When Does Mixing Candidate Sources Beat Resampling in Diffusion Language Models?

Anonymous authors
Paper under double-blind review

Abstract

Masked diffusion language models can spend extra generator calls in three ways: resample the same decoder, decode with a different rule, or query a different model. We ask when drawing from several of these sources beats resampling one. The analysis starts from a simple observation: repeated draws solve the prompts a source already handles, so the prompts left open concentrate on its weaknesses. For conditionally independent draws we derive how this residual set evolves and obtain a switching rule: another source gains ground when it succeeds where the current source fails, and a learned verifier returns only part of what it adds. The rule yields a greedy source order for tasks with an exact checker and a gated escalation rule for learned verifiers, both fixed on development data and tested prospectively. On Countdown and CRUXEval-I, mixing decoders and models outperforms resampling the released dVoting decoder by 16.8 to 21.2 points at the same budget, in the direction predicted, some smaller than predicted, and matches or beats the strongest single model given the same number of draws. With learned verifiers on MATH the gain shrinks to 3.1 to 5.5 points, and on GSM8K, where few prompts remain unsolved, it vanishes, as the analysis predicts.

1 Introduction

Masked diffusion language models such as LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025) generate text by iteratively revealing tokens from a masked sequence, and the order in which tokens are committed is itself a decoding choice. This creates a test-time allocation problem: given a fixed budget of generator calls (forward passes of the model), should we spend them on more samples from the same decoder, a different decoder of the same model, or candidates from another model? We call each such option a candidate *source*, a generator paired with a decoding rule. These options can cost much the same while succeeding on different prompts.

Recent diffusion-LM methods generate or combine additional candidates in several ways (Feng et al., 2026; Lee et al., 2026; Olausson et al., 2026; Dang et al., 2026; Bilal et al., 2026; Ichihara et al., 2026; Sun et al., 2026; Yun et al., 2026). Every such method returns one answer chosen from its candidates by a *selector*, a rule or model that scores them without seeing a reference answer. A comparison that changes the source together with the calls spent or the selector does not reveal whether the source created new correct answers, whether the selector chose them, or whether another draw would have done as well; we therefore hold the budget and the selector fixed and vary only the source.

We ask what makes a source worth its generator calls. First, what a source adds changes as draws are spent. Repeated samples preferentially solve the prompts the current source handles well, so the prompts left unresolved concentrate on its weaknesses. An alternative source is worth its calls in proportion to how well it solves what is left, and the best next source need not be the one that was best initially.

Second, a correct candidate helps only if the selector returns it. The *coverage* of a set of candidates is the fraction of prompts on which at least one of them is correct; what a new source adds is the coverage it contributes on prompts that the candidates already drawn miss; and *conversion* is the probability that the selector returns a correct candidate when one is available. A new source can change both, so we evaluate complete policies, which decide which candidates to generate and which one to return. When answers can be checked exactly, as in Countdown, where a proposed arithmetic expression either reaches the target from the given numbers or does not, the selector is an exact

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.

checker and every correct answer a new source adds is returned; a learned verifier, a model trained to score answers, returns only part of it. The two observations predict that mixing sources pays most on checkable tasks and when the added source is strong where the current one fails.

We make three contributions. First, we develop a theory of source value under repeated sampling. For conditionally independent draws, we derive how the prompts a source leaves unsolved evolve as it is resampled: further draws of the same source add less and less, while another source becomes worth more when it succeeds where the current one fails. Together with a model of the verifier's conversion, this predicts when drawing from several sources beats resampling one, from development data alone and before the test budget is spent (Section 4).

Second, we turn the theory into two policies. Every choice they make is fixed on *development* prompts, a set kept separate from the prompts used for evaluation. With an exact checker, a greedy order adds next the source that solves most of the still-unsolved development prompts per call and stops at the first verified answer. With a learned verifier, a gate draws further candidates only when the first one scores low, chooses what to draw by success on the first source's failures rather than by overall accuracy, and returns the top-scoring candidate. Both need no training, and both avoid the usual choice, the source with the best accuracy per call, which the theory shows goes wrong: after a few failed draws the most accurate source is often not the most useful next one.

Third, we test the prediction prospectively. We fix every choice on development data and only then generate the *sealed* evaluation sets: 600 Countdown puzzles and 300 CRUXEval-I problems, which ask for an input that makes a given Python function return a given output. With the same checker and budget, mixing decoders and models outperforms resampling our base source, the adaptive decoder released with dVoting (Feng et al., 2026), by 16.8 to 21.2 points and three released diffusion-LM decoding methods by 12.2 to 18.1; with eight Dream draws available to both, the greedy order over both models matches Dream alone on CRUXEval-I and beats it on Countdown, where the two models are complementary and the theory predicts a gain. With learned verifiers the gains are smaller, 5.5 and 3.1 points over resampling on two sealed MATH sets, and none on sealed GSM8K prompts.

Together, these results give a way to decide, before the test budget is spent, whether a diffusion model's extra calls are better used on more samples, another decoding rule or another model, and they show that the answer depends on what earlier samples left unsolved rather than on each source's accuracy. What is new is how repeated sampling changes the value of each source, and that decoding rules of one diffusion model act as distinct sources.

2 Related work

Diffusion-LM decoding and test-time scaling. Masked diffusion language models allow the order in which tokens are committed to change at inference time. Within a single decode, existing work studies confidence-based or learned unmasking rules (Wu et al., 2026; Jazbec et al., 2026), revision and reward-guided decoding (Mounier & Idehpour, 2025; Jeon et al., 2026; Li et al., 2024a), and step- and block-level allocation (Lu et al., 2026). Across decodes, methods generate or combine several candidates: repeated sampling with an agreement stop (Feng et al., 2026), which resamples one adaptive decoder and is our base source and resampling comparator; heterogeneous schedules and tempered, more diverse sampling (Lee et al., 2026; Olausson et al., 2026); trajectory refinement or regeneration (Dang et al., 2026; Ichihara et al., 2026); verifier-guided search over partial denoising trajectories (Bilal et al., 2026); and combination across trajectories or models (Miles et al., 2026; Sun et al., 2026; Yun et al., 2026). Our question is narrower: under a common call budget, which source to add given what earlier sampling left unsolved, and how much of its value survives selection.

Sampling, verification and adaptive allocation. For autoregressive language models, self-consistency aggregates multiple reasoning paths (Wang et al., 2023); repeated sampling keeps raising coverage, which a verifier turns into solved problems (Brown et al., 2024), at a rate set by the distribution of per-problem success probabilities (Schaeffer et al., 2025). Process or outcome reward models provide learned selection signals where no such check exists (Lightman et al., 2024; Zhang et al., 2025). Our Countdown and CRUXEval-I results sit in the first regime, our GSM8K and MATH results in the second. Other work allocates test-time compute adaptively through search, early stopping, cascades or model routing (Snell et al., 2025; Aggarwal et al., 2023; Li et al., 2024b; Chen et al., 2024; Ong et al., 2025; Dip et al., 2026), mixtures of agents combine complementary

models (Wang et al., 2025), and budgeted selection of a subset has greedy guarantees when its value is submodular (Nemhauser et al., 1978; Krause & Golovin, 2014). Our formulation is a specialisation of these familiar settings (best-of-N with a verifier, cascaded inference, and budgeted portfolio selection) to candidate sources that include different decoding rules of one diffusion model.

3 PROBLEM FORMULATION

This section sets up the allocation problem: what a source is, what a policy decides, and the two quantities, coverage and conversion, that determine its accuracy. A model answers a prompt by running a number of *generator calls*, forward passes of the network; the budget is a fixed expected number of such calls per prompt. Each call can go to one of several *sources*, and a policy decides which sources to run and which of the resulting answers to return.

A source s is any way of producing an answer: a model together with a decoding rule, for example a given checkpoint decoded with a fixed number of denoising steps, or the same checkpoint with an adaptive rule that decides at each step how many tokens to commit. A source is random, so it can be drawn repeatedly; we call repeated draws from one source *resampling*. A draw from s on prompt x costs an expected ℓ_s calls.

The candidates a policy has generated form a *pool* S, specified by which sources were run and how many draws were taken from each; $S(x)$ is the set of candidates it yields on prompt x. Let $z(x, y) \in \{0, 1\}$ indicate whether answer y is correct on prompt x. The pool's *coverage*, $C(S) = \Pr[\exists y \in S(x) : z(x, y) = 1]$, is the probability, over prompts and generation, that at least one of its candidates is correct. Throughout, probabilities are over prompts, generation and any randomised decisions unless a conditioning is shown.

Coverage is not accuracy, because one candidate still has to be chosen. The policy returns the candidate with the highest score under a *verifier* $V(x, y)$ that never sees a reference answer, $\hat{y}_S(x) = \arg\max_{y \in S(x)} V(x, y)$, and its *selected accuracy* is $A(S) = \Pr[z(x, \hat{y}_S(x)) = 1]$. Let $\mathcal{R}_S$ denote the event that no candidate in $S(x)$ is correct; then

$$A(S) = C(S)\ \Pr[z(x, \hat{y}_S(x)) = 1 \mid \mathcal{R}_S^c] \qquad \text{whenever } C(S) > 0 : \tag{1}$$

accuracy is coverage times *conversion*, the probability that the verifier returns a correct candidate when one exists. On some tasks a program can check an answer exactly from the prompt alone, for instance whether an arithmetic expression reaches a target with the given numbers, or whether a program passes its tests. Such an *exact checker* is the verifier $V = z$: conversion is one and accuracy equals coverage. A *learned verifier*, a model trained to score answers, converts only part of it.

A policy π uses only the prompt and the verifier scores of the candidates generated so far. Its selected accuracy $A(\pi)$ is the probability that the candidate it returns is correct (a fixed pool S is the policy that always draws S, so $A(\pi) = A(S)$), and its cost $L(\pi)$ is its expected number of calls. The allocation problem at budget B is

$$\max_{\pi\,:\,L(\pi) \le B} A(\pi). \tag{2}$$

Solving this problem exactly would require every source's success on every prompt, so the aim of the paper is the practical version: identify quantities that development data can estimate and that determine which source is worth the next call, build policies from them in Section 4, and test in Section 5 whether choices fixed on development data hold on prompts not seen before.

4 WHAT ONE MORE CALL IS WORTH

We first ask what one more call is worth and how that changes as calls are spent, then turn the answer into policies. Take a pool S and consider one more draw from a source s'. It raises coverage only on prompts the pool has not solved, and that coverage becomes accuracy only if the verifier then returns the new candidate, so the accuracy it adds through new coverage is

$$\underbrace{\Pr(\mathcal{R}_S)}_{\text{unsolved mass}} \times \underbrace{\Pr(I_{s'} = 1 \mid \mathcal{R}_S)}_{\text{rescue rate of } s'} \times \underbrace{\text{conversion}}_{\text{what the verifier returns}}, \tag{3}$$

where $I_{s'}$ indicates that the draw is correct. The first two factors are the coverage gain $\Delta C(s' \mid S)$; with an exact checker the third factor is one. A source is worth its cost when this value per call is large; the second factor is the source's accuracy on the prompts still open, not its overall accuracy. With a learned verifier a draw can also change the choice on prompts already covered, fixing a wrong choice or displacing a right one; Section 4.3 models the loss and Appendix B.5.1 measures both.

4.1 What repeated draws leave behind

The usual way to spend more calls is to draw again from the source in use. Each failed draw is informative: failures fall more often on prompts the source finds hard, so the prompts still open are increasingly the ones it finds hard. To make this precise, let $q_s(x)$ be the probability that a single draw from s solves prompt x; viewed across prompts, q_s is the source's *success profile*, and its average $p_s = \mathbb{E}_x[q_s(x)]$ is the usual one-draw accuracy. We assume draws are independent given the prompt. After k draws from s, all of which failed (k is the *depth*), three quantities describe what remains:

- the *unsolved mass* $Z_k = \mathbb{E}_x[(1 - q_s(x))^k]$, the probability that k draws all fail;
- the *residual distribution* P_k of the prompts still open, which reweights each prompt by $(1 - q_s(x))^k / Z_k$, so that hard prompts count more as k grows;
- the *rescue rate* $r_k = \mathbb{E}_{P_k}[q_s]$, the chance that one more draw solves a prompt drawn from those still open; $r_0 = p_s$.

How fast the rescue rate falls depends on how unequally hard the open prompts are, measured by the *dispersion* $D_k = \mathrm{Var}_{P_k}(q_s)/(r_k(1 - r_k)) \in [0, 1]$: zero when every open prompt is equally hard, one when each is either always or never solved. Let ΔC_{k+1} be the coverage the $(k+1)$th draw adds.

Proposition 1 (Residual dynamics of repeated sampling)**.** *If draws from s are independent given the prompt, then for every k with $Z_k > 0$ and $0 < r_k < 1$,*

$$Z_{k+1} = Z_k(1 - r_k), \qquad r_{k+1} = r_k(1 - D_k), \qquad \Delta C_{k+1} = Z_k r_k = Z_k - Z_{k+1}. \tag{4}$$

In particular r_k is non-increasing, strictly so unless all open prompts are equally hard, and $1 - Z_k \to \Pr_x[q_s(x) > 0]$ as $k \to \infty$.

Each further draw solves a fraction r_k of what is open, and then the next draw is less likely to succeed by the factor $1 - D_k$. Returns diminish for two reasons, fewer prompts remain open and those that remain are harder, and the second is invisible in the source's accuracy p_s. The decay of Z_k is known (Schaeffer et al., 2025); the recursion for r_k and its consequences are new (Appendix A.2).

A source that is weaker on average can therefore be the stronger next call; the measurements of Section 5.3 show this reversal on GSM8K.

4.2 When to switch source

The same quantities decide when to switch. On the prompts still open after k draws from s, another source s' has rescue rate $r_k(s') = \mathbb{E}_{P_k}[q_{s'}]$ and, by (3), one draw from it adds coverage $Z_k\, r_k(s')$.

Corollary 1 (Switching rule)**.** *Let draws from s and s' be independent given the prompt. After k draws from s, one draw from s' adds more coverage per call than one more draw from s if and only if $r_k(s')/\ell_{s'} > r_k/\ell_s$. As s keeps failing, its own rate falls as $r_{k+1} = r_k(1 - D_k)$, whereas the alternative's rate changes by $r_{k+1}(s') = r_k(s') - \mathrm{Cov}_{P_k}(q_s, q_{s'})/(1 - r_k)$, which can fall, stay or rise.*

So the best next source can change with depth: a source whose successes are concentrated where s fails (negative covariance on the open prompts) gains rescue rate relative to s with every failure of s. After a single draw of s, with $0 < p_s, p_{s'} < 1$ and $\phi_{ss'}$ the correlation of the two draws' correctness indicators, the gain of one draw of s' is

$$\Delta C(s' \mid \{s\}) = p_{s'}(1 - p_s) - \phi_{ss'} \sqrt{p_s(1 - p_s)\, p_{s'}(1 - p_{s'})}\,: \tag{5}$$

the gain rises with the alternative's accuracy and falls with its correlation to the source already drawn. When one source is stronger on every open prompt at no greater cost, the rule never switches, and the right allocation is that source; when the open prompts of one source are the strengths of another, the rule switches early. The experiments meet both cases: on CRUXEval-I one model, Dream, is stronger

on almost every open prompt, and on Countdown the two models are complementary. Whatever the rule says, the gain is scaled by Z_k, so once few prompts remain open, no source adds much.

4.3 What the verifier returns

Sections 4.1 and 4.2 covered the first two factors of (3); this subsection models the third, conversion, which a learned verifier makes smaller than one. Let α be its *pairwise accuracy*, the probability that it scores a correct candidate above an incorrect one on the same prompt, estimated on labelled development candidates. A policy needs a different quantity, the probability that the top-scoring candidate of a whole pool is correct. If the verifier scores candidates independently within a prompt, with independent Gumbel noise and a constant shift for correct candidates, so that pairwise comparisons follow a logistic model, then a pool with $n_+ \geq 1$ correct and $n_- \geq 0$ incorrect candidates returns a correct one with probability

$$\kappa(n_+, n_-; \alpha) = \frac{n_+\alpha}{n_+\alpha + n_-(1-\alpha)}, \tag{6}$$

each correct candidate counting α and each incorrect one $1-\alpha$; $\kappa = 0$ when $n_+ = 0$. An exact checker is the case $\alpha = 1$, where every covered prompt is returned. A learned verifier loses part of the coverage, and loses more as incorrect candidates accumulate, so a policy that adds candidates pays a selection cost that a checker never charges. The model is an approximation: ties, repeated answers and source-dependent scores violate its assumptions; Appendix B.5.2 tests it.

4.4 From values to policies

A policy decides the order of candidates and when to stop; how it can stop depends on the verifier.

Exact checker: an order is enough. The checker certifies an answer, so the policy stops at the first accepted candidate and only needs an order. We build it on development prompts by applying the switching rule to sets: start from an empty pool, add the source instance (one more draw from a source) with the largest coverage gain per call on the prompts the pool leaves open, recompute every gain, and repeat until every instance is placed. The released adaptive decoder conditions each sample on the previous one, so its samples enter only in recorded order; we call such orders *admissible*.

The budget fixes only how far down the order the policy may go, its *depth*: the most accurate depth whose mean development cost stays within B. Development coverage is submodular, so a greedy prefix of cost c (Appendix A.4) has at least a fraction $1 - e^{-c/B}$ of the best coverage at budget B, and on Countdown every prefix of it has the largest development coverage of any admissible set of no greater cost, as Appendix A.4 shows. Resampling one source is itself an admissible order, so the greedy order competes with it.

Learned verifier: a gate. No answer can be certified, so the policy decides from verifier scores whether to generate more candidates. A *one-sample gate* generates and scores one candidate from a base source and generates a further set of candidates, the *escalation set*, only when that score falls below a threshold θ; it returns the verifier's top-scoring candidate. Escalation sets come from a menu fixed in advance: further base samples, another decoding rule of the same model, a second model, or unions of these. The budget sets the threshold: θ is the largest threshold whose mean development cost meets the budget. A threshold is the optimal form of rule that sees only this score when the expected gain per further call is non-increasing in the score.

Which set to escalate to. The gate never observes whether the base candidate was correct, but it escalates (fires) on wrong base candidates at some rate g_0, and that rate is the same whichever set it escalates to, so every alternative's gain is scaled by the same factor. Let $\mathcal{G}$ denote the escalation event, I_s and $I_{s'}$ the correctness indicators of the base and escalation candidates, and $r_1(s')$ the rescue rate of s' on the base source's failures.

Proposition 2 (Gated escalation)**.** *Let the base and escalation draws be independent given the prompt, and let* $g_0 = \Pr(\mathcal{G} \mid I_s = 0)$. *The coverage added by escalating to one draw of* s' *is* $(1-p_s)\big[g_0\, r_1(s') + \mathrm{Cov}(\mathbf{1}_{\mathcal{G}}, I_{s'} \mid I_s = 0)\big]$. *If the escalation decision depends on the prompt and base candidate only through* I_s*, the covariance vanishes and the gain is* $g_0(1-p_s)\, r_1(s')$*; if in*

addition each alternative's cost on escalated prompts equals its overall cost $\ell_{s'}$, ranking alternatives by $r_1(s')/\ell_{s'}$ orders their gain per call at every shared threshold.

This is the switching rule at depth one, seen through a gate. The resulting *residual rule* ranks whole escalation sets rather than single candidates: each set is scored by its members' mean rescue rate on the base source's failures per call on development prompts, an approximation that ignores overlap among members, and when the gate fires the whole chosen set is drawn, its size fixed in advance. Appendix A.6 extends the proposition to sets and drops the cost condition when each set has its own threshold. The rule also excludes ranking sets by their success on escalated prompts, which counts prompts the base candidate had already solved. Both conditions of the proposition are approximations: a gate score also reflects how hard a prompt is, which can make the covariance negative, and costs are constant only for fixed-step decoders; Appendix B.6 measures the covariance error.

Choosing among escalation sets in advance. Comparing the menu's escalation sets before running them needs an estimate of each set's accuracy, which we call the *forecast*. Development candidates give how often the gate escalates, the base candidate's accuracy when it does not, and the coverage of the pools it draws when it does; only conversion is modelled, by (6) at the development α.

5 Experimental results

The experiments test the analysis of Section 4 on sealed instances, with every choice fixed on development data before the results were read. After the settings, Section 5.2 asks whether mixing sources beats resampling and the strongest single source, and whether development data predicted the gain; Section 5.3 asks whether the gain arises as the residual dynamics say; and Section 5.4 asks how much of it a learned verifier keeps.

5.1 Settings

Candidate sources. Our principal generator is LLaDA-8B-Instruct (Nie et al., 2025), whose base source is the entropy-adaptive decoder released with dVoting (Feng et al., 2026), which decides at each step from the predictive entropy how many tokens to commit. We run it as released, with the remasking step that makes each sample depend on the previous one, but record all five samples instead of stopping once their answers agree; where independence matters we add fresh samples. The alternatives are further samples from this decoder; a *fixed-step set* from the same checkpoint (Jazbec et al., 2026), one deterministic decode and four sampled at nonzero temperature, all with a fixed denoising schedule; and three Dream-7B (Ye et al., 2025) candidates from dVoting's adaptive sampler for Dream, each costing somewhat more than an adaptive sample. LLaDA-1.5 replaces LLaDA-8B in one Countdown run and is the base generator on MATH (configurations in Appendix C).

Tasks and verifiers. Three tasks have exact checkers that use only the instance. Countdown asks for an arithmetic expression that reaches a target from given numbers, with instances in the format of Zhao et al. (2025). CRUXEval-I (Gu et al., 2024) gives a short Python function and an output and asks for an input that produces it; we use its official prompt and grader and, fixed before any model was run, only the problems that no trivial input solves. On GSM8K and MATH the verifier is the frozen process reward model (PRM) Qwen2.5-Math-PRM-7B (Zhang et al., 2025); a second frozen reward model, AceMath-7B-RM, checks robustness to the verifier, and Section 5.4 fine-tunes it.

Policies, comparators and budgets. With an exact checker every policy draws candidates in a fixed order and returns the first the checker accepts. The *mixed-source* policy uses the greedy order of Section 4.4. Its comparators are *resampling* the base decoder (its five samples, followed on Countdown by further independent samples); the *best single source*, chosen at each budget on development instances among the base decoder, Dream and the fixed-step set; and released HEX (Lee et al., 2026), Two Temperatures (Olausson et al., 2026) and Fast-dLLM (Wu et al., 2026), with their released settings and the same checker. Budgets are mean calls per instance, 250 to 1,100 on Countdown and 250 to 700 on CRUXEval-I. With the learned verifier the policies are gated as in Section 4.4, with a menu of escalation sets evaluated at four budgets; the comparators are resampling and the *accuracy rule*, the usual choice of the escalation set whose members have the best development accuracy per call.

Prospective protocol. We call an evaluation set *sealed* when no outcome on it had been seen before the results were read. For every sealed test all choices were fixed on development instances before the sealed instances existed, and the prediction was recorded: for the exact-checker tests the development estimate of each difference, for the learned-verifier tests the chosen policy and the direction of each comparison; Appendix C.5 states how every prediction is computed. Each sealed set was read once for its recorded comparisons; analyses labelled post hoc reuse that read, and the complete-menu comparison of Section 5.2 is a labelled second read. Budgets count generator forward passes, excluding verifier computation (Appendix C.6); comparisons are paired on the same prompts with 95% prompt-bootstrap intervals, and a difference is *resolved* when its interval excludes zero.

5.2 What an extra source is worth, predicted and observed

Figure 1 shows, for every sealed test, the mixed-source policy's observed gain over each comparator (filled) beside the predicted gain (hollow), averaged over budgets.

First, spending extra calls on other sources beats resampling the base decoder on every task except GSM8K: by 16.8 to 21.2 points with an exact checker and by 3.1 and 5.5 with the learned verifier on MATH; on GSM8K, where the base decoder already solves most prompts, the difference is unresolved. Extra calls are better spent on another source, most clearly when answers can be checked. Second, the exact-checker gains were predicted: every one has the sign recorded in advance, three of the six lie inside the 95% interval of the development estimate, and the other three (LLaDA-1.5 against both comparators, CRUXEval-I against resampling) fall below it, real gains smaller than forecast. Development data thus say in advance whether mixing pays, if not by how much. Third, released HEX, Two Temperatures and Fast-dLLM trail the mixed policy on every benchmark where they were run, by 12.2 to 18.1 points with an exact checker (Table 1), even though their candidate counts are chosen on the evaluation instances, and on MATH released dVoting, with its own stopping and voting, trails our gated policy by 12.2 points [+8.5, +15.8]; the gain is not due to a weak comparator.

The right-hand panel needs one qualification: its single-source comparator has only three Dream draws and cannot spend the higher budgets. We therefore generated five further Dream draws for every instance, rebuilt the greedy order from this complete menu on development data, and read the sealed sets a second time (Table 1; Appendix B.1). With eight draws available to both, the order beats Dream alone by +8.5 [+5.7, +11.2] and +8.2 [+5.5, +11.0] points on Countdown, as predicted, and shows no resolved difference on CRUXEval-I (+0.7 [-0.5, +1.9]), where Dream is the stronger model and the switching rule says so: the order begins with Dream's draws and passes it only once they are spent. The Countdown gain thus comes from mixing, not from starving Dream of draws.

5.3 Where the value comes from

Section 4.1 predicts where the gain comes from: repeated draws solve what a source handles well and leave what it does not, so an alternative is judged on the leftovers. To measure this we drew 33 independent adaptive and 32 Dream candidates and a fixed-step set on 300 sealed prompts each of GSM8K and MATH. Disjoint halves of the adaptive draws determine whether the first k draws failed and estimate the next draw, giving each source's rescue rate after k failures.

The ordering reverses (Figure 2, left). Unconditionally an adaptive draw is more accurate than Dream on GSM8K (81.4% against 78.4%), but on prompts five adaptive draws missed the rates are 11.7% and 28.3%, and per 100 calls on the open prompts Dream's advantage over another adaptive draw moves from -29.1 points at $k = 0$ to +12.0 [+3.0, +22.5] at $k = 5$. MATH shows the same growth without a reversal, as Dream starts ahead. Ranking by overall accuracy thus picks the wrong next source once a few draws have failed.

The advantage belongs to Dream's generator–decoder pair rather than the model alone: on the GSM8K prompts the base candidate gets wrong, Dream rescues +22.0 [+11.4, +32.7] points more than a further adaptive LLaDA sample when each uses its adaptive decoder, but +0.4 [-8.0, +8.6] when both use the same fixed-step schedule, so a source must pair a generator with a decoding rule.

The predictions of Section 4.1 hold in part: Equation (5) predicts the sign of an alternative decoder's advantage over resampling after one base draw in 9 of 10 settings. The covariance of Corollary 1, estimated on disjoint draws, ranks the observed changes of the per-call advantage with depth with rank correlation 0.64 [0.47, 0.84], but fails the stricter criterion fixed in advance, matching signs in at

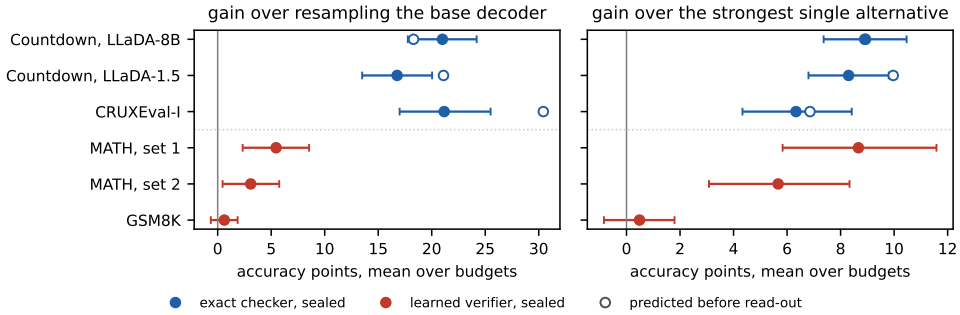

Figure 1: Gain in accuracy points of the mixed-source policy over resampling the base decoder (left) and over the strongest single alternative (right), averaged over budgets, on every sealed test of Table 1. Filled: observed, with 95% intervals. Hollow: the development estimate recorded before the results were read (exact-checker tests only; the learned-verifier protocol recorded the chosen policy, not a gain). The strongest single alternative is the best single source with an exact checker and the accuracy rule with the learned verifier.

		vs resampling		vs best single source		vs Dream ×8 (complete menu)		released methods trail by		
sealed test	n	observed	pred.	observed	pred.	observed	pred.	HEX	TT	FD
Countdown, LLaDA-8B	600	+21.0 [+17.8, +24.2]	+18.3	+8.9 [+7.4, +10.5]	+8.9	+8.5 [+5.7, +11.2]	+9.3	+17.9	+13.2	+13.2
Countdown, LLaDA-1.5	600	+16.8 [+13.5, +20.0]	+21.1	+8.3 [+6.8, +9.8]	+10.0	+8.2 [+5.5, +11.0]	+10.6	–	–	–
CRUXEval-I	300	+21.2 [+17.0, +25.5]	+30.4	+6.3 [+4.3, +8.4]	+6.9	+0.7 [−0.5, +1.9]	+1.7	+18.1	+12.2	+14.5
MATH, set 1	600	+5.5 [+2.3, +8.5]	–	+8.7 [+5.8, +11.6]	–	–	–	+13.6	+7.1	+6.0
MATH, set 2	600	+3.1 [+0.5, +5.8]	–	+5.7 [+3.1, +8.3]	–	–	–	–	–	–
GSM8K	1,128	+0.6 [−0.6, +1.9]	–	+0.5 [−0.8, +1.8]	–	–	–	–	–	–

Table 1: The numbers behind Figure 1; pred. is the development estimate, TT Two Temperatures and FD Fast-dLLM. Released methods keep their decoding settings and are scored by the same checker (Countdown, CRUXEval-I) or selected by the same PRM at matched calls (MATH); their number of candidates is chosen on the evaluation instances, which favours them. The complete-menu columns are a second read of the sealed sets after five further Dream draws were added to every menu. Per-budget accuracies are in Appendix B.1.

least 70% of cells (it reaches 58%); see Appendices B.3.2 and B.4. The theory predicts directions better than small shifts.

A changed ranking is worth little once sampling has shrunk the unsolved mass (Figure 2, right). Only 9.9% of GSM8K prompts survive two failures and 5.0% survive five, so the coverage a further draw adds falls from 104 to 0.6 points per 100 calls between $k = 0$ and $k = 6$; on Countdown, where the best single source solves under half of the instances at moderate budgets, the unsolved mass stays large, which is why mixing pays there and not on GSM8K.

The switching rule does not need a second model: in a post-hoc analysis, ordering only LLaDA's own two decoders by the same greedy construction beats resampling the adaptive decoder on every exact-checker set, by 16.7 [+13.6, +19.8] and 13.1 [+10.0, +16.2] points on Countdown with LLaDA-8B and LLaDA-1.5 respectively, and 6.7 [+3.5, +10.1] on CRUXEval-I. Adding Dream then contributes 4.3 and 3.7 points on Countdown and 14.5 on CRUXEval-I, where it is the stronger model. LLaDA's decoding rules give most of the gain on Countdown, and Dream most of it on CRUXEval-I.

5.4 What the verifier keeps

With a learned verifier part of the added coverage is lost. Selected accuracy falls further below coverage as pools grow, by up to 8.0 points on MATH for the gated policies, and (6), with α estimated on development candidates, tracks this loss to within a few points (Appendix B.5.2). The loss depends on the source: on MATH the PRM recovers 46% of the coverage the fixed-step set adds against 75% of Dream's, because it ranks correct fixed-step candidates less reliably.

Changing only the selector on identical pools isolates this effect: on the sealed Countdown instances, the mixed policy's pools gain +21.0 points over resampling's when the checker selects and +13.3 [+10.1, +16.6] when the PRM selects: the verifier gives up 7.7 [+5.8, +9.6] points of the gain, and the conversion model at the development α predicts +12.3 for the PRM-selected gain. On the two sealed MATH sets, where the residual rule escalates to Dream, the PRM keeps most of the gain an exact answer check would give (+5.5 of +5.7 and +3.1 of +4.3 points, a loss unresolved on both sets; the model predicts +5.4 and +4.0), so there the gain is limited by the coverage mixing adds.

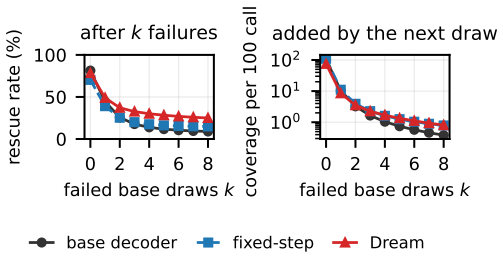

Figure 2: Value of the next draw on GSM8K. **Left:** probability that each source solves a prompt left unresolved by k base-decoder draws. **Right:** the corresponding unconditional coverage per 100 generator calls, a rate, so values above 100 are possible.

The same accounting explains the GSM8K null: on the prompts the base candidate gets wrong, a LLaDA-8B fixed-step candidate is correct 32.8% of the time and a further adaptive sample 32.7%, so the second decoder adds no rescue per candidate, only a lower cost. A prospective comparison on 1,016 further sealed GSM8K prompts finds no resolved gain for adding fixed-step candidates over resampling with its threshold tuned on development data (+0.7 [−0.4, +1.8] points; Appendix B.7).

Under a gate, the residual rule of Proposition 2 escalates to the set that solves most of the base source's development failures per call. Against the accuracy rule, it gains +8.7 [+5.8, +11.6] points of selected accuracy on 600 sealed MATH problems and +5.7 [+3.1, +8.3] on a second sealed set of 600; ranking by success on escalated prompts, which the proposition rules out, trails it on both sets. On 1,128 sealed GSM8K prompts, where the development choice was close, its advantage over the accuracy rule is unresolved (Appendix B.6). Escalation should follow base-source failures, not overall accuracy.

Changing only the selector recovers part of what the PRM fails to convert: AceMath-7B-RM fine-tuned on 53,000 labelled candidates from disjoint prompts (Appendix B.8) selects better than the PRM, and with it selecting for both the residual rule and resampling, the residual rule still exceeds resampling on the two MATH sets by +6.3 [+3.4, +9.2] and +3.2 [+0.5, +6.0] points, against +5.5 and +3.1 with the PRM. Changing source choice and selector together, the policy the forecast chooses reaches 64.9% and 63.2%, against 55.2% and 56.2% for resampling with the PRM. Better sources and a better selector thus add up rather than substitute for each other.

The forecast of Section 4.4 chooses gated policies as well as direct development tuning does on GSM8K, MATH and Countdown, and worse on MBPP+, whose visible tests are an incomplete checker: 2.5 points below the best menu entry against 0.9 for tuning. The forecast's added value is that it says whether a policy is limited by coverage or by selection.

6 Discussion

This paper asked when drawing from several candidate sources beats resampling one. The answer rests on the residual dynamics of repeated sampling: draws solve the prompts a source already handles, the open prompts concentrate on its weaknesses, and an alternative is worth its calls to the extent that it solves those prompts and the selector returns its answers. Where this reasoning predicts a gain, the gain appears: with an exact checker, mixing decoders and models gains 16.8 to 21.2 points over resampling at equal calls, as predicted in direction, and matches or beats the strongest single model given the same number of draws. Where it predicts little, little appears: a learned verifier keeps only the part of the gain the conversion model estimates, and on GSM8K, where few prompts remain unsolved, no source adds much.

Within the scope of the experiments, the predictions concern prompts exchangeable with the development instances and the recorded candidates of three models. The residual analysis assumes conditionally independent draws, hence fresh samples, and the covariance condition of Corollary 1 is checked only on further draws of the same GSM8K prompts. Budgets count forward passes, not FLOPs or latency, and the released methods run outside their native setups.

The same analysis points to what should come next. Under a learned verifier, selection rather than coverage now limits the gain, so training verifiers to convert the added coverage is the natural continuation. A switching rule updated from the failures observed at test time would extend the analysis to prompts without a development set.

Reproducibility statement

Section 5.1 and Appendix C describe the data splits, candidate sources, verifiers, policy menus, budgets and pre-specified protocols; Appendix A gives complete proofs; and Appendix C.8 describes the stored candidates, the scripts that replay every policy and compute the reported numbers, the compute used and the licences of all models and datasets. The code is in the supplementary material; the stored candidates will be released with it.

Use of generative AI

We designed and conducted the research, and we specified, reviewed and tested all AI-assisted work; we take full responsibility for the content of the paper, including its theory, claims, text and code. We used generative AI tools to write the code for the experiments and analyses; to give feedback on the interpretation of results; to assist with literature searches; and to improve the presentation of the text. We did not use them to develop theoretical models or conceptual frameworks, to generate synthetic data, for translation, or to clean or reformat datasets.

References

Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. Let's sample step by step: Adaptive-Consistency for efficient reasoning and coding with LLMs. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. arXiv:2305.11860.

Ahsan Bilal, Muhammad Ahmed Mohsin, Muhammad Umer, Asad Aali, Muhammad Usman Khanzada, Muhammad Usman Rafique, Zihao He, Emily Fox, and Dean F. Hougen. S^3: Stratified scaling search for test-time in diffusion language models. *arXiv preprint arXiv:2604.06260*, 2026.

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.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024. arXiv:2305.05176.

Meihua Dang, Jiaqi Han, Minkai Xu, Kai Xu, Akash Srivastava, and Stefano Ermon. Inference-time scaling of diffusion language models via trajectory refinement. In *Conference on Language Modeling (COLM)*, 2026. arXiv:2507.08390v5.

Sajib Acharjee Dip, Dawei Zhou, and Liqing Zhang. Think again or think longer? selective verification for budget-aware reasoning. *arXiv preprint arXiv:2606.19808*, 2026.

Sicheng Feng, Zigeng Chen, Xinyin Ma, Gongfan Fang, and Xinchao Wang. dVoting: Fast voting for dLLMs. *arXiv preprint arXiv:2602.12153*, 2026.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I. Wang. CRUXEval: A benchmark of code reasoning, understanding and execution. In *International Conference on Machine Learning (ICML)*, 2024. arXiv:2401.03065.

Yuki Ichihara, Naoto Iwase, Mohammad Atif Quamar, and Junpei Komiyama. Prefix-Denoising consistency: Test-time verification for diffusion language models. *arXiv preprint arXiv:2608.25311*, 2026.

Metod Jazbec, Theo X. Olausson, Louis Béthune, Pierre Ablin, Michael Kirchhof, João Monteiro, Victor Turrisi, Jason Ramapuram, and Marco Cuturi. Learning unmasking policies for diffusion language models. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2512.09106.

Kijung Jeon, Thuy-Duong Vuong, and Molei Tao. VGB for masked diffusion model: Efficient test-time scaling for reward satisfaction and sample editing. *arXiv preprint arXiv:2606.28301*, 2026.

Andreas Krause and Daniel Golovin. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*, pp. 71–104. Cambridge University Press, 2014.

Jihoon Lee, Hoyeon Moon, Kevin Zhai, Arun Kumar Chithanar, Anit Kumar Sahu, Soummya Kar, Chul Lee, Souradip Chakraborty, and Amrit Singh Bedi. Test-time scaling in diffusion LLMs via hidden Semi-Autoregressive experts. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2510.05040.

Xiner Li, Yulai Zhao, Chenyu Wang, Gabriele Scalia, Gokcen Eraslan, Surag Nair, Tommaso Biancalani, Shuiwang Ji, Aviv Regev, Sergey Levine, and Masatoshi Uehara. Derivative-free guidance in continuous and discrete diffusion models with soft value-based decoding. *arXiv preprint arXiv:2408.08252*, 2024a.

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 (ICLR)*, 2024b. arXiv:2401.10480.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri 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 (ICLR)*, 2024. arXiv:2305.20050.

Yi Lu, Deyang Kong, Jianing Wang, Linsen Guo, Xue Wang, Qi Guo, Tao Gui, Xuanjing Huang, Wei Ye, Shikun Zhang, and Wei Wang. Adaptive test-time compute allocation for block diffusion language models in complex reasoning. *arXiv preprint arXiv:2602.09555*, 2026.

Roy Miles, Aysim Toker, Andreea-Maria Oncescu, Songcen Xu, Jiankang Deng, and Ismail Elezi. Test-time scaling with diffusion language models via reward-guided stitching. *arXiv preprint arXiv:2602.22871*, 2026.

Nikita Mounier and Parsa Idehpour. Review, Remask, Refine (R3): Process-guided block diffusion for text generation. *arXiv preprint arXiv:2507.08018*, 2025.

George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14:265–294, 1978.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2502.09992.

Theo X. Olausson, Metod Jazbec, Xi Wang, Armando Solar-Lezama, Christian A. Naesseth, Stephan Mandt, and Eric Nalisnick. A tale of Two Temperatures: Simple, efficient, and diverse sampling from diffusion language models. In *Conference on Language Modeling (COLM)*, 2026. arXiv:2604.09921.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs with preference data. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2406.18665.

Rylan Schaeffer, Joshua Kazdan, John Hughes, Jordan Juravsky, Sara Price, Aengus Lynch, Erik Jones, Robert Kirk, Azalia Mirhoseini, and Sanmi Koyejo. How do large language monkeys get their power (laws)? In *International Conference on Machine Learning (ICML)*, 2025. arXiv:2502.17578.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2408.03314.

Zhenhong Sun, Hanqing Zhao, Yatao Bian, Rongcheng Tu, Liuyue Xie, Xu Zhang, Jue Wang, Davide Modolo, Daoyi Dong, and Dacheng Tao. Escaping confidence trap: Evolutionary decoding for mathematical reasoning in diffusion LLMs. *arXiv preprint arXiv:2608.00605*, 2026.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-Agents enhances large language model capabilities. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2406.04692.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency improves Chain of Thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2203.11171.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dLLM: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2505.22618.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

Heecheol Yun, Joonhyung Park, Joowon Kim, and Eunho Yang. Who should lead decoding now? tracking reliable trajectories for ensembling masked diffusion language models. *arXiv preprint arXiv:2606.16281*, 2026.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025. arXiv:2501.07301.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2504.12216.