is the target mean charged cost. Within a selected configuration, these controller choices remain fixed while the batch update adapts λ. The generator weights are unchanged throughout.

Batch prices and effective thresholds. For fixed $\kappa_t > 0$, token price, and block cost, the utility rule is a certainty threshold: $C_t \geq 1 - \lambda \widehat{d}_t / \kappa_t$. Thus, it is algebraically equivalent to a fixed certainty threshold when these quantities are constant. The initial MATH-500 prices are $\lambda_0 = 0.00150$ for the primary comparison and 0.00152 for the matched comparison, giving thresholds 0.904 and 0.90272 for $\kappa_t = 1$ and $\widehat{d}_t = 64$. Each price is fixed within a batch of 32 inputs and then updated. A 100-token overrun raises the next price by 10^{-4} and lowers the unclipped threshold by 0.0064. The update uses no correctness labels and does not override the stopping guards or guarantee exact budget feasibility.

A.2 Local Approximation Analysis

This analysis bounds the effect of continuation-value and cost-estimation errors on a single stopping decision at a fixed state and token price. It does not bound regret against an optimal multistep policy.

One-step action costs. Fix a state h_t, a token price $\lambda \geq 0$, and a one-checkpoint continuation procedure. All conditional expectations use this procedure and the same answer readout at the current and next states. Counts c_t and c_{t+1} follow the charged reasoning-token metric in Section 4. After subtracting the common baseline $\lambda \mathbb{E}[c_t \mid h_t]$, the two action costs are

$$Q_t(\texttt{halt}) = \mathbb{E}[\mathcal{L}(\hat{y}_t, y^\star) \mid h_t], \tag{8}$$

$$Q_t(\texttt{continue}) = \mathbb{E}[\mathcal{L}(\hat{y}_{t+1}, y^\star) \mid h_t, \texttt{continue}] + \lambda d_t. \tag{9}$$

Here `continue` means taking one further step and then stopping. Define the exact and proxy advantages as

$$a_t = Q_t(\texttt{halt}) - Q_t(\texttt{continue}) = \Delta_t - \lambda d_t, \qquad \widehat{a}_t = \widehat{\Delta}_t - \lambda \widehat{d}_t.$$

The exact rule continues when $a_t > 0$ and halts otherwise; the unguarded proxy rule uses the same convention for $\widehat{a}_t$.

Proposition (local decision stability). Suppose $|\widehat{\Delta}_t - \Delta_t| \leq \varepsilon_\Delta$ and $|\widehat{d}_t - d_t| \leq \varepsilon_c$ at h_t. Set $\varepsilon = \varepsilon_\Delta + \lambda \varepsilon_c$. The exact and proxy decisions agree whenever $|a_t| > \varepsilon$. For the proxy action $\widehat{u}_t$,

$$0 \leq Q_t(\widehat{u}_t) - \min_{u \in \{\texttt{halt}, \texttt{continue}\}} Q_t(u) \leq \varepsilon.$$

Proof. The triangle inequality gives $|a_t - \widehat{a}_t| \leq \varepsilon$. If $a_t > \varepsilon$, then $\widehat{a}_t > 0$, so both rules continue. If $a_t < -\varepsilon$, then $\widehat{a}_t < 0$, so both halt. When their actions differ, the excess cost is $|a_t|$. A sign disagreement under the stated tie convention implies $|a_t| \leq \varepsilon$. At $a_t = 0$, the actions have equal cost. This proves both claims.

Delayed benefits preclude a global guarantee. Consider a deterministic trajectory with losses $1, 1, 0$ at states $0, 1, 2$, one token per transition, and $\lambda = 0.1$. Let all safeguards be inactive. At state 0, the exact one-step improvement is zero. Even with zero estimation error, the one-step rule halts and incurs objective 1, whereas continuing for two steps incurs $0 + 2(0.1) = 0.2$. An optimal stopping policy must account for benefits beyond the next checkpoint. The local proposition therefore cannot be summed over steps to establish regret against a globally optimal policy.

Certainty is distinct from continuation value. Under zero–one loss, let p_t be the conditional probability that the current answer is correct and let $p_{t+1|t}$ be the corresponding probability after one continuation. Then

$$\Delta_t = p_{t+1|t} - p_t \leq 1 - p_t.$$

Even exact knowledge of current correctness provides only an upper bound on possible improvement, not the improvement itself. The certainty score C_t is not assumed to equal p_t, and its nonnegative residual $(1 - C_t)\kappa_t$ cannot represent a negative continuation gain. The proposition is thus conditional on value and cost estimation errors; it does not establish that these errors are small for the empirical controller.

Risk gating is a separate restriction. Adding a terminal risk penalty to the objective would produce the one-step advantage

$$\Delta_t + \rho\big(R_{\text{true}}(h_t) - \mathbb{E}[R_{\text{true}}(h_{t+1}) \mid h_t, \texttt{continue}]\big) - \lambda d_t.$$

A gate on current risk does not compute this quantity. AURA uses risk to restrict early stopping and trigger observations; it does not optimize a terminal-risk objective online. The local proposition excludes these restrictions, the consecutive-checkpoint requirement, verification actions, and price updates.

A.3 Additional Evaluation Results

Generated-token cost across benchmarks. Table 6 reports average generated-token cost under AURA on each benchmark. These values are descriptive diagnostics. They show how AURA allocates different average amounts of reasoning across tasks, but they should not be interpreted as a standalone measure of task difficulty because prompt format, answer format, and tokenizer behavior also affect token count.

Table 6: Average generated-token cost under AURA across benchmarks. These task-specific diagnostics are not normalized across prompt formats and do not give the halt position of the main trajectory.

Benchmark	Avg. Generated Tokens
AIME 2024	519
MATH-500	646
GPQA-Diamond	571
TCORE	634
TeleQuAD-Extractive	228

MATH-500 arithmetic consistency check. Table 7 reports a compact consistency check for selected MATH-500 primary results. The TPC values are computed from the reported accuracy and average reasoning-token counts. This table is included to make the efficiency calculation transparent.

Table 7: Arithmetic check for selected MATH-500 TPC values from the primary comparison.

Method	Acc. (%)	Avg. tokens	TPC
Static budget forcing	92.6	880	950
Plan-and-Budget (Lin et al., 2026)	93.2	748	803
Reinforcement-learning halter	93.2	751	806
AURA	93.2	646	693

Unmatched AIME halting comparison. Table 8 reports an unmatched-budget diagnostic comparison on AIME 2024 against learned halting baselines. Because the methods are not token-matched in this table, the comparison should be interpreted as an operational efficiency diagnostic rather than as the primary controlled evidence. The token-matched MATH-500 comparison in Table 13 controls average generated-token use; its paired accuracy intervals do not establish a quality advantage.

Table 8: Halting strategy comparison on AIME 2024 under unmatched budgets. Values are aggregate point estimates. TPC is computed from average reasoning tokens divided by accuracy.

Method	Accuracy (%)	Avg. tokens	TPC
Static budget forcing	56.7	731	1289
Classifier halter	57.8	655	1133
RL halter	58.0	642	1107
AURA	58.3	519	890

Sensitivity to minimum-token offset. Table 9 reports the nominal $\pm 5\%$ minimum-token-offset diagnostic.

Table 9: Sensitivity to the minimum-token offset.

Token offset	MATH-500 Acc. (%)	Avg. tokens	TPC	Interpretation
-5%	92.8	626	675	Slightly more aggressive halting
Default	93.2	646	693	Main AURA setting
$+5\%$	93.1	666	715	Slightly more conservative halting

AIME offset diagnostic. Table 10 reports the same offset perturbation on AIME 2024. The entries are aggregate point estimates.

Table 10: AIME 2024 offset sensitivity.

Token offset	AIME 2024 Acc. (%)	Avg. tokens	TPC
-5%	57.7	501	868
Default	58.3	519	890
$+5\%$	58.1	539	928

Prompt-perturbation robustness. Table 11 reports prompt-perturbation robustness on MATH-500. The settings cover concise instructions, verbose instructions, shuffled few-shot exemplars, and prompts that encourage longer reasoning. These aggregate diagnostics describe the selected perturbations without establishing invariance to arbitrary prompts.

Table 11: Prompt-perturbation robustness on MATH-500.

Prompt setting	Acc. (%)	Avg. tokens	TPC
Original prompt	93.2	646	693
Concise reasoning instruction	93.0	624	671
Verbose reasoning instruction	92.9	671	722
Shuffled few-shot exemplars	92.8	659	710
"Think carefully" instruction	93.0	686	738

Qualitative examples. Table 12 illustrates how stopping checks can distinguish redundant continuation from an incomplete answer.

Table 12: Examples of AURA's adaptive reasoning control.

Benchmark	Static generation	AURA
TCORE	Continues elaborating after resolving the SLA and protocol constraints.	An early halt is eligible once the answer and all stopping checks satisfy the controller rule.
AIME 2024	Repeats symbolic manipulation after producing an answer.	A stable answer permits an early halt only when the remaining checks also pass.

A.3.1 MATCHED-TOKEN MATH-500 RESULTS

Table 13 gives the matched-token configurations for AURA, Plan-and-Budget, and the RL halter. Configuration selection targets mean generated tokens on development data. TPC is the ratio of token cost to accuracy, rounded to the nearest integer. The paired accuracy and token differences appear in Table 2.

Table 13: Matched-token MATH-500 comparison. All three configurations use approximately 646–649 tokens per problem.

Method	Accuracy (%)	Tokens	TPC
Plan-and-Budget	93.0	649	698
RL halter	93.0	648	697
AURA	93.1	646	694

The paired TPC difference is -3.82 $[-21.85, 13.36]$ relative to Plan-and-Budget and -2.75 $[-19.55, 13.44]$ relative to the RL halter, using the same item-cluster bootstrap as the accuracy and token comparisons. Both intervals include zero. These contrasts use unrounded measurements and need not equal differences between the rounded table entries.

A.3.2 SCOPE OF THE SUPPLEMENTARY EVALUATIONS

GPQA-Diamond examines stopping at a lower accuracy level than MATH-500, while TeleQuAD-Extractive (Gebre et al., 2025) provides a short-answer diagnostic outside mathematics. Methods share the backbone, input prompts, and decoding constraints within each controlled comparison. Both supplementary evaluations used s1-32B with a Qwen2.5-compatible tokenizer and bf16 precision. GPQA-Diamond used the standard 198-question evaluation split. TeleQuAD-Extractive used the full official test set and the same five decoding seeds as MATH-500, with task-specific parameters selected on a disjoint development split.

A.3.3 GPQA-DIAMOND AT MATCHED TOKEN USE

Table 14 separates main and auxiliary generation. AURA uses 571.0 total tokens, with the two comparators within 0.53% of this mean. Plan-and-Budget's auxiliary count includes its planner output. Accounting for these outputs is necessary: comparing only the main-generation column would compare different portions of the inference procedures.

Relative to Plan-and-Budget, AURA generates 24.4 more main tokens and 27.4 fewer auxiliary tokens, yielding a net difference of -3.0 tokens. Thus, nearly equal totals can conceal different allocations between main reasoning and auxiliary computation. This decomposition describes the cost composition; its effect on answer quality is assessed by the paired comparison below.

At matched token budgets, the methods achieved similar observed accuracy on GPQA-Diamond, with accuracy ranging from 60.2% to 60.8% at 568–574 tokens per problem. These results extend the matched-budget comparison beyond the high-accuracy MATH-500 setting.

Table 14: GPQA-Diamond token decomposition. Accuracy is mean ± standard error; other columns show point estimates. Time is batch-amortized milliseconds per completed instance. Auxiliary generation includes probes and verification for AURA and planning for Plan-and-Budget.

Method	Accuracy (%)	Main	Auxiliary	Total	TPC	Time (ms)
AURA	60.8 ± 3.4	550.4	20.6	571.0	939	210
Plan-and-Budget	60.4 ± 3.5	526.0	48.0	574.0	950	213
RL halter	60.2 ± 3.5	568.0	0.0	568.0	943	211

Table 15: Paired GPQA-Diamond differences, AURA minus comparator. Brackets give the paired intervals. Accuracy differences are percentage points; token differences use total generated tokens.

Comparator	Accuracy difference (pp)	Token difference
Plan-and-Budget	$0.4\ [-2.2, 3.0]$	$-3.0\ [-18.1, 12.0]$
RL halter	$0.6\ [-2.1, 3.4]$	$3.0\ [-13.2, 19.1]$

A.3.4 Complete TeleQuAD certainty-score ablation

Table 16 includes the full, leave-one-out, single-signal, and uniform-weight configurations. Only the certainty-score composition changes; the surrounding stopping and risk checks remain fixed. A dropped coefficient is set to zero and the remaining coefficients are renormalized. A single-signal score assigns unit weight to that feature. Because TeleQuAD is an extractive short-answer task, these results assess controller behavior outside mathematics without demonstrating long-horizon reasoning performance.

Table 16: Complete certainty-score ablation on TeleQuAD-Extractive. EM denotes exact match. Values are point estimates. The full score uses weights $(0.45, 0.25, 0.20, 0.10)$ for confidence, $1-\widetilde{\mathrm{Ent}}$, coherence, and stability.

Certainty score	EM (%)	Tokens
Full	75.80	228.0
Drop confidence	75.50	251.6
Drop entropy term	75.68	230.0
Drop coherence	75.49	223.1
Drop stability	75.70	226.3
Confidence only	75.16	217.5
Entropy term only	74.99	238.2
Coherence only	75.14	264.0
Stability only	74.94	255.9
Uniform weights	75.74	243.1

The full score has the highest EM point estimate, while confidence-only uses the fewest tokens. The full-minus-confidence-only contrast is 0.6 pp $[-0.5, 1.8]$ in EM and 10.5 $[8.3, 12.7]$ tokens. Hence, the extra score terms incur an observed token cost in this comparison without an established accuracy benefit. Removing confidence increases token use by 23.6 $[20.5, 26.7]$, and uniform weighting increases it by 15.1 $[12.6, 17.6]$; the corresponding EM intervals contain zero. Together with the MATH-500 ablation, these contrasts support confidence as a strong score-level baseline and do not establish that all four terms are necessary.

The cumulative TeleQuAD stages in Table 5 describe a different intervention: adding safeguards and auxiliary calls changes the controller itself. Confidence-only in Table 16 retains these mechanisms, including entropy and stability in the risk checks. The two ablation families, therefore, address different questions and should not be interpreted as interchangeable tests of a standalone confidence threshold.

A.3.5 WEIGHT PERTURBATIONS

Eight half-weight and double-weight variants yield mean token counts from 646.0 to 686.7 on MATH-500 and from 224.7 to 239.2 on TeleQuAD. TeleQuAD EM ranges from 75.74% to 75.79%. These ranges summarize variation across the eight score-weight perturbations without isolating the effects of individual coefficients. In particular, sensitivity to score weights is distinct from sensitivity to the risk threshold, the minimum reasoning length, the consecutive-checkpoint count, or the batch-update step size. The score-composition ablations provide the more direct comparison with confidence-only.

Additional MATH-500 token summaries are 648.5 for dropping stability, 719.3 for entropy-only, and 728.7 for coherence-only. These token-only diagnostics complete the cost comparison of the component variants; without accompanying accuracy contrasts, they do not establish quality-preserving savings.

A.3.6 AUXILIARY GENERATION AND TIMING

At the primary MATH-500 operating point, the mean cost is 622.72 main tokens, 18.10 probe tokens, and 5.18 verification tokens, for a total of 646.00. The 23.28 auxiliary tokens constitute 3.60% of the total. Mean call counts are 1.1768 probe calls, 0.1052 verification calls, and 10.20 MiniLM evaluations per instance. Confidence and entropy reuse decoding logits, while stability and completeness checks use cached text. The TeleQuAD controller-stage analysis has 6.4 probe tokens and 1.8 verification tokens within its 228.0-token total, also approximately 3.60% auxiliary generation. These are average costs over all instances, not caps on individual calls.

Table 17 gives paired MATH-500 timing contrasts. At the primary operating point, AURA's 232-ms batch-amortized time is below the 270-ms and 272-ms comparator means, and the timing intervals exclude zero. At matched token use, the timing intervals include zero. This parallels the token comparisons: the timing advantage belongs to the selected primary operating points, and is not established after token matching.

Table 17: MATH-500 batch-amortized timing differences in milliseconds, AURA minus comparator. Brackets give the paired timing intervals. The matched-setting intervals are 95% confidence intervals.

Setting	Comparator	Time difference (ms)
Primary	Plan-and-Budget	-38.00 $[-43.30, -32.60]$
Primary	RL halter	-40.00 $[-45.30, -34.66]$
Matched	Plan-and-Budget	-2.00 $[-6.88, 2.69]$
Matched	RL halter	-2.00 $[-6.49, 2.31]$

The timed region includes prefill, extraction, encoder execution, and auxiliary generation, but excludes queueing, model loading, tokenization, and metric aggregation. Concurrent requests share the batch execution time, so batch-amortized measurements are not independent observations of request latency.

A.4 BASELINE CONFIGURATIONS

Table 18 summarizes the roles and method descriptions of the adaptive-reasoning baselines. All primary baselines are evaluated under the same prompt template and decoding constraints as AURA. Thresholds and budgets are selected on a held-out development split and then fixed for test evaluation. Learned stopping policies use held-out trajectories rather than test labels.

Table 18: Evaluation roles and descriptions of adaptive-reasoning baselines.

Method	Role	Implementation note
Static budget forcing	Primary comparator	Uses a fixed reasoning-token budget selected to match the target setting.
s1-style budget forcing (Muennighoff et al., 2025)	Primary comparator	Uses wait/terminate-style budget control under matched prompts and decoding settings.
Answer-convergence stopping	Primary comparator	Stops after the extracted answer remains stable across recent reasoning states.
Certainty-threshold stopping	Primary comparator	Internal confidence/entropy heuristic.
Plan-and-Budget (Lin et al., 2026)	Primary comparator	Planning and adaptive budget allocation; generated planner outputs are included in token-matched accounting.
Classifier halter	Primary comparator	Shallow classifier over held-out trajectories and decoding features; trained only on development data.
RL halter	Primary comparator	Length-penalized learned halting policy trained on held-out trajectories and evaluated without test-set retuning.
L1 length control (Aggarwal & Welleck, 2025)	Not evaluated	Method-specific reinforcement-learning length control; a relevant comparator outside the evaluated baseline set.
Budget guidance (Li et al., 2025b)	Not evaluated	Trains a remaining-length predictor and modulates token probabilities while keeping the generator frozen.
BEST-Route (Ding et al., 2025)	Not evaluated	Routing method rather than a single-model halting baseline; excluded from the primary halting comparison.

A.5 Sample Questions with Reasoning and Stopping Checks

The examples below are illustrative and are not used as additional quantitative evidence. They relate answer stability, redundancy, and completeness checks to short telecom explanations. The stopping descriptions are conceptual applications of the controller rule, not measured generation traces or halt positions.

A.5.1 Standards specification

Question: What is the purpose of the Nmfaf_3daDataManagement_Deconfigure service operation? [3GPP Release 18]

Answer: To configure the MFAF to stop mapping data or analytics received by the MFAF to outbound notification endpoints.

Reasoning: The Nmfaf_3daDataManagement service operation manages how the Messaging Framework Adaptor Function (MFAF) handles data routing. The Deconfigure operation indicates that a previously active configuration should be disabled. In this context, if the MFAF had been configured to map incoming data or analytics to external endpoints, the deconfigure operation stops that mapping.

Stopping checks: The answer specifies the mapping operation being disabled. Once it is complete and stable, the utility and risk checks determine whether further reasoning is warranted.

A.5.2 Research publications

Question: Which non-orthogonal multiple access scheme utilizes the low-complexity message passing algorithm at the receiver for user data detection?

Answer: SCMA.

Reasoning: Among non-orthogonal multiple access schemes, Sparse Code Multiple Access (SCMA) is distinguished by sparse codebooks and message-passing detection at the receiver. This sparse structure supports efficient multi-user detection when overlapping signals must be decoded.

Stopping checks: The explanation links sparse codebooks to message-passing detection. Semantic coherence can contribute to the certainty score, but does not certify the answer or justify halting by itself.

A.5.3 STANDARDS SPECIFICATION

Question: How does a supporting UE attach to the same core network operator from which it detached in a shared network? [3GPP Release 17]

Answer: It uses information stored in the UE when it was detached.

Reasoning: In shared mobile networks, a user equipment device may detach from one operator and later reattach. The UE can use locally stored operator-selection information from the detach event to identify the same core network operator without restarting the full selection process.

Stopping checks: A complete answer identifies the stored operator-selection information. Repetition of the same answer contributes to stability only after normalization and extraction.

A.5.4 RESEARCH PUBLICATIONS

Question: With two transmit antennas, one receive antenna, and independent fading paths, what is the diversity gain for each symbol in the Alamouti scheme?

Answer: 2.

Reasoning: The Alamouti scheme is a space-time block coding method with two transmit antennas and one receive antenna in the standard setting. Each symbol benefits from two independent transmit paths, so the diversity gain is 2.

Stopping checks: The stated antenna and fading assumptions make the answer well defined. A concise correct answer remains subject to the minimum-length, completeness, risk, and consecutive-checkpoint requirements.

A.5.5 RESEARCH PUBLICATIONS

Question: What is the typical coverage range of LPWANs in urban areas?

Answer: 2–5 km.

Reasoning: LPWAN technologies, including LoRa, Sigfox, and NB-IoT, are designed for low-power, long-range communication. In urban environments, buildings and other obstructions reduce propagation range. The quoted 2–5 km range is illustrative; actual coverage depends on technology, propagation conditions, antenna placement, and link budget.

Stopping checks: The answer gives a representative urban range rather than a deployment guarantee. Answer stability alone does not validate that range for a particular technology, environment, or link budget.

A.6 CODE RELEASE

Project repository: `https://anonymous.4open.science/r/AURA_26/`.

A.7 ASSET LICENSES

Models and benchmarks remain subject to their release licenses and access terms. TCORE combines TeleQnA (Maatouk et al., 2025) questions with teacher-generated annotations; TeleQuAD (Gebre et al., 2025) is a separate evaluation resource. Redistribution of TCORE depends on the terms of both the source questions and derived annotations.