FROM PARETO TO PREFERENCE: PERSONALIZED TEST-TIME SCALING VIA AMORTIZED AGENTIC POLICY DISCOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time scaling (TTS) improves the reasoning capabilities of large language models by allocating additional inference computation. Existing approaches to improving TTS efficiency largely optimize accuracy against one resource dimension at a time, advancing either the *accuracy–cost* or *accuracy–latency* Pareto frontier. Yet user requirements are multidimensional: users may specify accuracy, latency, and inference-cost requirements jointly, and different requirements can favor different controllers. We formulate *Personalized Test-Time Scaling* as discovering executable controllers that maximize the joint satisfaction rate of user-specific requirements. To reduce the overhead of repeated policy discovery for new user profiles, we propose **PersonTTS**, an amortized agentic policy-discovery framework that reuses prior search experience through requirement-matched controller initialization and source-distilled procedural guidance, while retaining target-profile evaluation for every candidate. Experiments on AIME and HMMT show that PersonTTS substantially outperforms strong TTS baselines in joint requirement satisfaction on unseen user profiles and held-out problems. Under the same candidate-evaluation budget, cross-user experience reuse further improves policy quality while substantially reducing discovery-agent time and cost.

1 INTRODUCTION

Test-time scaling (TTS) has emerged as a powerful paradigm for improving the reasoning capabilities of large language models (OpenAI, 2024; Kimi Team et al., 2025; DeepSeek-AI et al., 2025). By allocating additional computation at inference time, TTS enables extended exploration of the solution space, allowing models to explore diverse reasoning paths, evaluate candidate answers, and refine solutions for complex tasks such as mathematical reasoning (Wang et al., 2023b; Madaan et al., 2023; Lightman et al., 2024). However, scaling up this exploration incurs substantial inference overhead (Brown et al., 2024; Snell et al., 2025). To mitigate these overheads, existing approaches largely focus on advancing the *accuracy–cost* or *accuracy–latency* Pareto frontier by reducing redundant computation (Wu et al., 2025; Wang et al., 2025b; 2026a;b; Zheng et al., 2026b), for example, using adaptive sampling, early stopping, or selective branch pruning (Aggarwal et al., 2023; Li et al., 2024; Wang et al., 2025a; Zheng et al., 2026a).

While yielding promising efficiency gains, these approaches typically optimize accuracy against one resource dimension at a time, which does not fully capture a user's joint requirements. In practice, users may require a target accuracy subject to both latency and inference-cost limits (Huang et al., 2025; Wang et al., 2026c), so different requirements can favor different controllers even on the same problem set. Consequently, optimizing either Pareto frontier can fail to identify controllers that satisfy accuracy, latency, and inference-cost requirements jointly (Figure 1). We therefore formulate *Personalized Test-Time Scaling* as discovering executable controllers that maximize the joint satisfaction rate of user-specific accuracy, latency, and inference-cost requirements, shifting the focus from Pareto to preference.

Crucially, personalization affects not only how a controller is evaluated, but also how it should execute. An executable TTS controller coordinates model selection, reasoning width and depth, refinement, pruning, and stopping (Zheng et al., 2026b), and these decisions can depend on the target

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.

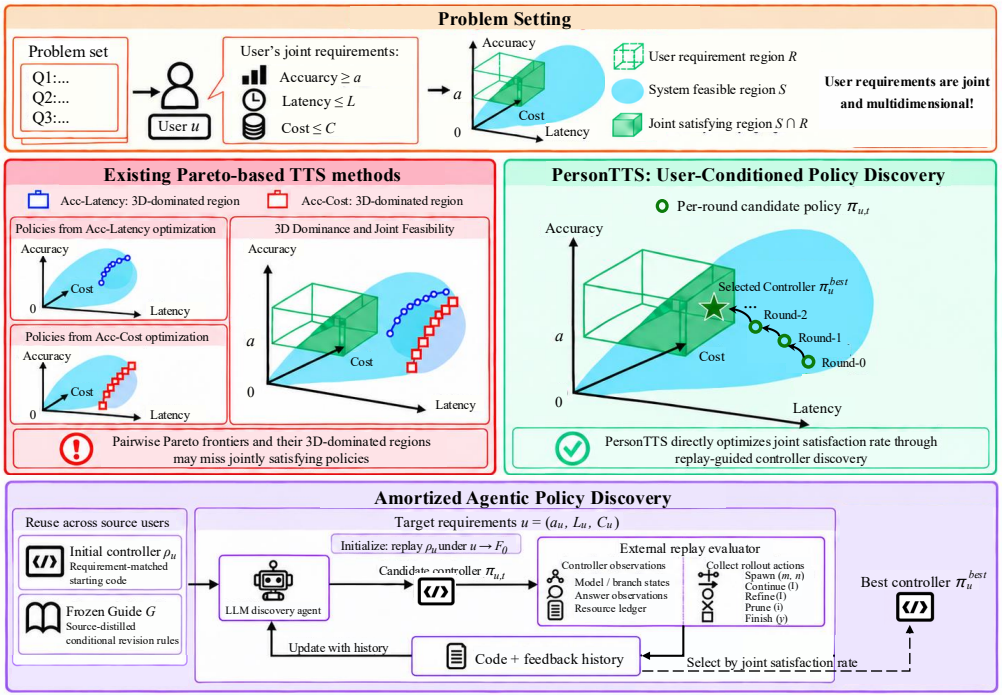

Figure 1: **Comparison of Existing Test-Time Scaling Paradigms and Personalized Test-Time Scaling.** Existing methods primarily optimize accuracy against latency or inference cost separately, whereas Personalized TTS targets their joint satisfaction under user-specific requirements. PersonTTS amortizes user-conditioned controller discovery by reusing requirement-matched controllers and procedural guidance from prior searches.

requirements. Candidate controllers must therefore be evaluated under the target profile rather than assumed to transfer across users. However, running policy discovery independently for every new profile can be costly, e.g., $39.9 and 160 minutes for a complete policy-discovery run (Zheng et al., 2026b).

To address this challenge, we propose **PersonTTS**, an amortized agentic policy-discovery framework that reuses prior search experience across users. Specifically, PersonTTS retrieves a controller from a source profile with similar requirements and re-evaluates it under the target profile to initialize discovery, while a frozen Guide distilled from source discovery histories informs subsequent proposals (see Figure 1). An LLM discovery agent then proposes executable controllers using the target requirements, replay feedback, and accumulated search history. Each candidate is evaluated under the target profile, and the controller with the highest joint satisfaction rate is retained. By combining controller reuse and procedural guidance with target-specific evaluation, PersonTTS amortizes design effort across users and enables more efficient personalized policy discovery.

We evaluate PersonTTS on AIME and HMMT (Dekoninck et al., 2026) using six Qwen3 models (Yang et al., 2025), with independently sampled source and target requirement profiles and separate held-out problem sets. PersonTTS substantially outperforms strong TTS baselines in joint requirement satisfaction on both unseen user profiles and held-out problems. Under the same candidate-evaluation budget, cross-user experience reuse further improves policy quality over independent target-side discovery while substantially reducing discovery-agent time and monetary cost. Ablation studies further characterize the distinct roles of requirement-matched initialization and procedural guidance in personalized policy discovery.

Our contributions are summarized as follows:

- We formulate **Personalized Test-Time Scaling** as executable controller discovery under joint user-specific accuracy, latency, and inference-cost requirements, with the objective of maximizing their joint satisfaction.

- We propose **PersonTTS**, an amortized agentic policy-discovery framework that reuses requirement-matched controllers and procedural discovery experience across users while preserving target-specific evaluation.
- We empirically validate PersonTTS on AIME and HMMT, demonstrating substantial gains over strong TTS baselines on unseen user profiles and held-out problems, while cross-user experience reuse further improves policy quality and reduces discovery overhead under the same candidate-evaluation budget.

2 Related Work

Efficient Test-Time Scaling. Test-time scaling improves reasoning by allocating additional inference computation through parallel sampling and aggregation (Wang et al., 2023b; Brown et al., 2024; Wu et al., 2025; Snell et al., 2025), iterative feedback and refinement (Madaan et al., 2023), or structured search over intermediate reasoning states (Yao et al., 2023; Besta et al., 2024). A growing body of work seeks to make this additional computation more efficient by reducing unnecessary exploration. Adaptive sampling and early stopping terminate generation once sufficient evidence has been accumulated (Aggarwal et al., 2023; Li et al., 2024), while methods conditioned on problem difficulty or reasoning quality allocate computation adaptively across problems or reasoning paths (Wan et al., 2025; Wang et al., 2025a; Snell et al., 2025; Wang et al., 2026a). Other approaches improve the organization of inference through branch pruning, adaptive control of search width and depth, parallel execution, or cross-branch information sharing (Wang et al., 2025b; Huang et al., 2025; Wang et al., 2026d;b; Zheng et al., 2026a;b). Despite their different mechanisms, these methods can largely be viewed as advancing the *accuracy–cost* or *accuracy–latency* Pareto frontier. In contrast, Personalized TTS considers user-specific accuracy, latency, and inference-cost requirements jointly, and seeks executable controllers that maximize their joint satisfaction rather than optimizing a single accuracy–resource Pareto frontier.

Agentic Discovery. Large language models are increasingly used as discovery agents that iteratively propose, evaluate, and refine executable artifacts, ranging from algorithms and programs to agentic workflows (Romera-Paredes et al., 2024; Liu et al., 2024; Novikov et al., 2025; Hu et al., 2025; Zhang et al., 2025). Recent approaches further leverage textual feedback or execution histories to guide subsequent proposals, enabling more targeted optimization of prompts, modules, and system code (Agrawal et al., 2026; Lee et al., 2026). AutoTTS brings this paradigm to test-time scaling by formulating TTS strategy design as executable controller synthesis in an offline replay environment, targeting the accuracy–cost Pareto frontier (Zheng et al., 2026b). PersonTTS extends this paradigm to user-specific joint accuracy, latency, and inference-cost requirements, while amortizing the otherwise repeated controller discovery across profiles through reusable prior experience.

Experience Reuse in LLMs. Prior work has explored reusing past experience to improve subsequent behavior, including reflections and distilled insights (Shinn et al., 2023; Zhao et al., 2024), reasoning and search experience distilled from successful and failed trajectories (Ouyang et al., 2026; Wang et al., 2026d), reusable executable skills (Wang et al., 2023a) and replay world simulators (Zheng et al., 2026c). At the design level, SWIFT distills reusable design knowledge from previous workflow searches, while FlowBank precomputes and selects workflows for new queries (Du et al., 2026; Yuan et al., 2026). PersonTTS instead focuses on reuse across user requirement profiles, using source controllers to initialize target discovery and distilled procedural experience to guide subsequent proposals, while retaining target-profile evaluation for controller selection.

3 Methodology

We formulate Personalized Test-Time Scaling as a user-conditioned controller-discovery problem:

> *Given a user's accuracy, latency, and inference-cost requirements, how can we efficiently automate personalized TTS policy design to maximize the probability of jointly satisfying these requirements?*

To address this problem, we propose PersonTTS, which combines feedback-driven program search with cross-user experience reuse. For each target requirement, retrieval provides an initial policy, a

frozen Guide informs candidate revisions, and replay-based selection retains the evaluated policy with the highest joint satisfaction rate.

3.1 Problem Formulation

We represent a user's requirements by a profile $u = (a_u, L_u, C_u)$, which specifies an accuracy floor $a_u \in [0, 1]$, a ceiling L_u on maximum per-question replay latency, and a ceiling C_u on mean per-question inference cost. In an environment e with models $\mathcal{M}_e$, a policy π is executable code that maps u and the current observation history h_k to an action. Public observations include model and branch availability, branch progress, intermediate answers and answer histories, and accumulated latency and cost. As correctness is assessed externally, the controller receives neither reference answers nor correctness labels.

Action space. A policy controls *model routing*, *reasoning width*, *reasoning depth*, and *self-refinement*. Calling a reasoning path a branch, we write $\pi(u, h_k) \in \mathcal{A}_e$, where

$$\mathcal{A}_e = \{\mathrm{Spawn}(m, n), \mathrm{Continue}(I), \mathrm{Refine}(I), \mathrm{Prune}(i), \mathrm{Finish}(y)\}. \tag{1}$$

Here, $\mathrm{Spawn}(m, n)$ opens n branches with model $m \in \mathcal{M}_e$, $\mathrm{Continue}(I)$ advances the selected branch set I, and $\mathrm{Refine}(I)$ starts self-refinement for eligible branches whose current stage has ended. $\mathrm{Prune}(i)$ excludes branch i from further reasoning and answer selection. $\mathrm{Finish}(y)$ ends the question with an observed answer y, or deterministic plurality voting if y is omitted. The environment determines action admissibility, while the full interface is given in Appendix G. Program search can therefore revise both resource allocation and the rules that adapt it to intermediate observations.

Joint satisfaction objective. Fix a labeled question set $\mathcal{D} = \{(q_i, y_i^\star)\}_{i=1}^N$, $N \geq 1$, a cache $\mathcal{R}_e$, and a nonempty finite replay-seed panel Ω. Each seed specifies a complete evaluation over $\mathcal{D}$, called a *seed batch*, and varies cached-branch consumption order. For complete finite replay records $\tau_u^\pi(s) = \mathrm{Replay}_e(\pi, u; \mathcal{R}_e, \mathcal{D}, s)$, define

$$A_u^\pi(s) = \frac{1}{N} \sum_i \chi_i(\tau_u^\pi(s)), \quad L_u^\pi(s) = \max_{q \in \mathcal{D}} \ell_s(q; \pi, u), \quad C_u^\pi(s) = \frac{1}{N} \sum_i c_i(\tau_u^\pi(s)), \tag{2}$$

where χ_i checks the returned answer against $y_i^\star$, c_i is the inference cost for question q_i, and $\ell_s(q; \pi, u)$ is its replay latency, with $L_u^\pi(s) = \Lambda_e(\tau_u^\pi(s))$. We therefore define the *joint satisfaction rate* (JSR) as the fraction of seed batches satisfying all three requirements simultaneously:

$$\widehat{S}_u(\pi; \mathcal{D}, \Omega) = \frac{1}{|\Omega|} \sum_{s \in \Omega} \mathbf{1}[A_u^\pi(s) \geq a_u,\ L_u^\pi(s) \leq L_u,\ C_u^\pi(s) \leq C_u]\,. \tag{3}$$

JSR measures the fraction of seed batches meeting all three requirements simultaneously. As changing u affects both the acceptance thresholds and potentially the policy's actions, candidates must be evaluated under the target profile.

3.2 User-Conditioned Agentic Policy Discovery

The search space includes branching, stopping, and refinement logic as well as model and sampling choices. To efficiently explore this space for policies satisfying a user's requirements, an LLM agent revises executable controller logic while retaining AutoTTS's offline replay foundation and within-search reuse of code and diagnostics (Zheng et al., 2026b). Cached trajectories, intermediate answers, and resource records let us evaluate candidates without additional rollouts. The controller acts on observations revealed by replay within each query, while the agent revises its code using the resulting feedback between evaluations.

A scalar JSR ranks candidates but cannot distinguish an accuracy shortfall from a resource violation. Each evaluation therefore returns $F_t = (X_t, \zeta_t, \mathcal{T}_t)$: the JSR X_t, constraint pass rates and margins in ζ_t, and sanitized executed traces $\mathcal{T}_t$ linking these outcomes to controller decisions. Appendix E.5 characterizes what exact constraint pass rates and JSR reveal about single-constraint failures. Starting from ρ_u, we obtain initial feedback F_0 and then generate B new candidates:

$$\pi_{u,t} \sim p_\theta(\cdot \mid e, u, \rho_u, \mathcal{H}_{u,<t}, \widehat{\eta}_{u,t}, G), \qquad t = 1, \ldots, B, \tag{4}$$

where $\mathcal{H}_{u,<t}$ contains accessible prior code and feedback, $\widehat{\eta}_{u,t}$ contains trace-based latency and token-cost estimates, and G is the optional Guide (Section 3.3). The initial code ρ_u is a retrieved source policy when warm-start is enabled and a generic template otherwise. It stays fixed across rounds, while the history accumulates all scored candidates, including unsuccessful revisions. Adaptation changes program code, not the agent parameters θ.

The agent inspects evidence and submits one candidate per round for external static validation and replay. However, it does not execute or score candidates itself. The full procedure is given in Algorithm 1 in Appendix A, where the discovery set, seed panel, and replay realizations are fixed. Only a strict JSR improvement replaces the incumbent, so ties retain the earlier policy. Initialization is evaluated separately from the B new scored candidates, while agent time and expenditure are measured separately.

3.3 Cross-User Experience Reuse

Offline replay removes repeated rollouts, but each new user still requires program design and diagnosis. To reduce this repetition, we retain source search histories in a *Policy Experience Bank*, including user profiles, candidate code, evaluation feedback, execution traces, and selected policies. These records support two forms of reuse: retrieval provides a starting policy, while a Guide distilled from candidate comparisons informs subsequent revisions under the new user's requirements.

Requirement-similarity policy initialization. Let $\mathcal{I}_e$ index source entries with compatible models and runtime configurations, with π_i^{src} selected for source profile u_i. To compare resource ratios alongside accuracy differences, we log-transform resource ceilings and standardize using compatible source profiles:

$$\mathbf{x}(u) = (a_u, \ln L_u, \ln C_u), \qquad z_{e,j}(u) = \frac{x_j(u) - \mu_{e,j}}{\sigma_{e,j}}, \quad j = 1, 2, 3, \tag{5}$$

where $\mu_{e,j}$ and $\sigma_{e,j}$ are source-coordinate means and standard deviations. Retrieval assumes a nonempty compatible bank, positive resource ceilings, and positive coordinate standard deviations. Equal-weight Euclidean distance then selects the initial program:

$$i^\star(u) \in \arg\min_{i \in \mathcal{I}_e} \|\mathbf{z}_e(u) - \mathbf{z}_e(u_i)\|_2, \qquad \rho_u = \pi^{\mathrm{src}}_{i^\star(u)}. \tag{6}$$

We replay the retrieved code under the target profile to obtain F_0. This establishes its target joint satisfaction rate (JSR) and supplies constraint diagnostics for the first revision, rather than relying on its source score.

Procedural skill distillation and guided discovery. The retrieved policy and its initial feedback provide a starting point, but deciding how to revise it requires evidence about alternative designs. We therefore compare source candidates under their associated user requirements, linking code changes to evaluation feedback and execution traces. An agent distills these comparisons into a shared Guide whose rules specify when a revision is appropriate, which changes to prioritize, and how to evaluate the resulting policy. Supporting and unsuccessful cases help identify the conditions under which each rule applies.

When enabled, the frozen Guide enters every proposal through Equation 4, while target feedback updates the agent's search evidence rather than the Guide itself. This allows source-side preparation to be shared across users while keeping candidate generation and final selection tied to each target profile. Construction details are provided in Appendices H and I.

3.4 Separating Requirement Changes from Execution Changes

Reusing a source policy changes both how its outcomes are judged and how it may execute. To separate these effects, fix source and target profiles u, v, a common environment, cache, question set, and finite nonempty panel Ω. For a policy π with complete finite outcomes under both profiles, we denote its joint satisfaction rate (JSR) by $f_{z,\Omega}(\pi) = \widehat{S}_z(\pi; \mathcal{D}, \Omega)$. Orient metrics so that larger is better: $\mathbf{w}_z^\pi(s) = (A_z^\pi(s), -L_z^\pi(s), -C_z^\pi(s))$ and $\mathbf{b}(z) = (a_z, -L_z, -C_z)$. For fixed positive unit

scales d_j, define

$$g^\pi(s) = \min_{1\le j\le 3} \frac{w^\pi_{u,j}(s) - b_j(v)}{d_j}, \qquad \bar{f}_\pi = \frac{1}{|\Omega|}\sum_{s\in\Omega} \mathbf{1}\{g^\pi(s) \ge 0\}. \tag{7}$$

The scales normalize units. Applying target thresholds to fixed source outcomes gives $\bar{f}_\pi$, so

$$f_{v,\Omega}(\pi) - f_{u,\Omega}(\pi) = [\bar{f}_\pi - f_{u,\Omega}(\pi)] + [f_{v,\Omega}(\pi) - \bar{f}_\pi]. \tag{8}$$

The first term captures threshold changes on fixed records, while the second captures the effect of requirement-conditioned execution. This decomposition explains why source-profile performance can inform target discovery without generally determining target-profile performance. Appendix E extends this analysis to comparisons between policies, including ranking reversals under changed requirements and the insufficiency of marginal metrics for determining JSR.

Proposition: target satisfaction bounds. Suppose finite bounds $r^\pi(s)$ satisfy $r^\pi(s) \ge \max_j |w^\pi_{v,j}(s) - w^\pi_{u,j}(s)|/d_j$ for every seed. Then $F^-_\pi \le f_{v,\Omega}(\pi) \le F^+_\pi$, where

$$F^-_\pi = \frac{1}{|\Omega|}\sum_{s\in\Omega} \mathbf{1}\{g^\pi(s) \ge r^\pi(s)\}, \qquad F^+_\pi = \frac{1}{|\Omega|}\sum_{s\in\Omega} \mathbf{1}\{g^\pi(s) \ge -r^\pi(s)\}. \tag{9}$$

The proof is in Appendix E.1. When $r^\pi(s) = 0$ for every seed, rethresholding determines target JSR exactly. More generally, the bound depends on execution deviation, not profile distance alone. Observed target deviations yield a post-replay diagnostic, while prediction before replay requires independently justified deviation bounds. Accordingly, PersonTTS uses source similarity and design experience to guide policy discovery, while retaining target execution for candidate evaluation and selection.

4 Experiments

4.1 Experimental Setup

Benchmarks and profiles. We evaluate PersonTTS on AIME and HMMT (Dekoninck et al., 2026) using six Qwen3 models with 0.6B, 1.7B, 4B, 8B, 14B, and 32B parameters (Yang et al., 2025). AIME24–25 provides 60 discovery problems, with AIME26 (30 problems) held out, while HMMT24 provides 30 discovery problems, with HMMT25 (30 problems) held out. For each problem–model pair, the replay pool contains 128 pre-sampled, checkpointed reasoning trajectories with intermediate answers, token counts for thinking, probing, and feedback, and latency records, as described in Appendix B.1. Each benchmark uses 100 source profiles for the main comparison and 20 independently sampled target profiles for cross-user experiments. We sample profiles by difficulty-stratified maximin coverage calibrated on discovery data, using the procedure in Appendix B.3; the source and target sets contain no duplicate threshold triples. The experience bank and Guide are constructed only from source-profile discovery histories, excluding target-profile records and held-out problem outcomes. Additionally, we provide the full evaluation protocol and discovery settings in Appendices B.2 and B.4, respectively.

Baselines. We compare against three baseline families. **AutoTTS** (Zheng et al., 2026b) optimizes a scalar accuracy–cost trade-off, for which we report $\beta = 0.5$ and $\beta = 1.0$ separately. **Self-Consistency** includes ASC (Aggarwal et al., 2023) and ESC (Li et al., 2024), also reported separately. Both use fixed model, width, depth, and refinement configurations while allowing runtime stopping decisions. **Parallel-Probe** (Zheng et al., 2026a) combines intermediate-answer probing, branch pruning, and self-refinement. Together, these baselines compare personalized policy discovery with scalar-objective search and predefined inference strategies under the same joint-satisfaction metric.

4.2 Main Results

To evaluate whether directly optimizing for user-specific joint requirements provides an advantage over existing TTS strategies, we compare PersonTTS and its no-reuse variant against strong external baselines under the same joint-satisfaction metric. Table 1 indicates that personalized controller

Table 1: JSR (%; higher is better) across problem and profile splits. Source and target profiles follow the Setup, and problem splits are labeled separately. PersonTTS variants use the policy with the highest JSR on discovery problems across initialization and all five rounds. PersonTTS uses both reuse mechanisms. "w/o reuse" removes both the Guide and warm-start, while the other "w/o" variants remove the named mechanism. Dashes denote unreported results. Bold marks the highest reported value in each column.

Method	**AIME24-25** (Discovery)		**AIME26** (Held-out)		**HMMT24** (Discovery)		**HMMT25** (Held-out)	
	Source	Target	Source	Target	Source	Target	Source	Target
ASC	13.53	0.14	7.03	0.00	11.91	4.89	12.03	4.98
ESC	13.28	0.14	6.22	0.00	11.22	4.89	11.83	4.98
ParallelProbeSR	23.77	10.19	23.89	9.35	23.81	19.30	18.07	11.51
AutoTTS ($\beta = 0.5$)	12.11	16.41	10.52	16.03	7.04	2.96	7.25	3.23
AutoTTS ($\beta = 1.0$)	31.15	35.58	30.11	35.47	0.00	0.01	0.00	0.15
PersonTTS (w/o reuse)	**88.35**	85.57	**78.60**	79.22	**91.41**	87.04	**85.71**	69.16
PersonTTS (w/o warm-start)	–	96.45	–	83.50	–	88.27	–	**79.09**
PersonTTS (w/o Guide)	–	95.13	–	79.60	–	**91.19**	–	74.04
PersonTTS	–	**96.57**	–	**83.85**	–	90.86	–	77.28

discovery substantially outperforms these baselines across both benchmarks and retains this advantage on held-out problems, while the no-reuse variant already preserves most of this advantage. This suggests that the primary gain comes from optimizing executable controllers for the joint user requirements themselves, rather than from cross-user experience reuse alone. Building on this stronger personalized objective, cross-user reuse further improves target-profile policies under the same candidate-evaluation budget, with the improvement persisting on held-out problems, indicating that source experience helps steer discovery toward controller designs that transfer beyond the problems used during search.

4.3 Ablation Study

To disentangle how the two forms of cross-user experience reuse contribute to policy discovery, we separately remove requirement-matched initialization and the Guide under the same target profiles and candidate-evaluation budget. Figure 2 and Table 1 reveal distinct roles for the two mechanisms: warm-start mainly improves the initial search point, whereas the Guide informs subsequent revisions and provides more consistent held-out gains across benchmarks. Their gains are not uniformly additive, suggesting that retrieved controllers act as target-dependent initialization priors, while procedural guidance is less dependent on a particular source controller.

To determine whether the policy-quality gains from experience reuse are accompanied by lower discovery overhead, we further compare agent-call time and cost under the same five-round protocol. Table 2 reveals that the Guide accounts for most of the consistent efficiency improvement: relative to no reuse, Guide-only discovery reduces agent-call time by approximately 46% and cost by approximately 36% across the two benchmarks, whereas warm-start alone mainly shortens elapsed time and can slightly increase cost. Together with the policy-quality ablation, this pattern suggests that distilled procedural experience reduces repeated diagnosis and revision effort throughout discovery, while requirement-matched retrieval primarily serves as an initialization prior whose value is more dependent on the target setting.

4.4 Analysis

We analyze the proposed PersonTTS from the following perspectives: (1) discovery-to-held-out generalization, (2) experience bank scaling, (3) cross-benchmark generalization (Appendix D.1).

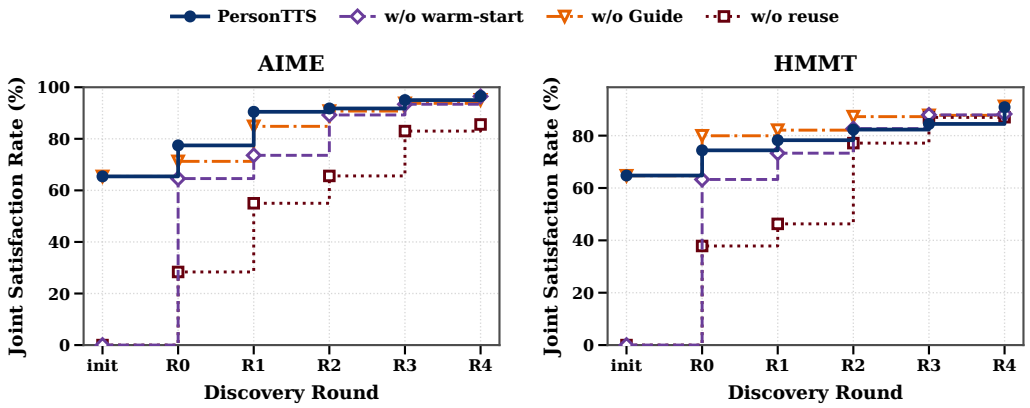

Figure 2: **Best-observed JSR on discovery sets across rounds.** Curves show the cumulative maximum JSR on the discovery sets after initialization and each of the five candidate rounds (R0–R4), averaged over target profiles on AIME24–25 and HMMT24. The four curves compare PersonTTS with the variants without warm-start, without the Guide, and without both mechanisms.

Table 2: Agent-call elapsed time and cost during target-profile discovery on discovery problems. Each cell reports time (minutes) | cost (USD), with lower values better for both metrics. Values are averaged over 20 target profiles per benchmark and round. Total gives the per-profile sum across the five rounds. Variant names follow Table 1.

Method	Round					Total
	R0	R1	R2	R3	R4	
AIME						
PersonTTS (w/o reuse)	18.67 | 1.58	17.37 | 2.29	14.56 | 2.20	12.33 | 1.94	11.70 | 2.29	74.63 | 10.30
PersonTTS (w/o Guide)	12.74 | 1.61	13.72 | 2.03	14.70 | 2.36	12.85 | 2.27	12.31 | 2.37	66.33 | 10.64
PersonTTS (w/o warm-start)	13.60 | 1.24	7.05 | 1.29	7.56 | 1.46	6.13 | 1.28	5.35 | 1.37	39.69 | 6.64
PersonTTS	8.49 | 1.15	8.90 | 1.51	8.49 | 1.67	7.83 | 1.57	6.70 | 1.50	40.41 | 7.40
HMMT						
PersonTTS (w/o reuse)	22.06 | 2.05	18.63 | 2.47	14.82 | 2.40	15.80 | 2.72	14.18 | 2.70	85.49 | 12.34
PersonTTS (w/o Guide)	15.41 | 2.25	16.14 | 2.50	14.80 | 2.46	17.12 | 2.99	12.83 | 2.56	76.30 | 12.76
PersonTTS (w/o warm-start)	14.05 | 1.40	9.00 | 1.54	8.29 | 1.67	7.66 | 1.63	7.29 | 1.62	46.29 | 7.87
PersonTTS	11.68 | 1.74	11.34 | 1.80	10.40 | 2.03	7.45 | 1.61	7.63 | 1.66	48.51 | 8.84

4.4.1 Discovery-to-Held-out Generalization

To examine whether round-wise discovery JSR is informative of out-of-sample policy quality, we evaluate the policy produced at each discovery round on both the optimization and held-out problems. Figure 3 shows that improvements in discovery JSR are generally accompanied by stronger held-out policies, with PersonTTS finishing above independent discovery on both benchmarks. The two trajectories are not perfectly aligned, however, and the persistent discovery–held-out gap indicates that discovery JSR is an informative search signal rather than a direct estimate of out-of-sample joint satisfaction.

4.4.2 Experience Bank Scaling

To understand how the amount of reusable source experience affects target discovery, we vary the number of source-profile–policy pairs available for retrieval while keeping the Guide and target-side candidate-evaluation budget fixed. For each benchmark, five bank-sampling seeds construct nested banks of 20 and 60 source-profile–policy pairs from the full 100-pair bank, and each sampled bank is used in a separate discovery run for the same target profiles.

The results in Table 3 indicate that enlarging the bank consistently improves policy quality on the discovery problems, suggesting that broader coverage of requirement-specific controller designs increases the chance of retrieving a useful starting point. This trend does not extend monotonically to

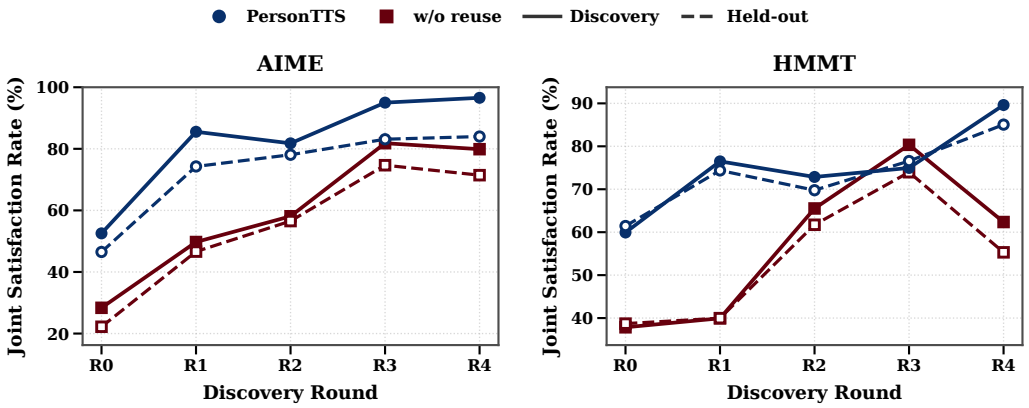

Figure 3: **Discovery-to-held-out generalization across rounds.** Solid and dashed lines report the JSR of the policy produced at each discovery round on the discovery and held-out problems, respectively, for PersonTTS and the variant without cross-user reuse on AIME and HMMT. Held-out evaluations are used only for analysis and never exposed to the discovery agent or used for policy selection.

Table 3: Experience-bank scaling: JSR (%; higher is better) of the final discovery-selected policies. Scores are averaged over available bank-seed runs within each target profile, then equally over 20 profiles. Bold marks the highest reported value in each column. The 100-profile bank is the full source bank used in the main comparison.

Bank Size	**AIME24–25** (Discovery)	**AIME26** (Held-out)	**HMMT24** (Discovery)	**HMMT25** (Held-out)
20	92.24	80.30	86.22	**86.09**
60	94.51	81.38	87.40	85.91
100	**96.57**	**83.85**	**90.86**	77.28

held-out problems, however: larger banks continue to help on AIME but reverse on HMMT, showing that broader retrieval coverage can facilitate target-side search without guaranteeing stronger cross-problem generalization. Because the Guide and candidate-evaluation budget remain fixed, this pattern isolates a limitation of retrieval-side scaling and motivates calibrated coverage or retrieval-confidence estimates rather than treating bank size as a uniformly beneficial scaling axis.

5 Conclusions

In this work, we formulate Personalized Test-Time Scaling, which seeks executable controllers that jointly satisfy user-specific accuracy, latency, and inference-cost requirements rather than optimizing a single accuracy–resource Pareto frontier. We propose PersonTTS, an amortized agentic policy-discovery framework that combines target-conditioned controller search with cross-user reuse of requirement-matched source controllers and procedural guidance, while retaining target-profile evaluation for candidate selection. Experiments on AIME and HMMT demonstrate substantial gains over strong TTS baselines on unseen user profiles and held-out problems, while cross-user experience reuse further improves policy quality and reduces discovery overhead under the same candidate-evaluation budget. Overall, our results highlight the value of amortizing personalized TTS policy design across users. Future work could combine calibrated coverage or uncertainty estimates with request-frequency-aware deployment decisions, reusing matched controllers when the experience bank is reliable and triggering background discovery for frequent or poorly covered profiles whose search cost can be amortized over future requests.

AI USE STATEMENT

In this work, we used generative AI tools to polish the manuscript. We have not used generative AI tools for other tasks with required disclosure, and the remaining disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to improve the manuscript's readability. We have reviewed the polished text. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This study uses offline replay with simulated operational requirement profiles and mathematical reasoning problems; it involves no human participants and does not measure subjective satisfaction. The reported joint-satisfaction metric is defined by accuracy and resource constraints in this replay setting. Deployment would require validation on the intended workload and resource accounting.

REPRODUCIBILITY STATEMENT

Section 3 defines the controller interface, joint-satisfaction objective, retrieval procedure, and candidate-selection loop. Section 4.1 describes the problem and profile splits, replay evaluation protocol, and discovery settings. The appendix documents replay-pool construction, profile sampling, the discovery prompt and public API, Guide distillation, round-wise measurements, and the proofs for the conditional transfer analysis.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Yiming Yang, et al. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with llms. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12375–12396, 2023.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 17682–17690, 2024.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng

Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. 2026. URL https://arxiv.org/abs/2605.00674.

Shiyi Du, Jiayuan Liu, Weihua Du, Yue Huang, Jiayi Li, Yingtao Luo, Xiangliang Zhang, Vincent Conitzer, and Carl Kingsford. Why search when you can transfer? amortized agentic workflow design from structural priors. *arXiv preprint arXiv:2604.25012*, 2026.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025.

Jenny Y Huang, Mehul Damani, Yousef El-Kurdi, Ramon Astudillo, and Wei Sun. Latency and token-aware test-time compute. *arXiv preprint arXiv:2509.09864*, 2025.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1.5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. In *International Conference on Learning Representations*, volume 2024, pp. 14751–14768, 2024.

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 *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 46534–46594, 2023.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

OpenAI. Learning to reason with llms, 2024. URL https://openai.com/index/learning-to-reason-with-llms/.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 10131–10165, 2025.

Guangya Wan, Yuqi Wu, Jie Chen, and Sheng Li. Reasoning aware self-consistency: Leveraging reasoning paths for efficient LLM sampling. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3613–3635. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.184. URL https://aclanthology.org/2025.naacl-long.184/.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023a.

Xinglin Wang, Shaoxiong Feng, Yiwei Li, Peiwen Yuan, Yueqi Zhang, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. Make every penny count: Difficulty-adaptive self-consistency for cost-efficient reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 6919–6932, 2025a.

Xinglin Wang, Yiwei Li, Shaoxiong Feng, Peiwen Yuan, Yueqi Zhang, Jiayi Shi, Chuyi Tan, Boyuan Pan, and Yao Hu. Every rollout counts: Optimal resource allocation for efficient test-time scaling. *Advances in Neural Information Processing Systems*, 38:102312–102338, 2026a.

Xinglin Wang, Hao Lin, Shaoxiong Feng, Peiwen Yuan, Yiwei Li, Jiayi Shi, Yueqi Zhang, Chuyi Tan, Ji Zhang, Boyuan Pan, et al. Share more, search less: Collaborative parallel thinking for efficient test-time scaling. *arXiv preprint arXiv:2605.27030*, 2026b.

Xinglin Wang, Zishen Liu, Shaoxiong Feng, Peiwen Yuan, Yiwei Li, Jiayi Shi, Yueqi Zhang, Chuyi Tan, Ji Zhang, Boyuan Pan, et al. On time, within budget: Constraint-driven online resource allocation for agentic workflows. *arXiv preprint arXiv:2605.06110*, 2026c.

Xinglin Wang, Jiayi Shi, Shaoxiong Feng, Peiwen Yuan, Yiwei Li, Yueqi Zhang, Chuyi Tan, Ji Zhang, Boyuan Pan, Yao Hu, et al. Do not waste your rollouts: Recycling search experience for efficient test-time scaling. *arXiv preprint arXiv:2601.21684*, 2026d.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023b. URL https://openreview.net/pdf?id=1PL1NIMMrw.

Zili Wang, Tianyu Zhang, Haoli Bai, Lu Hou, Xianzhi Yu, Wulong Liu, Shiming Xiang, and Lei Zhu. Faster and better llms via latency-aware test-time scaling. *arXiv preprint arXiv:2505.19634*, 2025b.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for llm problem-solving. In *The Thirteenth International Conference on Learning Representations*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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, 2023.

Lingzhi Yuan, Chenghao Deng, Fangxu Yu, Souradip Chakraborty, Mohammad Rostami, and Furong Huang. Flowbank: Query-adaptive agentic workflows optimization through precompute-and-reuse. *arXiv preprint arXiv:2606.11290*, 2026.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Tong Zheng, Chengsong Huang, Runpeng Dai, Yun He, Rui Liu, Xin Ni, Huiwen Bao, Kaishen Wang, Hongtu Zhu, Jiaxin Huang, et al. Parallel-probe: Towards efficient parallel thinking via 2d probing. *arXiv preprint arXiv:2602.03845*, 2026a.

Tong Zheng, Haolin Liu, Chengsong Huang, Huiwen Bao, Sheng Zhang, Rui Liu, Runpeng Dai, Ruibo Chen, Chenxi Liu, Tianyi Xiong, et al. Llms improving llms: Agentic discovery for test-time scaling. *arXiv preprint arXiv:2605.08083*, 2026b.

Tong Zheng, Xidong Wu, Zheng Zhang, Zhankui He, Chaoyi Zhang, Benjamin Coleman, Ruoqiao Wei, Di Bai, Haolin Liu, Rui Liu, Xue Wang, Yue Zhuan, Wang-Cheng Kang, Renkai Xiang, Heng Huang, Xinwu Cheng, and Yunsong Guo. Dream-rsi: Recursive self-improvement through evolving worlds, 2026c. URL https://arxiv.org/abs/2609.14858.

A User-Conditioned Policy Discovery Algorithm

Algorithm 1 User-Conditioned Policy Discovery

Require: Environment e, cache $\mathcal{R}_e$, requirements u, fixed discovery set $\mathcal{D}$ and seed panel Ω; initial policy ρ_u with a well-defined JSR; optional Guide G; B new scored candidates

$F_0 \leftarrow \textsc{Evaluate}(\rho_u; e, \mathcal{R}_e, u, \mathcal{D}, \Omega)$
$(\pi_{\text{best}}, X_{\text{best}}) \leftarrow (\rho_u, \text{JSR}(F_0))$; *records* $\leftarrow [(\rho_u, F_0)]$
for $t = 1, \ldots, B$ **do**
  $\mathcal{H}_{u,<t} \leftarrow$ accessible context from *records*
  $\widehat{\eta}_{u,t} \leftarrow$ calibration estimates from available traces
  $\pi_{u,t} \leftarrow \textsc{AgentGenerate}(e, u, \rho_u, \mathcal{H}_{u,<t}, \widehat{\eta}_{u,t}, G)$
  $\textsc{StaticValidate}(\pi_{u,t})$
  $F_t \leftarrow \textsc{Evaluate}(\pi_{u,t}; e, \mathcal{R}_e, u, \mathcal{D}, \Omega)$
  Append $(\pi_{u,t}, F_t)$ to *records*
  if $\text{JSR}(F_t) > X_{\text{best}}$ **then**
    $(\pi_{\text{best}}, X_{\text{best}}) \leftarrow (\pi_{u,t}, \text{JSR}(F_t))$
  end if
end for
return π_{best}

B Implementation Details

B.1 Offline Replay Pool Construction

We construct a frozen replay pool from offline model rollouts. For each problem–model pair in Section 4.1, we collect 128 rollouts containing extracted answers, think/probe/feedback token counts, and wall-clock latency records. Each rollout consists of an initial reasoning stage followed by up to 5 self-refinement stages. Within each stage, we place a checkpoint every 500 thinking tokens and use an answer probe of approximately 30 tokens to extract the current answer.

Rollout generation uses temperature $T = 0.6$, top-$p = 0.95$, and top-$k = 20$. Correctness is checked by answer extraction and symbolic comparison with the reference answer. The replay cache excludes question text, raw reasoning, and feedback text; correctness labels are not exposed to controllers.

B.2 Replay Evaluation and Cost Accounting

To compare policies under identical evaluation conditions, all methods share profiles, replay pools, 1,000 evaluation seeds, and resource-accounting rules within each benchmark split. The same seed panel is used across discovery rounds. Seeds reorder branch consumption rather than problems; replay is deterministic given the policy, profile, problem set, cache, seed, and execution settings, so these seeds are not independent discovery runs. All controller evaluations use CPU-only offline replay without additional task-model calls.

Within each seed, accuracy and weighted compute are per-question means, while latency is the maximum per-question replay latency defined in Equation 2, $\Lambda_e(\tau_u^\pi(s)) = \max_{q \in \mathcal{D}} \ell_s(q; \pi, u)$. This maximum is a replay statistic, not measured batch wall time. Latency is computed from stored rollout timings: sequential phases add, while each parallel phase contributes its cohort maximum.

We calculate weighted compute by summing think, probe, and feedback tokens over consumed branches, with each token weighted by its model's parameter count normalized to Qwen3-8B [1]. Profile cost budgets apply to the mean weighted compute per question. Discovery-agent dollar expenditure is measured and reported separately from replay inference cost.

We report joint satisfaction rate (JSR), as defined in Equation 3, averaged equally over profiles within each benchmark; cross-benchmark summaries average the two benchmark scores. Controller access restrictions are specified in Appendix C.

[1] https://www.alibabacloud.com/help/en/model-studio/model-pricing

B.3 Requirement Profile Sampling

Calibration and profile generation are performed separately for AIME and HMMT, using discovery problems only. The calibration script, `calibrate_profile_grid.py`, evaluates the Cartesian product in Table B.1, giving $6 \times 5 \times 7 \times 6 = 1{,}260$ anchor configurations per benchmark.

Table B.1: Calibration grid. Depth is a checkpoint limit per stage; the refinement limit counts additional stages after the initial stage.

Dimension	Values
Model	Qwen3-0.6B, 1.7B, 4B, 8B, 14B, 32B
Rollout width	$\{1, 4, 8, 16, 64\}$
Depth per stage	$\{1, 2, 4, 8, 16, 32, 66\}$ checkpoints
Self-refinement limit	$\{0, 1, 2, 3, 4, 5\}$

The calibration policy spawns the configured width as one parallel cohort, uses no early stopping or pruning, and refines only branches that reach a cached terminal stage within the depth limit. It returns a deterministic plurality vote over branch answers. Each configuration is replayed under 100 seeds over all 60 AIME or 30 HMMT discovery problems. Within-seed metrics are aggregated as in Equation 2. Let $(A_k^{\mathrm{anc}}, L_k^{\mathrm{anc}}, C_k^{\mathrm{anc}})$ denote the recorded calibration summary for anchor k.

The profile-generation script, `generate_profiles_stratified.py`, uses empirical quantile functions Q_A, Q_L, Q_C from the anchor summaries. Candidate thresholds are drawn independently in quantile space:

$$\begin{aligned} z_A, z_L, z_C &\overset{\mathrm{iid}}{\sim} \mathrm{Uniform}(0.03, 0.97), \\ u &= (Q_A(z_A), Q_L(z_L), Q_C(z_C)) = (a_u, L_u, C_u). \end{aligned} \tag{10}$$

Its feasibility ratio is the fraction of anchor configurations whose recorded calibration summaries satisfy all three thresholds:

$$\mathrm{FR}(u) = \frac{1}{1260} \sum_{k=1}^{1260} \mathbf{1}[A_k^{\mathrm{anc}} \geq a_u \ \wedge \ L_k^{\mathrm{anc}} \leq L_u \ \wedge \ C_k^{\mathrm{anc}} \leq C_u]\,. \tag{11}$$

A lower ratio marks a harder requirement profile relative to this anchor family. We use the ratio to stratify profile difficulty; it measures anchor coverage rather than feasibility for every possible controller.

Candidates are divided into five equal-frequency bins by feasibility ratio. Within each bin, greedy maximin selection chooses the candidate with the greatest minimum distance to the profiles already selected in that bin, using the sampler's normalized constraint space. With generation seed 20260901, the source set contains 100 profiles per benchmark, with 20 selected from each bin. The 20-profile target sets are sampled independently with distinct seeds using the same procedure; no complete threshold triple is shared with the corresponding source set.

B.4 Discovery Settings

Policy discovery follows Algorithm 1 with $B = 5$ new-candidate evaluations per profile and a separate initialization evaluation. All scored rounds are retained even after incumbent joint satisfaction rate (JSR) reaches one. We use `claude-opus-4-6` as the discovery agent, with a one-hour session limit and a 4,000-token thinking-budget setting. The agent interface does not expose temperature, so the provider default is used. Infrastructure retries have a separate allowance. The discovery prompt is provided in Appendix F. Source runs use PersonTTS (w/o reuse). On the same target profiles, we compare PersonTTS (w/o reuse), PersonTTS (w/o Guide), PersonTTS (w/o warm-start), and PersonTTS under the same budget. For each PersonTTS variant, the policy with the highest JSR on discovery problems is frozen for evaluation on held-out problems. Outcomes on held-out problems never inform selection, while target-local discovery feedback remains available during adaptation.

C Controller and Discovery Configuration

Table C.1 separates controller execution, candidate generation, and external evaluation.

Table C.1: Information and permissions by component.

Component	Inputs	Operations
Controller	Requirements, model availability, remaining rollouts, branch progress, answer histories, and resource ledger.	Permitted replay actions only.
Discovery agent	Starting code, accessible code/feedback history, bounded sanitized trace summaries, and optional Guide.	Edits `candidate.py`; cannot execute or score it.
External evaluator	Submitted code, replay cache, and reference answers.	Executes code, records costs, checks answers, and returns permitted feedback.

Static validation restricts controller code to the permitted API and prohibits file, network, and reflection access. These restrictions do not prevent the discovery agent from editing candidate code. Branch models are fixed at spawning. Gold answers and held-out evaluation results are not returned to the controller or discovery agent.

The agent interface does not expose temperature, so the provider default is used. Infrastructure retries have a separate allowance. All conditions retain five scored rounds, including after incumbent joint satisfaction rate (JSR) reaches one.

D Additional Analysis

D.1 Cross-Benchmark Generalization

To isolate how much controller structure can transfer across problem distributions without target-specific search, we directly reuse requirement-matched policies discovered on the other benchmark and evaluate them unchanged on the target benchmark. This setting removes the discovery agent and Guide, allowing the effect of requirement-matched policy transfer to be examined independently.

Table D.1 shows that controllers retrieved by requirement similarity remain useful even without target-benchmark discovery, indicating that the retrieved policies can retain useful controller structure under a shift in problem distribution. Their performance nevertheless remains substantially below target-native PersonTTS, suggesting that requirement matching is best viewed as a useful initialization mechanism rather than a substitute for target-specific discovery.

Table D.1: Cross-benchmark evaluation: JSR (%; higher is better), averaged over 20 target profiles with 1,000 replay seeds each. Baseline mean/max summarize the five external configurations in Table 1 plus AutoTTS-template, which is not reported there. PersonTTS reports the target-native result on the corresponding target profiles and held-out problems in that table.

Target	Baseline mean	Baseline max	Matched transfer	PersonTTS
HMMT25 (← AIME)	4.14	11.51	25.10	77.28
AIME26 (← HMMT)	10.14	35.47	43.04	83.85

E What Source Comparisons Imply under New Requirements

E.1 Target Satisfaction Bounds

We prove the bound in Equation 9. The source margin $g^{\pi}(s)$ uses the target thresholds, and $r^{\pi}(s)$ bounds the normalized source–target execution deviation on the same finite seed panel.

Proof. Define the minimum standardized target-execution margin

$$g_v^\pi(s) = \min_j \frac{w_{v,j}^\pi(s) - b_j(v)}{d_j}. \tag{12}$$

By assumption, each normalized target margin differs from its source counterpart by at most $r^\pi(s)$. Their minima therefore satisfy

$$|g_v^\pi(s) - g^\pi(s)| \le r^\pi(s). \tag{13}$$

Hence $g^\pi(s) \ge r^\pi(s)$ implies $g_v^\pi(s) \ge 0$, while $g_v^\pi(s) \ge 0$ implies $g^\pi(s) \ge -r^\pi(s)$. Therefore

$$\mathbf{1}\{g^\pi(s) \ge r^\pi(s)\} \le \mathbf{1}\{g_v^\pi(s) \ge 0\} \le \mathbf{1}\{g^\pi(s) \ge -r^\pi(s)\}. \tag{14}$$

Averaging over $s \in \Omega$ gives $F_\pi^- \le f_{v,\Omega}(\pi) \le F_\pi^+$. □

When $r^\pi(s) = 0$ for every seed, rethresholding determines the target joint satisfaction rate (JSR) exactly. More generally, the bound depends on execution deviation rather than profile distance alone.

E.2 Comparing Programs on Fixed Source Records

Fix an environment, trajectory cache, question set $\mathcal{D}$ with $N \ge 1$ questions, and a nonempty finite indexed replay panel Ω, with $m = |\Omega|$. Two complete programs π, π' have complete, finite outcomes $(A_u^p(s), L_u^p(s), C_u^p(s))$ under the same source profile u, for $p \in \{\pi, \pi'\}$ and $s \in \Omega$, with accuracy in $[0, 1]$ and nonnegative resources. Accuracy is the batch average, latency the maximum over questions, and cost the mean over questions, as in Equation 2. Comparisons concern whole programs rather than isolated causal effects. Seed independence is not assumed.

Let $\mathcal{P} = [0, 1] \times [0, \infty) \times [0, \infty)$ be the space of finite requirement thresholds. Each stored outcome defines an acceptance region and a reclassified score:

$$\begin{aligned} R_{u,s}^p &= [0, A_u^p(s)] \times [L_u^p(s), \infty) \times [C_u^p(s), \infty), \\ \bar{J}_{u\to z}^p(s) &= \mathbf{1}\{z \in R_{u,s}^p\}, \qquad \bar{S}_{u\to z}(p) = \frac{1}{m}\sum_{s\in\Omega} \bar{J}_{u\to z}^p(s), \\ \bar{\Delta}_u(z) &= \bar{S}_{u\to z}(\pi') - \bar{S}_{u\to z}(\pi). \end{aligned} \tag{15}$$

Bars denote threshold changes applied to *fixed source outcomes*, not executions under z. Write $\widehat{S}_z(p) = \widehat{S}_z(p; \mathcal{D}, \Omega)$ for the actual execution joint satisfaction rate (JSR) from Equation 3. At the source, $\bar{S}_{u\to u}(p) = \widehat{S}_u(p)$. Reclassification at another profile need not equal $\widehat{S}_z(p)$.

Proposition 1 (Requirement-dependent comparisons of fixed records)**.** *Define the gained and lost success sets $G_u(z) = \{s \in \Omega : z \in R_{u,s}^{\pi'} \setminus R_{u,s}^{\pi}\}$ and $H_u(z) = \{s \in \Omega : z \in R_{u,s}^{\pi} \setminus R_{u,s}^{\pi'}\}$. For every $z \in \mathcal{P}$,*

$$\bar{\Delta}_u(z) = \frac{|G_u(z)| - |H_u(z)|}{m}. \tag{16}$$

The comparison is constant within each region induced by the threshold hyperplanes $a_z = A_u^p(s)$, $L_z = L_u^p(s)$, and $C_z = C_u^p(s)$ for $p \in \{\pi, \pi'\}$ and $s \in \Omega$, with equality faces and domain boundaries treated separately. If v relaxes u, namely $a_v \le a_u$, $L_v \ge L_u$, $C_v \ge C_u$, let $n_p(u, v)$ count seeds unsuccessful at u but successful at v under reclassification. Then

$$\bar{\Delta}_u(v) = \bar{\Delta}_u(u) + \frac{n_{\pi'}(u, v) - n_\pi(u, v)}{m}. \tag{17}$$

In particular, a strict source advantage $\bar{\Delta}_u(u) > 0$ reverses strictly if and only if

$$n_\pi(u, v) - n_{\pi'}(u, v) > m\bar{\Delta}_u(u). \tag{18}$$

Equality gives a tie. A smaller left-hand side preserves strict advantage.

Proof. For each seed, the two joint indicators differ by $+1$ on $G_u(z)$, -1 on $H_u(z)$, and zero otherwise. Summing gives Equation 16. Within each specified region, every comparison of a threshold coordinate with a stored outcome coordinate has fixed truth value. All joint indicators are therefore constant there. Treating equality faces separately handles the convention that meeting

a threshold exactly counts as success. Under relaxation, a successful fixed record cannot become unsuccessful. Consequently, $\bar{S}_{u\to v}(p) = \widehat{S}_u(p) + n_p(u,v)/m$. Subtracting the identity for π from that for π' gives Equation 17. Rearranging gives the strict reversal, tie, and preservation conditions. □

Each fixed-record score is nondecreasing under relaxation, while its difference from another score need not be. This monotonicity does not automatically extend to profile-conditioned executions.

E.3 Identical Marginals Can Conceal a Ranking Reversal

Table E.1 gives synthetic arrays consistent with the aggregation definitions. With $N = 10$, accuracy 1 or 0.5 means ten or five correct answers, and each question can have the row's latency and cost. These are not measured records or a claim of realizability in the particular cache.

Table E.1: Synthetic source records with identical marginal distributions. Each triple is (batch accuracy, maximum question latency, mean question cost).

Seed	π: $(A_u^\pi, L_u^\pi, C_u^\pi)$	π': $(A_u^{\pi'}, L_u^{\pi'}, C_u^{\pi'})$
1	$(1, 2, 1)$	$(1, 1, 1)$
2	$(1, 3, 1)$	$(1, 3, 3)$
3	$(0.5, 1, 3)$	$(0.5, 2, 1)$

Choose $u = (0.75, 1.5, 2)$ and $v = (0.75, 3.5, 2)$, so only the latency ceiling is relaxed. At u, π succeeds on no seed and π' on seed 1. At v, reclassification accepts seeds 1 and 2 for π, but still only seed 1 for π'. Source scores $(0, 1/3)$ therefore become $(2/3, 1/3)$, reversing $\bar{\Delta}_u$ from $1/3$ to $-1/3$. Here $n_\pi = 2$ and $n_{\pi'} = 0$, verifying the reversal criterion.

Yet both programs have the same marginal multisets: $A : \{1, 1, 0.5\}$, $L : \{1, 2, 3\}$, and $C : \{1, 1, 3\}$. Their cross-seed mean vectors are both $(5/6, 2, 5/3)$, and their marginal pass rates at v are both $(2/3, 1, 2/3)$. Their identical full marginals give identical individual pass rates at *every* threshold, yet different joint scores here. Thus even full marginals, and hence their means or individual pass rates, cannot determine joint satisfaction rate (JSR) or comparison direction because they omit which conditions hold together in the same seed batch.

A sufficient ordering condition. Stronger paired evidence does preserve noninferiority. If every s satisfies

$$A_u^{\pi'}(s) \geq A_u^\pi(s), \qquad L_u^{\pi'}(s) \leq L_u^\pi(s), \qquad C_u^{\pi'}(s) \leq C_u^\pi(s), \tag{19}$$

then $\bar{\Delta}_u(z) \geq 0$ for all $z \in \mathcal{P}$. Indeed, these inequalities imply $R_{u,s}^\pi \subseteq R_{u,s}^{\pi'}$, so each old success is also a new success. Conversely, per-seed region containment implies the three inequalities by choosing $z = (A_u^\pi(s), L_u^\pi(s), C_u^\pi(s))$. This equivalence is per seed. Paired dominance is only sufficient for the aggregate ordering. Swapping $(1, 1, 2)$ and $(1, 2, 1)$ between two seeds preserves every aggregate JSR but violates paired dominance. Nor does dominance imply strict improvement everywhere. At $a_z = 0$ and ceilings above all finite recorded resources, both scores equal one.

E.4 Reclassification versus Target Execution

Now suppose both programs are also executed under v on the same environment, cache, question set and seed panel, with complete finite target outcomes. For $z \in \{u, v\}$, define the actual comparison and execution discrepancies

$$\Delta(z) = \widehat{S}_z(\pi') - \widehat{S}_z(\pi), \qquad \epsilon_p(u,v) = \widehat{S}_v(p) - \bar{S}_{u\to v}(p). \tag{20}$$

Adding and subtracting the reclassified scores yields

$$\begin{aligned} \Delta(v) - \Delta(u) = \quad & \underbrace{\bar{\Delta}_u(v) - \bar{\Delta}_u(u)}_{\text{threshold changes on source records}} \\ + \quad & \underbrace{\epsilon_{\pi'}(u,v) - \epsilon_\pi(u,v)}_{\text{profile-conditioned execution changes}} . \end{aligned} \tag{21}$$

To verify the identity, substitute $\widehat{S}_v(p) = \bar{S}_{u\to v}(p) + \epsilon_p(u,v)$ for each program and use $\Delta(u) = \bar{\Delta}_u(u)$. The fixed-record analysis above describes only the first term. If every paired outcome triple remains unchanged for both programs, then each $\epsilon_p = 0$. In this case, reclassification gives the exact target joint satisfaction rate (JSR) and the fixed-record comparison applies to actual execution. This special case is not assumed for the source bank. The same code, cache and seed do not establish outcome invariance when the program reads the profile.

Source comparisons support contextual revision hypotheses, while target replay evaluates proposed programs. This analysis does not assert complete per-seed bank records, implemented cell enumeration or dominance checks, or sufficient Guide conditions. It guarantees neither nearest-neighbor transfer, held-out generalization, nor discovery-cost savings.

E.5 What Constraint Diagnostics Identify

Fix one program, one profile, and a complete finite panel Ω of $m \geq 1$ equally weighted seeds. Let $E_A, E_L, E_C \subseteq \Omega$ contain the seeds passing each requirement, and write $r_i = |E_i|/m$ and $J = |E_A \cap E_L \cap E_C|/m = \widehat{S}_u(\pi)$. These are exact pass fractions, not rounded estimates. For distinct $i, j, k \in \{A, L, C\}$, define $H_i = |E_j \cap E_k \cap E_i^c|/m$, the fraction failing only constraint i. Complements are relative to Ω.

Proposition 2 (Sharp diagnostic limits)**.** *Every feasible summary (r_A, r_L, r_C, J) on an m-seed panel satisfies*

$$\max\{0, r_j + r_k - 1 - J\} \leq H_i \leq \min\{1 - r_i, r_j - J, r_k - J\}. \tag{22}$$

For each fixed i separately, both endpoints and every intervening multiple of $1/m$ are attained by abstract pass panels with the same summary. For modified sets satisfying $E'_j = E_j$ and $E'_k = E_k$, the new joint rate obeys $-J \leq J' - J \leq H_i$. If also $E_i \subseteq E'_i$, then $0 \leq J' - J \leq H_i$.

Proof. Let $a = mr_j$, $b = mr_k$, $c = mr_i$, $t = mJ$, and $q = |E_j \cap E_k|$. Two-set intersection bounds, containment of the t joint successes, and room for the $c - t$ other members of E_i give

$$\max\{t, a + b - m\} \leq q \leq \min\{a, b, t + m - c\}.$$

Since $mH_i = q - t$, subtracting t proves the bounds. Conversely, take any integer q in this interval. Partition the panel into four sets of sizes $q, a - q, b - q, m - a - b + q$, representing respectively the intersection of E_j, E_k, their two exclusive parts, and neither. All sizes are nonnegative. Choose E_i to contain t intersection members and $c - t$ complement members. The interval guarantees room for both choices. This realizes the prescribed summary and $mH_i = q - t$, proving sharpness at every allowed grid value.

Finally, preserving E_j, E_k gives $0 \leq J' \leq q/m$, hence $-J \leq J' - J \leq H_i$. If $E_i \subseteq E'_i$, no old success is lost. Without the pass-preservation restriction, $E'_i = \emptyset$ or Ω attains the respective extreme changes at the pass-array level without asserting executable realizability. □

When summaries suffice. If $r_A = 1$, then $E_A = \Omega$, and hence $H_L = r_C - J$, $H_C = r_L - J$, and $H_A = 0$. If also $r_C = 1$, then $H_L = 1 - J = 1 - r_L$. More generally, $r_C = J$ implies that any strict joint satisfaction rate (JSR) improvement must increase the cost pass fraction, since $J' \leq r'_C$. This does not require $r_A = 1$ and holds even if accuracy and latency pass sets change.

When summaries remain ambiguous. Consider synthetic three-seed pass arrays, with coordinates (A, L, C):

$$P = ((1,1,1),(0,1,1),(1,0,0)), \qquad Q = ((1,1,1),(1,0,1),(0,1,0)).$$

Both have $r_A = r_L = r_C = 2/3$ and $J = 1/3$, but their (H_A, H_L) values are $(1/3, 0)$ and $(0, 1/3)$ respectively. Thus even the joint score and all marginal pass rates need not identify which isolated repair could help. These arrays are illustrative, not measured cache outcomes.

This application of elementary finite-set intersection bounds concerns outcome-level opportunities. It does not implement a repair rule or assume that a continuation-cap edit changes latency alone. Source summaries describe source executions. When the paired outcome triples remain unchanged, reclassification of source records coincides with target execution. Target replay continues to evaluate the full generated program.

F PROMPTS

In this section, we present the full prompt used in our policy-discovery experiments.

PersonTTS Policy-Discovery Agent Prompt

System Message:
You are the analysis-and-implementation Agent for one AutoTTS-style policy-discovery round. Analyze the supplied aggregate feedback and sanitized executed traces, then edit `candidate.py` in this same session. You may write only `candidate.py`. Your only permitted workspace is {`allowed_root`}. For structured file tools, use absolute paths only under that exact workspace. For Bash, the working directory is already set: use relative paths only. Never use `/home/work` or inspect any parent, root, or other filesystem path.

User Message:
You are improving `candidate.py`, a controller for an offline mathematical-reasoning replay environment. Use one continuous Agent session to inspect the supplied search evidence and then implement the policy improvement directly in `candidate.py`. Work only inside the current workspace. Read `POLICY_API.md`, `SEED_POLICY.py`, `POLICY_BASE.py`, `feedback.json`, `history.json`, `TRACE_GUIDE.md`, `TRACE_SUMMARY.json`, and `STRATEGY_GUIDE.md`; in personalized runs also read `REQUIREMENT_PROFILES.json`. `STRATEGY_GUIDE.md` contains cross-profile optimization experience — proven patterns and known failure modes from hundreds of completed runs; read it first to avoid common mistakes. `CANDIDATE_HISTORY/` contains proposal code from completed rounds, and `TRACE_HISTORY/` contains their complete sanitized executed traces. `SEARCH_TRACES.jsonl` is the latest completed-round trace alias. When present, `BASELINE_TRACES.jsonl` is the complete sanitized template-baseline trace. Do not spend tool calls searching outside these documented inputs.

Hard data boundary:
- Never inspect parent directories, backing outputs, private replay caches, held-out datasets, or any filesystem path outside the supplied workspace.
- Never search for question text, gold answers, raw reasoning, raw feedback, source paths, or unexecuted future checkpoints.
- The trace JSONL files contain only anonymized observations and actions actually executed by evaluated policies. They remain read-only.
- Do not load or ‘Read‘ a complete trace JSONL into model context. Use ‘TRACE_ANALYZER.py summary‘ and bounded, paginated ‘cases –offset N –limit M‘ queries for statistical analysis and targeted inspection; use aggregate summaries first.
- Do not add file, network, subprocess, reflection, dynamic import, or external private-attribute access to the candidate. Deterministic helpers defined on ‘CandidatePolicy‘ are allowed. Trace inspection is evidence gathering, not a coverage proof. Inspect the aggregate results, then autonomously choose useful anomalous groups and representative traces. You are not required to claim that every record entered context, report ‘records_analyzed‘, or create an analysis report. After sufficient analysis, edit ‘candidate.py‘ in this same session. Finish only after it contains one valid ‘CandidatePolicy‘; do not merely describe a plan.

Model availability rule:
- ‘POLICY_API.md‘ and runtime ‘env.models‘ are authoritative for the complete model pool available to a policy.
- Models observed in traces only show what the incumbent or historical candidate actually spawned. Never infer that an observed subset, including only Qwen3-4B, is the available model set. Algorithm contract:
- A branch chooses its model in ‘spawn(model)‘ and can never switch models.
- ‘refine(branch_id)‘ is legal only when the current stage is terminal.
- ‘env.requirements‘ is the current user’s explicit accuracy, latency, and weighted-compute hard-requirement profile. Use all three thresholds directly to choose model, width, checkpoints, and refine behavior.
- Personalized discovery has no ‘beta‘. Do not invent or infer a replacement scalar from the three explicit requirements.
- Public observations and action return values are dictionaries. Use bracket access such as ‘state[”branches”]‘ and ‘branch[”branch_id”]‘.
- Maximize the explicit utility in ‘feedback.json‘: the profile-weighted fraction of complete seed batches that simultaneously satisfy all three hard constraints. A proposal is accepted only when this utility strictly exceeds the incumbent utility.

- Latency follows replay cohort semantics: sequential actions add; each '*_many' phase adds the maximum cohort latency. Weighted compute includes think, probe, and feedback tokens weighted by model size.
- Base decisions only on public observations already revealed by executed actions, and keep the controller deterministic for a fixed replay seed. Only 'candidate.py' is writable. Do not edit the API, template/baseline copies, profiles, feedback, history, candidate history, analyzer, summaries, or traces. Do not claim performance absent from the supplied evidence.

Figure 4: **Policy-discovery prompt.** The System Message and User Message are presented together in the single framed prompt box used for one policy-improvement round.

G PUBLIC POLICY API

Public Policy API

CandidatePolicy(config={"requirements": profile}) must expose `solve(self, env)` and return an observed answer or `None`. Personalized runs expose only explicit requirements and resources; legacy beta-grid runs separately receive `beta`.

The policy may use: - `env.models`: the authoritative complete set of model names available in the current episode. Models observed in baseline or historical traces only describe what those policies spawned and must never be used to narrow this set.
- `env.beta`: available only in legacy beta-grid evaluation. Personalized discovery has no beta field in the environment, observation, or policy config.
- `env.requirements`: an isolated copy of the current profile, containing exactly one hard constraint for accuracy, latency, and weighted compute. The canonical `constraints` list is preserved, and each constraint is also available directly by metric name (for example, `env.requirements["latency_s"]["value"]`). It is `None` only in legacy beta evaluation.
- `env.observe()`: current public state with model pool counts, opened branches, and charged cost.
- `env.spawn(model)`: create one model-bound branch and reveal checkpoint 0.
- `env.spawn_many(model, count)`: atomically open a fixed-model cohort.
- `env.continue_branch(branch_id)`: reveal the next checkpoint in the current stage.
- `env.continue_many(branch_ids)`: atomically advance one synchronous checkpoint round.
- `env.refine(branch_id)`: after a terminal stage, charge feedback tokens and enter the next stage without revealing its first checkpoint.
- `env.refine_many(branch_ids)`: atomically refine a survivor cohort after every member is terminal.
- `env.can_afford_complete_candidates(...)`: report only whether a complete configured candidate/window fits the remaining pool and weighted-compute budget.
- `env.prune(branch_id)`: deactivate a branch.
- `env.finish(answer=None)`: terminate using an observed answer, or use deterministic plurality when omitted.

`env.observe()` and every state-changing action return Python dictionaries. Use bracket access for returned fields.

```
state = env.observe()
for branch in state["branches"]:
    if branch["can_continue"]:
        env.continue_branch(branch["branch_id"])
```

```
opened = env.spawn_many(env.models[0], 4)
env.continue_many(
    [branch["branch_id"] for branch in opened if branch["can_continue"]]
)
```

Each public branch dictionary contains `branch_id`, `model`, `stage_index`, `stage_kind`, `checkpoint_index`, `latest_answer`, `answer_history`, `stage_terminal`, `can_continue`, `can_refine`, and `pruned`. No question, gold answer, raw generation, raw feedback, source path, future answer, or future token delta is public. In personalized discovery, `REQUIREMENT_PROFILES.json` is the immutable profile population. Each seed evaluates one complete configured discovery-problem batch for every profile. A batch scores 1 only when all three constraints are satisfied; the discovery utility is the

profile-weighted batch satisfaction rate. There is no compatibility beta in this mode: use the three thresholds directly. `SEED_POLICY.py` contains the adapted AutoTTS baselines `ASCConfigured4D`, `ESCConfigured4D`, and `ParallelProbeSR`. Their standard evaluation uses an explicit fixed `ResourceProfile`; it does not map `beta` to a hand-authored model/width/depth/stage schedule. Allowed imports are `collections`, `math`, `statistics`, `typing`, `policies.base`, and `policies.seeds`. Candidate validation rejects I/O, subprocesses, networking, dynamic imports, reflection, external private attributes, and forbidden data references. A constructor may call `super().__init__(...)`; deterministic private helpers defined on `CandidatePolicy` are also allowed.

Figure 5: **Public policy API.** The public environment contract, state fields, actions, personalized utility definition, and candidate restrictions used by the offline replay controller.

H META-PROMPT FOR STRATEGY GUIDE GENERATION

Strategy Guide Generation from the Discovery Experience Bank

You are a research agent tasked with distilling cross-user procedural knowledge from a historical policy-discovery experience bank. Produce a compact natural-language skill that helps a discovery agent design and revise personalized inference policies for new user profiles.

Objective and scope:
Distill how to search, rather than which historical policy to copy. Improve initial candidate design, diagnosis of evaluation feedback, and subsequent revisions under user-specific accuracy, latency, and compute requirements. Use the original evaluator's joint profile satisfaction rate as the policy-quality objective. Preserve metric directions, aggregation levels, evaluation units, and profile weights. Per-constraint satisfaction rates and margins are diagnostic information, not replacement objectives. Distinguish policy satisfaction from proposal acceptance, preserve high-quality rejected candidates, and do not classify ceiling-score ties as policy failures. Do not modify the evaluator, policy interface, model weights, candidate-selection rule, or downstream discovery protocol. Do not introduce historical-policy retrieval or warm-start initialization. Under a fixed-round protocol, preserve all rounds, including score plateaus. Use only authorized source experience and supplied system specifications; keep original artifacts read-only and do not launch new discovery experiments or unconfigured external model calls.

Inventory, alignment, and scale-adaptive processing:
Inventory authorized profiles, runs, rounds, environments, and evaluation configurations. Recover candidate, incumbent, parent, and feedback relationships from artifact identities, code versions or hashes, logs, and evaluation configurations; equal utility values alone do not verify identity. Do not assume a fixed bank size, balanced strata, complete trajectories, or that feedback stored in a round evaluates that round's proposal. Build a compact case index containing profile requirements, pre-generation feedback, candidate identity, design summary, joint satisfaction, constraint diagnostics, execution signals, and source pointers. When the bank exceeds the context budget, process histories in a reproducible configured order, preserve temporal context, persist intermediate summaries with source pointers, and merge rules across batches. Recompute statistics from unique underlying records, retain legitimate repeated runs, revisit source cases supporting retained rules, and report exact coverage, omissions, exclusions, ordering, batching, and any configured subsampling.

Analyze discovery experience:
Include initial designs, improvements, unchanged outcomes, regressions, and recoveries. Prioritize within-profile comparisons under compatible evaluation conditions. Distinguish satisfaction changes from progress relative to the incumbent, track numerical versus structural changes, and account descriptively for requirements, environments, discovery stage, and incumbent level. Aggregate within profiles before averaging across profiles, preserve dependence across repeated rounds, and do not reconstruct joint satisfaction from marginal rates. Inspect actual code and available execution traces for evidence-bearing cases. Static parsing can identify patterns but does not establish runtime behavior; a named variable, API call, or confidence condition is not proof of execution. Describe simultaneous changes as bundles unless attribution is supported. Treat undetected behavior as unknown, not absent.

Abstract and audit conditional rules:
For every retained rule, establish the chain: observable requirements and prior feedback, diagnostic hypothesis, proposed design or revision, execution and joint-satisfaction outcomes, applicability conditions, and limitations. Search for supporting, ineffective, and contradictory cases. Separate measured outcomes from plausible explanations and label evidence as MECHANISM, OBSERVATION,

HYPOTHESIS, or INCONCLUSIVE. Express each rule with a Rule ID and title, Trigger, Evidence, Action, and Verify. The trigger must be observable during discovery; the evidence must include supporting statistics or mechanisms and source pointers; the action must provide a concrete design choice and fallback ordering; and verification must compare the resulting joint satisfaction with the incumbent. Rules must guide both initial design and later feedback-driven revisions without requiring hidden labels, inaccessible feasibility information, or fixed benchmark identities.

Separate contracts, calibration, and experience:
Organize the generated skill into Environment Contracts, Local Calibration, and Experience-Derived Guidance. Keep interface invariants distinct from empirical preferences. Check units and aggregation levels in every formula; distinguish means from tail quantiles, action costs from episode costs, token counts from checkpoint counts, and loop limits from realized depth. Treat approximate estimates as lower bounds or heuristics rather than feasibility certificates. Allow joint consideration of model, width, depth, and stopping behavior. Handle legal shallow configurations, missing estimates, empty answers, and state refreshes according to the actual interface. Evaluate revisions by joint satisfaction: a change in one constraint is not sufficient by itself to reject a revision that improves the joint objective.

Deliverables and final review:
Produce a self-contained English `STRATEGY_GUIDE.md` within the configured length limit, organized as Environment Contracts, Local Calibration, Conditional Search Heuristics, Decision Flow, and Failure Diagnosis. Produce `SKILL_EVIDENCE.md` using the same rule IDs, with source coverage, definitions, supporting and contradictory cases, artifact locations, limitations, and processing provenance. Do not reproduce the entire bank, include historical answer lookups, force one controller template, or claim improved efficiency or generalization before independent evaluation. After drafting, perform one consolidated review of evidence provenance, arithmetic, units, boundary cases, and consistency across rules, examples, decision flow, and diagnosis tables. Correct unsupported claims without launching effectiveness experiments. Save the reviewed outputs, freeze them, and conclude with a brief coverage and limitation statement.

Figure 6: **Strategy-guide generation meta-prompt.** A provenance-aware procedure for distilling conditional cross-user controller-design knowledge from authorized discovery experience while separating environment contracts, local calibration, and empirical guidance.

I General Methodology from Training-Set Discovery Traces

We distill the training-set discovery traces into a controller-design workflow. The summary separates replay contracts, local calibration, search heuristics, and failure diagnosis so that the design can be applied to new user profiles with profile-specific re-estimation.

Controller Design Methodology from Training-Set Traces

1. Evidence and scope:
Read the strategy guide before inspecting traces. Use aggregate summaries first, then inspect bounded representative cases and anomalous groups. The evidence consists of executed training-set policies; raw questions, gold answers, private caches, and unexecuted checkpoints remain outside the evidence.

2. Calibrate the replay cost model:
Estimate the mean phase latency from the trace summary and estimate tokens per continue action from aggregate action counts. Use the authoritative model weights and the current `env.requirements`. Parallel `spawn_many` and `continue_many` use cohort maxima for latency, whereas weighted compute is summed over all branches. These estimates are heuristic because the scorer uses maximum per-question latency within each seed batch and accounts for action-level variation.

3. Optimize the joint hard-constraint utility:
A seed batch is satisfied only when accuracy, latency, and weighted compute all pass simultaneously. The optimization target is the profile-weighted fraction of satisfied batches, and proposal acceptance is strict improvement over the incumbent. Per-constraint satisfaction rates and margins diagnose failures; joint satisfaction rate (JSR) remains the acceptance criterion.

4. Construct a feasible policy before optimizing quality:
Try several `(width, loop_limit)` combinations at the current model using the calibrated estimates. If none is feasible, move to a lighter model. The default controller structure is parallel initial spawning,

parallel continuation rounds, and deterministic plurality voting over usable final answers. A zero continuation limit is valid when the budget permits only spawn actions.

5. Match depth and width to the bottleneck:
In the observed math-reasoning traces, accuracy is the most common bottleneck. Increase depth when branches remain active at the loop limit and accuracy is binding; reduce depth when resources bind and branches terminate earlier. Width primarily adds voting diversity and should increase only when compute headroom supports it.

6. Treat early stopping and refine as conditional mechanisms:
Use confidence-based early stopping or refinement only when the relevant budget and failure diagnosis justify it. Early stopping trades latency for accuracy, while refinement adds tokens and latency; verify the joint utility after either change.

7. Diagnose failures before changing parameters:
After a failed round, read per-constraint satisfaction rates and margins, then check whether the loop limit was actually reached. Small negative margins call for an incremental change of one or two rounds or branches; large gaps call for recalibration from the new trace summary. Do not assume that fixing the lowest individual satisfaction rate will improve the joint rate, because different seed batches can fail on different constraints.

8. Reallocate only verified headroom:
Latency headroom can support a small increase in continuation depth, and compute headroom can support a small increase in branch width, but both changes must be checked against the other constraints. Reducing depth or width to exploit accuracy headroom is risky because terminal-answer quality can change non-linearly.

9. Generalization and limits:
The guide summarizes 488 failed proposals, 800 observed transitions, and 200 training profiles. These statistics define its empirical scope; preserve determinism, use public observations, and re-estimate costs and joint satisfaction after each meaningful policy change.

Figure 7: **General controller-design methodology.** A training-trace-derived workflow for calibrating replay costs, satisfying explicit accuracy–latency–compute constraints, diagnosing failures, and making conservative policy updates. Empirical statistics are confined to the training-set discovery evidence and should not be interpreted as universal laws.