ON THE EFFECT OF SAMPLING DIVERSITY IN SCALING LLM INFERENCE

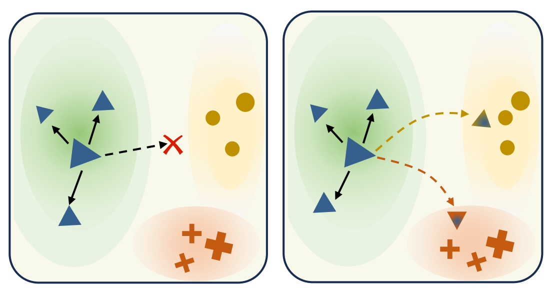
Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) scaling inference is key to unlocking greater performance, and leveraging diversity has proven an effective way to enhance it. Motivated by the observed relationship between solution accuracy and meaningful response diversity, we systematically study the effect of prompt diversity in scaling inference. We theoretically explain why diversified sampling improves Best-of-N scaling, showing that responses generated from meaningful diverse prompts after Best-of-N selection exhibit significantly lower error rates than those produced from stationary prompts. To promote solution diversity, we analyze perturbation fidelity and show that moderately relevant perturbations improve performance, providing guidance for effective perturbation design. Further, we present a set of effective perturbations, including task-level and query-level ones, and analyze the conditions under which they succeed. We systematically evaluate diversified sampling across tasks, finding relative gains of 10.8% in EM@100 for reasoning, 9.6% for mathematics, and 9.5% in Pass@100 for code generation.

1 INTRODUCTION

Large language models (LLMs) have shown impressive performance across diverse tasks. As their capabilities grow, studying and improving their inference processes becomes increasingly crucial. LLM scaling inference is known to exhibit non-determinism, with variability arising from stochastic decoding, floating-point precision limits, and system-level concurrency (Yuan et al., 2025; Atil et al., 2024). Recent work has sought to eliminate this variability: (Yuan et al., 2025) traced accuracy drops to precision-induced rounding differences and advocated higher-precision inference, while (He et al., 2025) introduced batch-invariant kernels to stabilize GPU scheduling. However, such nondeterminism can be beneficial for test-time scaling (Wang et al., 2023b; Li et al., 2025; 2023; Mu et al., 2024; Naik et al., 2023; Zeng et al., 2024; Wu et al., 2024; Nori et al., 2024; Snell et al., 2024; Brown et al., 2024; Gandhi et al., 2024; Snell et al., 2025; Lee et al., 2025; Wang et al., 2025), where model parameters remain fixed, and performance improvements must therefore arise from encouraging diverse and thus non-deterministic exploration. Previous literature has investigated temperature scaling as a means to increase output diversity and, in turn, improve downstream task performance (Zhang et al., 2024b; Holtzman et al., 2019), while other studies (Li et al., 2023; Mu et al., 2024; Naik et al., 2023) have focused on designing diversified prompts to improve inference accuracy in reasoning tasks, eliciting varied chain-of-thought intermediate steps that guide the model's reasoning process toward convergent answers through consensus.



(a) Direct Sampling (b) Diversified Sampling

Figure 1: A brief sketch of (a) direct sampling without diversification and (b) diversified sampling.

However, Best-of-N sampling (Cobbe et al., 2021; Lightman et al., 2023), as a typical form of test-time scaling, aims to maximize the utility of pre-trained models by efficiently exploring multiple responses and selecting the most accurate one. In such cases, sampling solutions from an LLM

Strategies	Pass@100	tf-idf sim.	BERT sim.	lev. sim.	seq. sim.
None	0.8212	0.2152	0.9997	0.2219	0.2244
Role	0.8262	0.2063	0.9996	0.2161	0.2208
Instruction	0.8992	0.1625	0.9968	0.1767	0.1884
Jabberwocky	0.8161	0.2173	0.9997	0.2267	0.2285

Table 1: **Effects of different injection strategies.** 100 solutions were generated using `gpt-4o-mini` (OpenAI, 2023b) for each strategy on the MBPP benchmark.

using the same prompt often leads to similar outputs, "trapped" within a local cluster (Figure 1(a)). The concentrated nature of the trapped solutions might stem from the limited diversity imposed by post-training objectives, which are typically designed to optimize zero-shot performance and align LLM as instruction-following chatbots (Xiang et al., 2025). These objectives often prioritize optimizing the model to produce a single, correct answer, which mismatches with the goal of repeated sampling. The commonly used distillation technique may also diminish model diversity (Cideron et al., 2024; DeepSeek-AI et al., 2025). Diverse candidate solutions should span multiple clusters, with responses distributed across a broader solution space, breaking out of local clusters (Figure 1(b)). An intuitive strategy is to introduce diversity at the prompt level, which in turn broadens the exploration space. We conducted fundamental empirical studies within the Best-of-N framework by applying diversified prompt perturbations, as shown in Table 1. The diversity strategies employed to promote varied responses include *Role* and *Instruction* prompt perturbations, described in Section 4.2 as task-level approaches. These are compared against direct sampling (*None*) and *Jabberwocky*, the latter introducing a segment of poetry as random textual noise. The **Pass@k** rate measures the proportion of correct solutions among k generated attempts in code generation tasks, whereas **tf-idf**, **BERT**, **Levenshtein**, and **token sequence** measure the similarity between solutions (see Appendix C for details). Table 1 shows that the pass rate improves when perturbation strategies produce candidate solutions with reduced similarity. *This observation motivates us to investigate the effect of exploration diversity on scaling inference.*

In this paper, we systematically study the effect of diversified sampling. We first ❶ provide a theoretical perspective showing why general exploration diversity improves best-of-N performance. We then ❷ examine the effect of perturbation fidelity, ❸ outline several perturbation styles, and ❹ study their behavior under different conditions. Extensive evaluations on reasoning, mathematics, and code generation show that the perturbations yeilds relative gains of 10.8% in EM@100 for reasoning, 9.6% for mathematics, and 9.5% in Pass@100 for code generation compared to direct sampling. In summary, we address our contributions as follows:

- ★ We theoretically show why exploration diversity, possibly achieved through perturbation-induced prompt variation, can improve Best-of-N performance. Our theories predict that sampling attempts from a policy richer in solver mode diversity can improve selection performance, achieving faster convergence and lower asymptote.
- ★ We analyzed perturbation fidelity and found that moderately relevant perturbations improve performance, while overly similar or irrelevant ones yield no performance gains and can even lead to degradation. Guided by this, we outline several potentially effective perturbation styles.
- ★ We analyzed the conditions under which diversified exploration remains effective, finding that it works across temperatures, with CoT steps, and under LLM-as-Judge verification, but its benefits depend on thinker model strength, perturbation cardinality, and may vanish under majority-voting.
- ★ We systematically evaluate the effectiveness of diversified sampling across reasoning, math, and code-generation tasks. We find that some of the task-level and query-level perturbations, as well as their combination, consistently improve performance across these tasks.

2 PRELIMINARIES

Task Description. We consider sets of tasks defined by a tuple $\langle \boldsymbol{p}, \mathcal{Q}, V \rangle$ of an instruction prompt $\boldsymbol{p}$, a distribution $\mathcal{Q}$ over the question set and a verifier V. For a solver of the task, the **prompt** $\boldsymbol{p}$ and a **question** $\boldsymbol{q}$ sampled from the distribution $\mathcal{Q}(\cdot)$ are given, from which the solver predicts an **answer** $\boldsymbol{s}$. This answer is finally judged by the **verifier** $V(\boldsymbol{s}|\boldsymbol{p}, \boldsymbol{q})$, which assigns 1 to accepted answers and 0 to rejected answers. In **reasoning** and **math** tasks, the prompt $\boldsymbol{p}$ asks the solver to choose answer $\boldsymbol{s}$ from an answer set $\boldsymbol{A}$ for some question $\boldsymbol{q} \sim \mathcal{Q}$, and the verifier V checks if the answer exactly matches the hidden ground truth. In **code generation** task, the solver is given a prompt and object pair $\langle \boldsymbol{p}, \boldsymbol{o} \rangle$ in natural language with $\boldsymbol{o} \sim \mathcal{Q}$, which asks the solver to write code for some object $\boldsymbol{o}$. The objective is to implement $\boldsymbol{o}$ such that it passes all hidden tests evaluating its correctness. A solution s' is deemed correct by the verifier V if it successfully passes all hidden tests.

Best-of-N sampling. Repeatedly sampling i.i.d. responses $[\boldsymbol{s}]_N := [\boldsymbol{s}_1, \boldsymbol{s}_2, ..., \boldsymbol{s}_N] \sim \mathrm{LLM}(\cdot|\boldsymbol{p}, \boldsymbol{q})$ given prompt $\boldsymbol{p}$ and question $\boldsymbol{q}$ from the LLM solver. For reasoning and math tasks, a task is considered to be solved if at least one response exactly matches the ground truth (Wang et al., 2023a); in this case the proportion of tasks that are solved by the LLM solver with k attempts is called the **EM@k rate**. For code generation tasks, a task is solved if at least one attempt passes all hidden tests (this is equivalent to selecting the answer that passes the highest number of validation tests (Chen et al., 2024a)); in this case the proportion of tasks that are solved with k attempts is called the **Pass@k rate** (Chen et al., 2021). More details on evaluation metrics can be found in Appendix C.

3 Why Solution Diversity Improves Performance: A theoretical Perspective

We provide a theoretical perspective on the importance of solution diversity in this section. For a more technically rigid description and the proof of our theorem, please refer to Appendix B.

Setting. We use $\boldsymbol{r} = [\boldsymbol{p}, \boldsymbol{q}]$ to denote concatenated inputs to the LLM. To characterize different sampling strategies, we configure each attempt by a *decoding mode* $\zeta \in \mathcal{Z}$ (e.g., decoding seed/sampler, temperature, reasoning style), and define a *diversity policy* ν as a distribution over modes. Given $\boldsymbol{r}$ and $\zeta \sim \nu$, the *solver* produces $\boldsymbol{s} \sim \mathrm{LLM}(\cdot \mid \boldsymbol{r}, \zeta)$ and a *verifier* V assigns a $0/1$ correctness score. We define the per–mode failure log–probability to be

$$q(\boldsymbol{r}, \zeta) \;:=\; \log \mathbb{P}_{\boldsymbol{s} \sim \mathrm{LLM}(\cdot|\boldsymbol{r}, \zeta)} \big[V(\boldsymbol{s}) = 0 \big]. \tag{1}$$

Hybrid diversity. We factor the diversified attempt mode as $\zeta = (\eta, \xi)$, where $\eta \sim \nu_0$ is the *base* decoding randomness configuration following baseline policy (e.g., a fixed sampling strategy in the simplest case), while $\xi \sim \Pi$ is an *auxiliary* diversity source; we write interchangeably $q(\boldsymbol{r}, \eta, \xi) = q(\boldsymbol{r}, \zeta)$. The goal is to prove this additional diversity boosts performance.

Hypotheses. We posit two hypotheses—*dispersion* and *fidelity*—to characterize the basic properties of diversified sampling, in preparation for our main results.

Hypothesis 3.1 (Variation under auxiliary diversity)**.** Fix an input $\boldsymbol{r}$ for which there exists $\zeta = (\eta, \xi)$ with $q(\boldsymbol{r}, \zeta) \neq 0$. Define the first absolute central moment with respect to varying ξ:

$$M_1(\boldsymbol{r}; \nu_0, \Pi) \;:=\; \mathbb{E}_{\eta \sim \nu_0} \, \mathbb{E}_{\xi \sim \Pi} \, \big| \, q(\boldsymbol{r}, \eta, \xi) \;-\; \mathbb{E}_{\xi \sim \Pi} q(\boldsymbol{r}, \eta, \xi) \, \big|, \tag{2}$$

then there exists a constant $\hat{\mu}_1 > 0$ such that $M_1(\boldsymbol{r}; \nu_0, \Pi) \;\geq\; \hat{\mu}_1$.

Remark 3.2 (Intuition for Hypothesis 3.1). Unless $\boldsymbol{r}$ is truly unsolvable for *all* (η, ξ), varying the auxiliary RNG ξ modifies the success probability even when the base randomness η is held to its usual variability. Averaging over η reflects that we compare attempts under the same baseline stochasticity.

Hypothesis 3.3 (Fidelity with fixed auxiliary RNG)**.** There exist $\epsilon \in [0, 1)$ such that, when fixed, any auxiliary setting $\xi^\star$ used by the diversified policy performs almost as well as the baseline reference policy ξ^0 for Best-of-N pass rate for $N \geq 1$:

$$\frac{\mathbb{E}_{\boldsymbol{r} \sim \mathcal{R}} \, \mathbb{E}_{\eta \sim \nu_0} \big[\exp \big\{ N \, q(\boldsymbol{r}, \eta, \xi^\star) \big\} \big]}{\mathbb{E}_{\boldsymbol{r} \sim \mathcal{R}} \, \mathbb{E}_{\eta \sim \nu_0} \big[\exp \big\{ N \, q(\boldsymbol{r}, \eta, \xi^0) \big\} \big]} \;\leq\; 1 + \epsilon. \tag{3}$$

Remark 3.4 (Intuition for Hypothesis 3.3). Here the auxiliary choice ξ is *fixed* on both sides so that both policies enjoy the same amount of base variability ($\eta \sim \nu_0$). Fidelity requires that switching to a new auxiliary setting $\xi^\star$ does not globally worsen the response quality compared to a reference ξ^0.

Main result. Now we move on to compare the base diversity policy of $(\eta, \xi_0) \sim \nu_0 \times \{\xi_0\}$ and hybrid diversity policy of $(\eta, \xi) \sim \nu = \nu_0 \times \Pi$. We write Best-of-$N$ failure probabilities as

$$P^N_{\mathrm{div}} \;:=\; \mathbb{P}\Big[V(\boldsymbol{s}_k) = 0, \; \forall k \in [N] \; \Big| \; \boldsymbol{s}_k \sim \mathrm{LLM}(\cdot \mid \boldsymbol{r}, \zeta_k), \; \zeta_k \overset{\text{i.i.d.}}{\sim} \nu, \; \boldsymbol{r} \sim \mathcal{R} \Big], \tag{4}$$

$$P^N_{\mathrm{reg}} \;:=\; \mathbb{P}\Big[V(\boldsymbol{s}_k) = 0, \; \forall k \in [N] \; \Big| \; \boldsymbol{s}_k \sim \mathrm{LLM}(\cdot \mid \boldsymbol{r}, \zeta), \; \zeta \sim \nu_0 \times \{\zeta_0\}, \; \boldsymbol{r} \sim \mathcal{R} \Big]. \tag{5}$$

As $N \to \infty$, these converge to limits $P^{\mathrm{inf}}_{\mathrm{div}}$ and $P^{\mathrm{inf}}_{\mathrm{reg}}$, which reflect the fractions of inputs that remain unsolved under the respective policies.

Theorem 3.5 (Diversity improves Best-of-N). *Under Hypotheses 3.1 and 3.3, there exists a positive sequence $C_N = \Omega\big(\hat{\mu}_1^2 N/(1+\epsilon)\big)$, increasing in N, such that*

$$P^N_{\mathrm{div}} \;\le\; \big(P^N_{\mathrm{reg}} - P^{\mathrm{inf}}_{\mathrm{reg}}\big)/(1 + C_N) \;+\; P^{\mathrm{inf}}_{\mathrm{div}}, \qquad \textit{with} \quad P^{\mathrm{inf}}_{\mathrm{div}} \;\le\; P^{\mathrm{inf}}_{\mathrm{reg}}. \tag{6}$$

This theorem implies two distinct advantages of introducing auxiliary diversity: (i) ***Lower asymptote.*** *Diversity shrinks the "blind-spot" set of instances that remain unsolved as N grows. (ii)* ***Faster convergence.*** *The error reduction factor improves with richer (but faithful) diversity, yielding steeper Best-of-N gains.*

4 Towards Encouraging Exploration Diversification

In this section, following our theoretical insights, we study prompt perturbation as a means to enhance solution diversity. We begin by analyzing the relationship between perturbation fidelity and inference effectiveness to identify which perturbations have a positive impact. Building on this guidance, we further outline several empirical designs of perturbations that are likely to be effective.

4.1 Patterns of Perturbation Effects

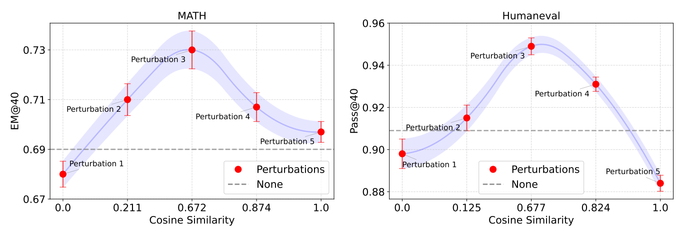


Figure 2: **Effect of perturbation relevance.** Relationship between perturbation-question similarity and task performance. EM rate (math) and Pass rate (code) measured from 40 solutions under five perturbation types (1–5). Results are obtained with `GPT-4o-mini` and are reported as the mean and standard deviation over five independent runs.

To explore potentially effective perturbation patterns, we begin by considering their relationship to the question. Specifically, the model is prompted to generate N solution ideas, which are injected into the original prompt as perturbations. We analyze five perturbation styles that are characterized by varying levels of fidelity to the question: Perturbation 1, ideas entirely unrelated to the task, for example generating baking recipe flavor suggestions even when the task is math or code; Perturbation 2, loosely related ideas that differ in topic from the task domain but still connect to learning or knowledge; Perturbation 3, directly relevant ideas that align with the question itself; Perturbation 4, rephrasings of the original question; and Perturbation 5, verbatim repetition of the original question. The prompts used to obtain these perturbations are provided in Appendix G.1.

We computed the average embedding cosine similarity between these perturbations and the question content using `all-MiniLM-L6-v2` (Reimers & Gurevych, 2019), as well as their EM rate and Pass rate. As shown in Figure 2, we find that EM and Pass rates vary non-monotonically with perturbation relevance. Performance exhibits a rise-then-fall pattern: irrelevant ideas (Perturbation 1) and verbatim repetition (Perturbation 5) fail to improve performance and may even degrade it, while performance increases with higher relevance, peaks with task-aligned ideas (Perturbation 3), and then declines again when similarity becomes excessive.

This empirical tradeoff effect is also a direct consequence from our theory: the first-moment $\hat{\mu}_1$ from Hypothesis 3.1 should be low for highly relevant perturbations, thereby yielding a less significant performance boost; on the other hand, the degradation in response quality measured by ϵ in Hypothesis 3.3 can be large as perturbation relevance decreases, leading to reduced performance. We analyze this in more detail in Appendix B.5.1.

The relationship between perturbation relevance and scaling inference performance is non-linear. Moderately relevant perturbations contribute positively, whereas overly low or excessively high similarity offers no benefit and may even degrade performance.

4.2 Perturbation Design

Section 4.1 tells that only meaningful perturbations are effective. Guided by this, we outline two categories of perturbations: task-level and query-level. Task-level perturbations are task-dependent but independent of specific questions, whereas query-level perturbations directly tied to the questions.

Task-level Perturbations. These perturbations are independent of specific question content and are sampled from a pool of predefined candidates. (1) **Role** injection samples predefined identity-descriptive sentences (e.g., “mentor", “optimizer", “innovator") into prompts (Shanahan et al., 2023; Kong et al., 2024), steering the model to generate outputs aligned with different personas. (2) **Strategical Instruction** injection introduces stepwise guidance or problem-solving heuristics (Zhou et al., 2023b; Cook et al., 2020; Naik et al., 2023), steering the model toward generating logical and contextually aligned outputs. We refer readers to Appendix G.2 for detailed descriptions.

Query-level Perturbations. To obtain more meaningful perturbations, we consider two strategies: (1) **Random Idea Injection** (`RandIdeaInj`), where an LLM (either the target model or another) acts as a thinker to propose task-related ideas which are then injected into the original prompt for perturbation. (2) **Random Query Rephraser** (`RandQReph`) restates the input question (Deng et al., 2023), yielding a modified query $\boldsymbol{q}'_k$ that replaces $\boldsymbol{q}_k$ in $(\boldsymbol{p}, \boldsymbol{q}'_k)$. Rephrasing can also be achieved through back-translation (Beddiar et al., 2021), which produces alternative phrasings while maintaining contextual consistency. Both strategies support three variants: **Single**, where the model itself generates ideas or rephrasings; **Dual**, where a separate model is used; and **Diverse**, where a pool of models each provides varied perturbations in advance, and at each iteration the perturbation is selected from this perturbation set.

5 When Diversified Perturbations are Effective

In this section, we analyze whether the diversified exploration remains effective under different conditions. Specifically, we examine the effects of sampling temperature, thinker model, perturbation cardinality, the presence of Chain-of-Thought reasoning, and the choice of verifier. For each setting, we provide analyses and highlight the corresponding takeaways.

5.1 Varying Sampling Temperatures

Sampling temperature is a widely used decoding parameter that directly controls the randomness of generation, and higher temperatures have been shown to increase output diversity (Holtzman et al., 2019). We investigate the effectiveness of perturbations across varying sampling temperatures. Specifically, we evaluate the task-level perturbation Strategical Instruction and the query-level perturbation Dual under varying temperature settings to assess their performance. Figure 3 shows the results of sweeping the temperature from 0.0 to 1.2 in increments of 0.2 on the Humaneval dataset. Our findings show that perturbations and direct sampling all exhibit improvements at higher temperatures. Instruction attains the highest Pass@100, outperforming the best performance of direct sampling by 3.9%, while Dual’s best Pass@100 exceeds the direct sampling maximum by 1.9%.

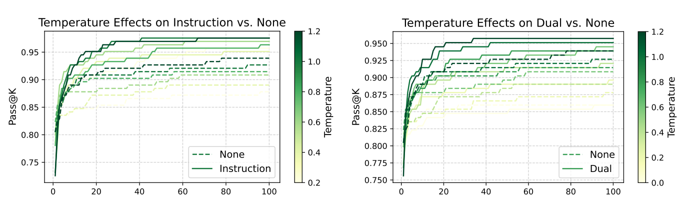


Figure 3: Sweep over temperature in 0.2 increments from 0.0 to 1.2 on Humaneval using `GPT-4o-mini`. Perturbations and direct sampling improve Pass@k at higher temperatures, though the gains plateau with further increases in temperature.

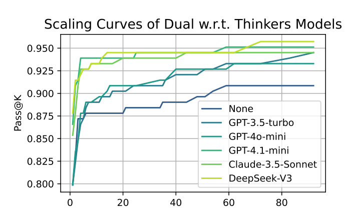


Figure 4: Scaling curves of the Dual strategy across thinker models, with stronger models yielding higher performance.

5.2 Effect of Thinker Models

We analyze the impact of different thinker models in query-level perturbation. Figure 4 presents the evaluation of the Dual strategy with various thinker models while using `GPT-4o-mini` as the generator. The results indicate that stronger thinker models, such as DeepSeek-V3 Liu et al. (2024), raise the scaling curve. We suggest that practitioners choose the thinker model according specific use cases and resource constraints.

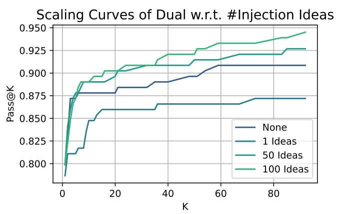


Figure 5: Scaling curves for the Dual strategy as injection cardinality increases.

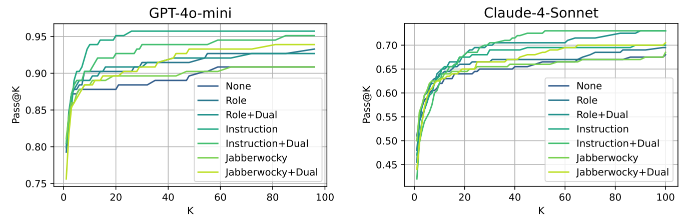


Figure 6: Perturbations with Chain-of-Thought (CoT). *Left*. HumanEval results using `GPT-4o-mini`. *Right*. APPS results using `Claude-4-Sonnet`.

5.3 Effect of Perturbation Cardinality

We analyze the effect of varying the number of perturbations. Figure 5 shows the Dual strategy on HumanEval with 1, 50, and 100 distinct injection ideas, where `GPT-4o-mini` generates the solutions and `GPT-3.5-turbo` serves as the thinker. We find that increasing the number of injection ideas raises the scaling curve, whereas using only a single injection candidate yields a noticeably lower curve. These findings are in line with our theory: the larger the moment $\hat{\mu}_1$ in Assumption 3.1 are, the more significant the improvement of diversified sampling over regular sampling is. We leave more detailed discussions of this relation to Remark B.6 in Appendix B.

While scaling curves improve as sampling temperature increases, diversified perturbations still yield additional gains. Query-level perturbations are influenced by the strength of the thinker model and the richness of perturbation cardinality; scaling performance improves as thinker models become stronger and the number of perturbations increases.

5.4 Chain-of-Thought with Perturbations

Chain-of-Thought (CoT) prompting structures reasoning into explicit intermediate steps, helping LLMs arrive at more accurate answers on complex reasoning (Wei et al., 2022; Wang et al., 2023b). Building on CoT, we introduce perturbations and conduct analyses with `GPT-4o-mini` and `Claude-4-Sonnet`. We evaluate task-level perturbations, Role and Instruction, along with their combinations with the query-level strategy Dual using the `GPT-3.5-turbo` thinker. These are compared against the random perturbation Jabberwocky and direct sampling without perturbations. All perturbations are applied under the Chain-of-Thought (CoT) setting, with prompt details provided in Appendix G.4. Results in Figure 6 show that task-level and query-level perturbations improve performance under CoT, yielding up to a 4.7% relative gain in Pass@100 on HumanEval with `GPT-4o-mini` and a 7.4% relative gain on APPS with `Claude-4-Sonnet`.

Diversified perturbations improves inference performances under the CoT setting.

5.5 Effect of Verification

Although ground-truth reward is commonly used (Zhong et al.; Lewkowycz et al., 2022; Wang et al., 2023c), considering a solution correct if it matches the hidden answer set or passes all hidden tests, outcome reward models (ORMs) offer an alternative by scoring candidate solutions (Zhong et al.; Lightman et al., 2023). However, this approach relies heavily on the quality of the ORM itself. In practice, the ORM's own model performance may introduce biases or errors, and how to train a truly reliable ORM for the verification phase remains an open question that lies outside the scope of our study. To avoid potential interference from external ORMs, one alternative verifier is to use the model itself to evaluate and select the final solution. We evaluate task-level perturbations on MATH and HumanEval using `GPT-4o-mini`. For reasoning and math tasks, the model evaluates and scores each solution, and the top-10 highest-scoring ones can be selected; the instance is marked correct if any of them exactly matches the ground truth (Figure 7(a)). For code generation tasks, the model is prompted to produce 10 unit tests, and a solution is marked correct if it passes at least one of them (Figure 7(b)). The ORM prompt templates are provided in Appendix G.5. The results show that perturbations remain effective when using the LLM-as-a-judge verifier. We also analyze the effectiveness of ORM from our theoretical standpoint: when the ORM has high quality, the ORM pass@k rate is very close to the oracle pass@k rate. See detailed analyses in Appendix B.5.2.

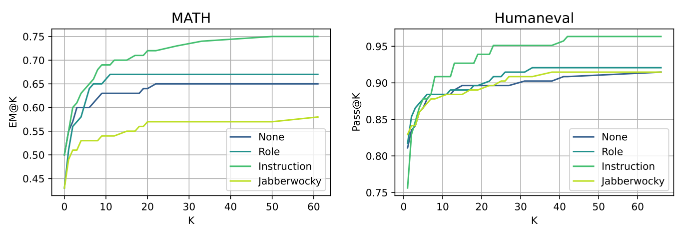


Figure 7: Perturbations are evaluated using the LLM-as-a-Judge verifier. *Left.* Solutions from MATH are rated and selected by the model. *Right.* Solution from Humaneval are evaluated on 10 generated unit tests.

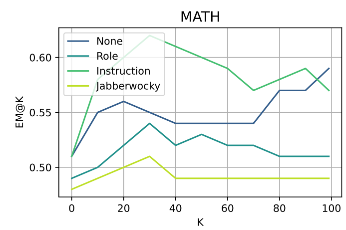


Figure 8: Perturbations are evaluated on MATH, with final answers derived by a majority-voting verifier.

Another verification approach is the majority-voting reward, where the most frequent candidate solution is treated as the final answer. However, this approach is suitable only for tasks such as reasoning and mathematics, where the final answers are well-defined. In the context of code generation, each response is a code snippet that may vary substantially across generations. Even when multiple functions pass the same test set, they can differ in structure, for example by using different function names, variable names, or logical sequences. In such cases, majority-voting reward is rarely used. Figure 8 reports the performance of perturbations under majority-voting reward on the math task. We find that perturbations do not yield consistent improvements compared to direct sampling. The key reason lies in the underlying assumption of the evaluation protocol: for Best-of-N inference without counting majority, diversity is beneficial because it increases the likelihood that at least one correct solution appears among the N attempts, hence greater divergence is desirable. However, under majority-voting reward, performance improves when the majority of solution candidates converge toward the correct solution; thus, solution divergence does not guarantee the benefit. We back this phenomenon theoretically that majority voting does not produce the same performance boost as pass@k in Appendix B.5.3, even degrading performance in the worst case.

Diversified sampling appear effective under the LLM-as-a-Judge setting, yet their benefits may not hold under majority-voting reward as performance gains require convergence toward the correct solution rather than diversity.

6 How Effective Are Diversified Perturbations

In this section, we empirically evaluate how effective are diversified sampling across reasoning, mathematics, and code generation tasks.

6.1 Evaluation Setup

Datasets. We evaluate perturbations across six benchmarks spanning reasoning (**MMLU-Pro** (Wang et al., 2024)), mathematics (**GSM-Hard** (Gao et al., 2023), **MATH** (Hendrycks et al., 2021b)), and code generation (**HumanEval** (Chen et al., 2021), **MBPP** (Austin et al., 2021), **APPS** (Hendrycks et al., 2021a)). Detailed dataset descriptions and setup are provided in Appendix E.1.

Implementing Details. For simplicity, we configured the models with a temperature of 0.6 for all datasets. We did not use nucleus sampling across the experiments. Perturbations are evaluated against direct sampling without perturbation, denoted as **None** in all experiments. **Jabberwocky**, a

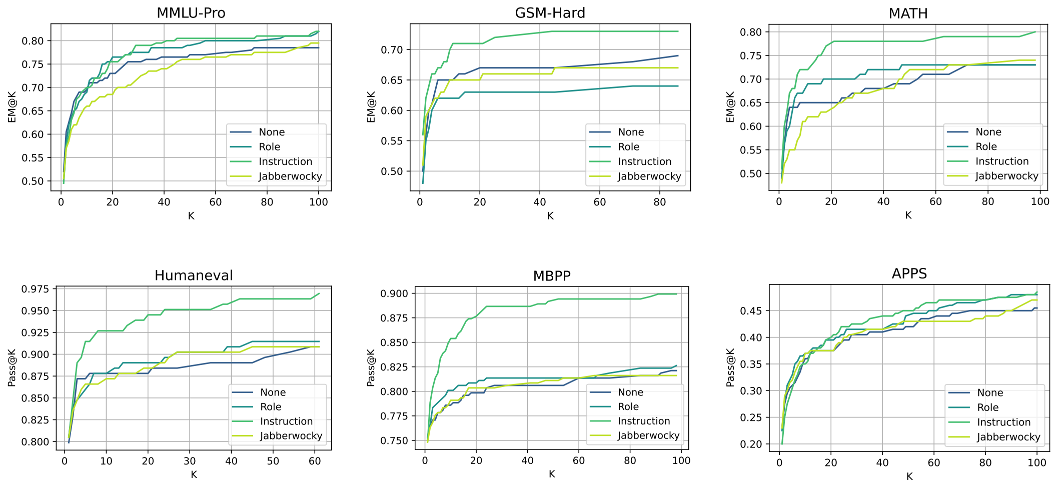


Figure 9: EM@k or Pass@k graphs of task-level perturbations versus direct sampling using `GPT-4o-mini` across six datasets.

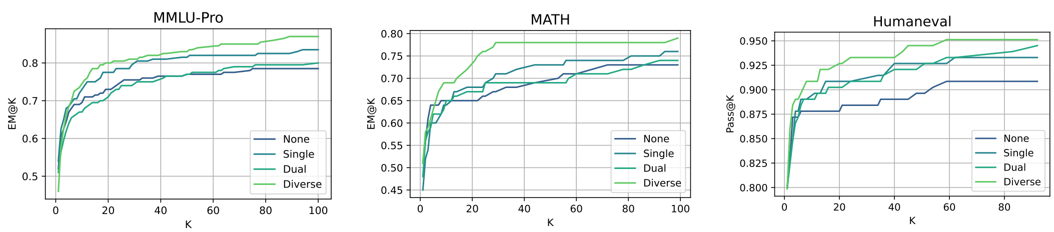


Figure 10: EM@k or Pass@k graphs of Single, Dual and Diverse variants of `RandIdeaInj` versus direct sampling on the MMLU-Pro, MATH and Humaneval benchmarks using `GPT-4o-mini`. In the Dual strategy, `GPT-3.5-turbo` OpenAI (2023a) serves as the thinker. The Diverse method utilizes a set of four models, including `GPT-3.5-turbo`, `GPT-4o-mini`, `Llama-3.1-8B-Instruct` Meta (2024) and `Qwen2.5-7B-Instruct` Yang et al. (2024).

random segment of poetry, is used as a reference perturbation. We run experiments on a server with 4 NVIDIA A100 GPUs, each one with 80GB RAM.

6.2 Results of Task-level Perturbations

We evaluate the task-level perturbations across six benchmarks spanning reasoning, mathematics, and code generation, comparing them to direct sampling. All method evaluations are allocated the same search budget of 100 solutions. Figure 9 shows their scaling curves of evaluation on `GPT-4o-mini` OpenAI (2023a). We find that injections consistently yield improvements across all tasks, with the instruction strategy performing the best, achieving notable increases of 6.7% EM@100 on MMLU-Pro, 9.6% EM@100 on MATH, and 9.5% Pass@100 on MBPP, over direct sampling.

6.3 Results of Query-level Perturbations

Random Idea Injection. The evaluation involves a range of `RandIdeaInj` strategies in Section 4.2, including the Single, Dual and Diverse variants, evaluated across the benchmarks MMLU-Pro, MATH, and HumanEval. Figure 10 displays the scaling curves of evaluations conducted with the generative model `GPT-4o-mini`. Evaluations are allocated the same search budget of 100 solutions. `RandIdeaInj` exhibits consistent improvement with idea-injected prompts, achieving a 10.8% increase in reasoning on MMLU-Pro, a 8.2% increase in the mathematics on MATH, and a 4.7% increase in coding on the Humaneval dataset, over the direct sampling. Similarities of diversified

Table 2: Effects of query-level perturbation variants (Single, Dual, and Diverse) on solution diversity compared to direct sampling.

Dataset	Strategy	Pass@100 (EM@100)	tf-idf sim.	BERT sim.	Lev. sim.	Seq. sim.
MMLU-Pro	None	0.7850	0.6565	0.9959	0.5350	0.6047
	Single	0.8350	0.5770	0.9930	0.4933	0.5473
	Dual	0.8000	0.6517	0.9959	0.5110	0.5961
	Diverse	0.8700	0.5688	0.9916	0.4838	0.5299
MATH	None	0.7300	0.7248	0.9971	0.5801	0.6819
	Single	0.7600	0.7188	0.9969	0.5771	0.6756
	Dual	0.7400	0.7193	0.9968	0.5782	0.6777
	Diverse	0.7900	0.6615	0.9960	0.5493	0.6344
HumanEval	None	0.9085	0.1907	0.9996	0.1941	0.1928
	Single	0.9329	0.1717	0.9993	0.1822	0.1846
	Dual	0.9451	0.1728	0.9993	0.1820	0.1863
	Diverse	0.9512	0.1525	0.9991	0.1582	0.1604

solutions are list in Table 2. Additional results for `RandIdeaInj` and its combination with task-level perturbations across models are provided in Appendix E.2.

Random Query Rephraser. Results for `RandQReph`, including both rephrasing and back-translation variants, are provided in Appendix E.3.

7 Additional Related Work

Scaling inference has explored diverse strategies for enhancing LLM capabilities through adaptive test-time compute allocation (Snell et al., 2024; Brown et al., 2024; Manvi et al., 2024; Guan et al., 2025; Chen et al., 2024b). Typically, LLM inference involves decomposing complex questions into sequential intermediate steps that lead to the final answer, exemplified by chain-of-thought (CoT) prompting (Wei et al., 2022; Sprague et al., 2024; Wang & Zhou, 2024) and its variants (Kojima et al., 2022; Zhou et al., 2023a; Wang et al., 2023c; Li et al., 2023). With the increasing number of steps in a single chain, these methods often suffer from error propagation and struggle with complex computations (Chen et al., 2023). To address the limitation, CoT (Li et al., 2024) has been improved with search-based methods (Zhang et al., 2024c; Yao et al., 2024b; Luo et al., 2024; Light et al., 2025), such as beam search (Xie et al., 2024b) and Best-of-N (Snell et al., 2024). Subsequently, tree search algorithms including MCTS and A* (Yao et al., 2024b; Luo et al., 2024; Zhang et al., 2024a; Hao et al., 2023; Zhou et al., 2024; Choi et al., 2023; Yao et al., 2024a; Chen et al., 2024c; Xie et al., 2024a; Zhang et al., 2025) further introduced diversity into inference computation by exploring multiple reasoning paths at different levels. In parallel, diverse prompting strategies (Li et al., 2023; Mu et al., 2024; Naik et al., 2023) have been developed to elicit multiple CoT reasoning trajectories, which ultimately converge to more accurate solutions for complex reasoning tasks. While these methods show that extended inference-time search improves performance, they do not systematically examine the effect of exploration diversity in scaling inference. In this paper, we study why sampling diversity benefits Best-of-N, how to promote it through perturbations, when diversified sampling succeeds, and how effective it is across tasks. See Appendix F for extended discussion of literature.

8 Conclusion

This work provides a systematic study of exploration diversity in scaling inference. We offered a theoretical perspective on why exploration diversity enhances Best-of-N performance, analyzed how to encourage diversity through meaningful perturbations and conditions under which diversification succeeds. We empirically show how effective are diversified sampling across reasoning, math and code generation tasks. Our findings highlight exploration diversity as an effective approach for improving test-time scaling in most cases. Future research will focus on designing adaptive perturbation strategies and integrating diversity into broader inference frameworks.

Ethics statement

All authors of this paper confirm that they have read and pledged to uphold the ICLR Code of Ethics. This study focuses on evaluating the effect of diversified sampling in LLM inference, with the goal of advancing the understanding of test-time scaling. Our experiments are conducted solely on publicly available benchmarks for reasoning, mathematics, and code generation, without involving human subjects or sensitive data. We aim to contribute to the responsible development of advanced AI technologies by analyzing techniques that enhance inference efficiency and reliability.

Reproducibility statement

For detailed reproducibility information, including full implementation details, hyperparameters, and evaluation protocols, please refer to the main text and the appendix. All proofs are presented in the main text and appendix with detailed explanations and assumptions. We carefully report implementation details to facilitate verification, with the aim of ensuring that all results can be reliably reproduced and extended by the community. All source code, data, and configuration files will be released to ensure the full reproducibility of our results.

References

Anthropic. Introducing claude 3.5 sonnet, June 2024. URL `https://www.anthropic.com/news/claude-3-5-sonnet`.

Berk Atil, Sarp Aykent, Alexa Chittams, Lisheng Fu, Rebecca J Passonneau, Evan Radcliffe, Guru Rajan Rajagopal, Adam Sloan, Tomasz Tudrej, Ferhan Ture, et al. Non-determinism of" deterministic" llm settings. *arXiv preprint arXiv:2408.04667*, 2024.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Djamila Romaissa Beddiar, Md Saroar Jahan, and Mourad Oussalah. Data expansion using back translation and paraphrasing for hate speech detection, 2021. URL `https://arxiv.org/abs/2106.04681`.

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.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Alphamath almost zero: process supervision without process. *arXiv preprint arXiv:2405.03553*, 2024a.

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

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do not think that much for 2+3=? on the overthinking of o1-like llms, 2024b. URL `https://arxiv.org/abs/2412.21187`.

Ziru Chen, Michael White, Raymond Mooney, Ali Payani, Yu Su, and Huan Sun. When is tree search useful for llm planning? it depends on the discriminator, 2024c. URL `https://arxiv.org/abs/2402.10890`.

Sehyun Choi, Tianqing Fang, Zhaowei Wang, and Yangqiu Song. Kcts: Knowledge-constrained tree search decoding with token-level hallucination detection. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 14035–14053, 2023.

Geoffrey Cideron, Andrea Agostinelli, Johan Ferret, Sertan Girgin, Romuald Elie, Olivier Bachem, Sarah Perrin, and Alexandre Ramé. Diversity-rewarded cfg distillation, 2024. URL `https://arxiv.org/abs/2410.06084`.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Sara Cothren Cook, Lauren W Collins, Lisa L Morin, and Paul J Riccomini. Schema-based instruction for mathematical word problem solving: An evidence-based review for students with learning disabilities. *Learning Disability Quarterly*, 43(2):75–87, 2020.

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, 2025. URL `https://arxiv.org/abs/2501.12948`.

Yihe Deng, Weitong Zhang, Zixiang Chen, and Quanquan Gu. Rephrase and respond: Let large language models ask better questions for themselves. *arXiv preprint arXiv:2311.04205*, 2023.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1536–1547, 2020.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D Goodman. Stream of search (sos): Learning to search in language. *arXiv preprint arXiv:2404.03683*, 2024.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. rstar-math: Small llms can master math reasoning with self-evolved deep thinking, 2025. URL `https://arxiv.org/abs/2501.04519`.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Empirical Methods in Natural Language Processing*, pp. 8154–8173, 2023.

Horace He et al. Defeating nondeterminism in llm inference. *Thinking Machines Lab: Connectionism*, 2025. `https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/`.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021b.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

A Kong, S Zhao, H Chen, Q Li, Y Qin, R Sun, X Zhou, E Wang, and X Dong. Better zero-shot reasoning with role-play prompting. arxiv. Technical report, Retrieved 2024-05-18, from http://arxiv. org/abs/2308.07702, 2024.

Kuang-Huei Lee, Ian Fischer, Yueh-Hua Wu, Dave Marwood, Shumeet Baluja, Dale Schuurmans, and Xinyun Chen. Evolving deeper llm thinking, 2025. URL `https://arxiv.org/abs/2501.09891`.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models, 2022. *URL https://arxiv. org/abs/2206.14858*, 1, 2022.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. Making language models better reasoners with step-aware verifier. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5315–5333, 2023.

Yiwei Li, Ji Zhang, Shaoxiong Feng, Peiwen Yuan, Xinglin Wang, Jiayi Shi, Yueqi Zhang, Chuyi Tan, Boyuan Pan, Yao Hu, et al. Revisiting self-consistency from dynamic distributional alignment perspective on answer aggregation. *arXiv preprint arXiv:2502.19830*, 2025.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems, 2024. URL `https://arxiv.org/abs/2402.12875`.

Jonathan Light, Wei Cheng, Wu Yue, Masafumi Oyamada, Mengdi Wang, Santiago Paternain, and Haifeng Chen. Disc: Dynamic decomposition improves llm inference scaling. *arXiv preprint arXiv:2502.16706*, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language models by automated process supervision, 2024. URL https://arxiv.org/abs/2406.06592.

Rohin Manvi, Anikait Singh, and Stefano Ermon. Adaptive inference-time compute: Llms can predict if they can do better, even mid-generation. *arXiv preprint arXiv:2410.02725*, 2024.

Meta. Meta-llama 3.1-8b instruct, 2024. URL https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct. Accessed: 2024-09-03.

Lin Mu, Wenhao Zhang, Yiwen Zhang, and Peiquan Jin. Ddprompt: Differential diversity prompting in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 168–174, 2024.

Ranjita Naik, Varun Chandrasekaran, Mert Yuksekgonul, Hamid Palangi, and Besmira Nushi. Diversity of thought improves reasoning abilities of llms. *arXiv preprint arXiv:2310.07088*, 2023.

Harsha Nori, Naoto Usuyama, Nicholas King, Scott Mayer McKinney, Xavier Fernandes, Sheng Zhang, and Eric Horvitz. From medprompt to o1: Exploration of run-time strategies for medical challenge problems and beyond, 2024. URL https://arxiv.org/abs/2411.03590.

OpenAI. Gpt-3.5-turbo, 2023a. URL https://platform.openai.com/docs/models/gpt-3-5.

OpenAI. Gpt-4o-mini, 2023b. URL https://platform.openai.com/docs/models/gpt-4.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

Murray Shanahan, Kyle McDonell, and Laria Reynolds. Role play with large language models. *Nature*, 623(7987):493–498, 2023.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling test-time compute optimally can be more effective than scaling LLM parameters. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Zayne Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. To cot or not to cot? chain-of-thought helps mainly on math and symbolic reasoning, 2024. URL https://arxiv.org/abs/2409.12183.

Evan Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, Will Song, Vaskar Nath, Ziwen Han, Sean Hendryx, Summer Yue, and Hugh Zhang. Planning in natural language improves LLM search for code generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=48WAZhwHHw.

Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Y Wu, and Zhifang Sui. Math-shepherd: A label-free step-by-step verifier for llms in mathematical reasoning. *arXiv preprint arXiv:2312.08935*, 2023a.

Xuezhi Wang and Denny Zhou. Chain-of-thought reasoning without prompting, 2024. URL https://arxiv.org/abs/2402.10200.

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*, 2023b. URL https://openreview.net/forum?id=1PL1NIMMrw.

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 *International Conference on Learning Representations*, 2023c.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models, 2024. URL https://arxiv.org/abs/2408.00724.

Violet Xiang, Charlie Snell, Kanishk Gandhi, Alon Albalak, Anikait Singh, Chase Blagden, Duy Phung, Rafael Rafailov, Nathan Lile, Dakota Mahan, Louis Castricato, Jan-Philipp Franken, Nick Haber, and Chelsea Finn. Towards system 2 reasoning in llms: Learning how to think with meta chain-of-thought, 2025. URL https://arxiv.org/abs/2501.04682.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P. Lillicrap, Kenji Kawaguchi, and Michael Shieh. Monte carlo tree search boosts reasoning via iterative preference learning, 2024a. URL https://arxiv.org/abs/2405.00451.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. Self-evaluation guided beam search for reasoning. *Advances in Neural Information Processing Systems*, 36, 2024b.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

Huanjin Yao, Jiaxing Huang, Wenhao Wu, Jingyi Zhang, Yibo Wang, Shunyu Liu, Yingjie Wang, Yuxin Song, Haocheng Feng, Li Shen, and Dacheng Tao. Mulberry: Empowering mllm with o1-like reasoning and reflection via collective monte carlo tree search, 2024a. URL https://arxiv.org/abs/2412.18319.

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, 2024b.

Jiayi Yuan, Hao Li, Xinheng Ding, Wenya Xie, Yu-Jhe Li, Wentian Zhao, Kun Wan, Jing Shi, Xia Hu, and Zirui Liu. Give me fp32 or give me death? challenges and solutions for reproducible reasoning. *arXiv preprint arXiv:2506.09501*, 2025.

Zhiyuan Zeng, Qinyuan Cheng, Zhangyue Yin, Bo Wang, Shimin Li, Yunhua Zhou, Qipeng Guo, Xuanjing Huang, and Xipeng Qiu. Scaling of search and learning: A roadmap to reproduce o1 from reinforcement learning perspective, 2024. URL https://arxiv.org/abs/2412.14135.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search, 2024a. URL https://arxiv.org/abs/2406.03816.

Shimao Zhang, Yu Bao, and Shujian Huang. Edt: Improving large language models' generation by entropy-based dynamic temperature sampling. *arXiv preprint arXiv:2403.14541*, 2024b.

Xuan Zhang, Chao Du, Tianyu Pang, Qian Liu, Wei Gao, and Min Lin. Chain of preference optimization: Improving chain-of-thought reasoning in llms, 2024c. URL `https://arxiv.org/abs/2406.09136`.

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, 2025. URL `https://arxiv.org/abs/2501.07301`.

Li Zhong, Zilong Wang, and Jingbo Shang. Debug like a human: A large language model debugger via verifying runtime execution step-by-step, 2024. *URL https://arxiv. org/abs/2402.16906*.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *International Conference on Machine Learning*, 2024.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, et al. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations*, 2023a.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023b.

APPENDIX

A THE USE OF LLMS

We use LLMs as general-purpose assistants for language editing and refinement of manuscript drafts.

B THEORETICAL DETAILS

B.1 PRELIMINARIES AND NOTATION

In this entire section, we use upper case letters K for the number of attempts in the analysis below and lower case letters k for attempt indices. Inputs are $\boldsymbol{r} = [\boldsymbol{p}, \boldsymbol{q}]$ with $\boldsymbol{r} \sim \mathcal{R} := \{\boldsymbol{p}\} \times \mathcal{Q}$ and support R. Each attempt is configured by a *hybrid* mode $\zeta = (\eta, \xi)$, where

- $\eta \sim \nu_0$ is the *base* decoding randomness/configuration (e.g., seed, sampler, temperature, CoT style);
- $\xi \sim \Pi$ is an *auxiliary* diversity source (to be instantiated later).

Given $(\boldsymbol{r}, \eta, \xi)$, the solver draws $\boldsymbol{s} \sim \mathrm{LLM}(\cdot \mid \boldsymbol{r}, \eta, \xi)$ and the verifier V returns $0/1$. Define the per–mode failure log–probability

$$q(\boldsymbol{r}, \eta, \xi) \; := \; \log \mathbb{P}_{\boldsymbol{s} \sim \mathrm{LLM}(\cdot|\boldsymbol{r},\eta,\xi)}\big[V(\boldsymbol{s}) = 0\big]. \tag{7}$$

Unsolvable sets. Let

$$\mathrm{R}_0 \; := \; \big\{\boldsymbol{r} \in \mathrm{R} : \; \mathbb{P}[V(\boldsymbol{s}) = 0 \mid \boldsymbol{r}, \eta, \xi^0] = 1 \text{ for } \eta \sim \nu_0\big\}, \tag{8}$$

$$\mathrm{R}_0^\star \; := \; \big\{\boldsymbol{r} \in \mathrm{R} : \; \mathbb{P}[V(\boldsymbol{s}) = 0 \mid \boldsymbol{r}, \eta, \xi] = 1 \text{ for all } \eta, \xi\big\}. \tag{9}$$

Here ξ^0 is a fixed auxiliary reference used by the baseline policy; clearly $\mathrm{R}_0^\star \subseteq \mathrm{R}_0$. We denote by $\mu_{\mathcal{R}}$ the probability measure induced by $\mathcal{R}$ on R.

Best-of-K failure. For a policy that samples K i.i.d. modes (η_k, ξ_k) and produces $\boldsymbol{s}_k \sim \mathrm{LLM}(\cdot \mid \boldsymbol{r}, \eta_k, \xi_k)$, the failure probability is

$$P^K \; := \; \mathbb{P}\Big[V(\boldsymbol{s}_k) = 0, \; \forall k \in [K] \;\Big|\; \boldsymbol{r} \sim \mathcal{R}, \; (\eta_k, \xi_k) \text{ i.i.d.}\Big]. \tag{10}$$

We write P^K_{reg} for the *baseline* where $(\eta_k, \xi_k) = (\eta_k, \xi^0)$ with $\eta_k \overset{\text{i.i.d.}}{\sim} \nu_0$, and P^K_{div} for the *hybrid diversified* case $(\eta_k, \xi_k) \overset{\text{i.i.d.}}{\sim} \nu_0 \times \Pi$. Their limits as $K \to \infty$ are $P^{\inf}_{\mathrm{reg}} = \mu_{\mathcal{R}}(\mathrm{R}_0)$ and $P^{\inf}_{\mathrm{div}} = \mu_{\mathcal{R}}(\mathrm{R}_0^\star)$.

B.2 HYPOTHESIS 1: DISPERSION UNDER AUXILIARY DIVERSITY

We formalize the dispersion contributed by the auxiliary source while keeping the base randomness.

Assumption B.1 (Dispersion first moment)**.** There exists $\hat{\mu}_1 > 0$ such that for all $\boldsymbol{r} \notin \mathrm{R}_0^\star$,

$$M_1(\boldsymbol{r}; \nu_0, \Pi) \; := \; \mathbb{E}_{\eta \sim \nu_0} \, \mathbb{E}_{\xi \sim \Pi} \Big| \, q(\boldsymbol{r}, \eta, \xi) \; - \; \mathbb{E}_{\xi \sim \Pi} q(\boldsymbol{r}, \eta, \xi) \Big| \;\; \geq \;\; \hat{\mu}_1. \tag{11}$$

Optionally, one may also posit a (weaker) second-moment lower bound $\mathbb{E}_{\eta,\xi}\big(q - \mathbb{E}_\xi q\big)^2 \geq \hat{\mu}_2 > 0$; our bounds only require the first moment.

Remark B.2. Intuitively, unless $\boldsymbol{r}$ is genuinely unsolvable for *all* (η, ξ), varying the auxiliary ξ changes the failure likelihood while the base randomness η remains at its usual variability.

Variance decomposition (used implicitly). By the law of total variance applied to $q(\boldsymbol{r}, \eta, \xi)$ with independent $\eta \sim \nu_0, \; \xi \sim \Pi$,

$$\mathrm{Var}_{\eta,\xi}\big(q(\boldsymbol{r}, \eta, \xi)\big) = \mathbb{E}_\eta\Big[\mathrm{Var}_\xi\big(q(\boldsymbol{r}, \eta, \xi)\big)\Big] \; + \; \mathrm{Var}_\eta\Big(\mathbb{E}_\xi[q(\boldsymbol{r}, \eta, \xi)]\Big) \; \geq \; \mathbb{E}_\eta\Big[\mathrm{Var}_\xi\big(q(\boldsymbol{r}, \eta, \xi)\big)\Big]. \tag{12}$$

Assumption B.1 ensures the *absolute* first central moment over ξ is uniformly bounded away from zero after averaging over η, hence dispersion is nontrivial.

B.3 Hypothesis 2: Fidelity with Fixed Auxiliary Setting

We compare policies under the *same* base randomness, and with auxiliary fixed on both sides.

Assumption B.3 (Localized fidelity)**.** There exist $\epsilon \in [0, 1)$ and auxiliary settings $\xi^\star, \xi^0$ such that for all $K \geq 1$,

$$\frac{\mathbb{E}_{\boldsymbol{r}\sim\mathcal{R}}\,\mathbb{E}_{\eta\sim\nu_0}\Big[\exp\{K\,q(\boldsymbol{r},\eta,\xi^\star)\}\Big]}{\mathbb{E}_{\boldsymbol{r}\sim\mathcal{R}}\,\mathbb{E}_{\eta\sim\nu_0}\Big[\exp\{K\,q(\boldsymbol{r},\eta,\xi^0)\}\Big]} \;\leq\; 1+\epsilon. \tag{13}$$

Remark B.4. The auxiliary choice is *fixed* on both sides so that both policies share the same base variability ($\eta \sim \nu_0$). This guards against conflating auxiliary gains with changes in base decoding.

B.4 Main Theorem and Proof

We restate the main comparison bound in the hybrid framework and prove it.

Theorem B.5 (Hybrid diversity improves Best-of-K)**.** *Under Assumptions B.1 and B.3, there exists a sequence $C_K = \Omega\big(\hat{\mu}_1^2\,K/(1+\epsilon)\big)$, positive and increasing in K, such that*

$$N_{\mathrm{div}}^K \;\leq\; \frac{N_{\mathrm{reg}}^K - N_{\mathrm{reg}}^{\inf}}{1+C_K} \;+\; N_{\mathrm{div}}^{\inf}, \qquad \textit{with} \quad N_{\mathrm{div}}^{\inf} \;\leq\; N_{\mathrm{reg}}^{\inf}. \tag{14}$$

Proof. Write $q_k := q(\boldsymbol{r},\eta_k,\xi_k)$, where $(\eta_k,\xi_k) \overset{\text{i.i.d.}}{\sim} \nu_0 \times \Pi$ for the diversified policy. By independence of attempts,

$$N_{\mathrm{div}}^K = \mathbb{E}_{\boldsymbol{r}\sim\mathcal{R}}\,\mathbb{E}_{(\eta_k,\xi_k)}\Big[\exp\Big\{\sum_{k=1}^K q_k\Big\}\Big]. \tag{15}$$

Introduce the sample mean $\bar{q} := \frac{1}{K}\sum_{k=1}^K q_k$ and the mixed mean $\bar{q}_\Pi(\boldsymbol{r}) := \mathbb{E}_{\eta,\xi}\,q(\boldsymbol{r},\eta,\xi)$ (distinct from $\bar{q}$). For $g(x) = e^x - x - 1 \geq \min\{0.25x^2, 0.5|x|\}$,

$$\frac{\frac{1}{K}\sum_{k=1}^K \exp\{Kq_k\}}{\exp\{\sum_{k=1}^K q_k\}} = \frac{1}{K}\sum_{k=1}^K \exp\{K(q_k-\bar{q})\} = \frac{1}{K}\sum_{k=1}^K \big(1+K(q_k-\bar{q})+g(K(q_k-\bar{q}))\big). \tag{16}$$

Hence

$$\frac{\frac{1}{K}\sum_{k=1}^K \exp\{Kq_k\}}{\exp\{\sum_{k=1}^K q_k\}} \;\geq\; 1 \;+\; \min\Big\{0.5\sum_{k=1}^K |q_k-\bar{q}|\,,\; 0.25\,K\sum_{k=1}^K (q_k-\bar{q})^2\Big\}. \tag{17}$$

By Hoeffding (or Bernstein) and Assumption B.1, with probability at least $1-\delta$ (over draws of (η_k,ξ_k)) we have, uniformly for $\boldsymbol{r} \notin \mathrm{R}_0^\star$,

$$\sum_{k=1}^K |q_k-\bar{q}| \;\geq\; \hat{\mu}_1 K - C\sqrt{K\log(1/\delta)}, \tag{18}$$

$$\sum_{k=1}^K (q_k-\bar{q})^2 \;\geq\; \frac{1}{K}\Big(\sum_{k=1}^K |q_k-\bar{q}|\Big)^2 \;\geq\; \hat{\mu}_1^2 K - C'\sqrt{K\log(1/\delta)}. \tag{19}$$

Plugging equation 18–equation 19 into equation 17, and absorbing deviations into constants C_1, C_2 independent of $\boldsymbol{r}$, yields

$$\frac{\frac{1}{K}\sum_{k=1}^K \exp\{Kq_k\}}{\exp\{\sum_{k=1}^K q_k\}} \;\geq\; 1 + C_1 K - C_2\sqrt{K}. \tag{20}$$

Hence, conditioning on $\boldsymbol{r} \notin \mathrm{R}_0^\star$ and taking expectations,

$$\mathbb{E}\Big[\exp\Big\{\sum_{k=1}^K q_k\Big\}\;\Big|\;\boldsymbol{r}\Big] \;\leq\; \frac{1}{1+C_1K - C_2\sqrt{K}} \;\cdot\; \frac{1}{K}\sum_{k=1}^K \mathbb{E}\big[\exp\{Kq_k\}\;\big|\;\boldsymbol{r}\big]. \tag{21}$$

Now apply Assumption B.3 with $(\eta_k, \xi_k) \sim \nu_0 \times \Pi$ versus baseline (η_k, ξ^0), and average over $\boldsymbol{r} \sim \mathcal{R}$:

$$\frac{1}{K}\sum_{k=1}^{K} \mathbb{E}_{\boldsymbol{r}} \mathbb{E}_{\eta,\xi}\big[\exp\{Kq(\boldsymbol{r}, \eta, \xi)\}\big] \;\le\; (1+\epsilon)\, \mathbb{E}_{\boldsymbol{r}} \mathbb{E}_{\eta}\big[\exp\{Kq(\boldsymbol{r}, \eta, \xi^0)\}\big]. \tag{22}$$

Combining equation 15, equation 21, equation 22, and then splitting the expectation over R into $\mathrm{R} \setminus \mathrm{R}_0^\star$ and $\mathrm{R}_0^\star$ (where $\exp\{\sum q_k\} \equiv 1$), we obtain

$$N_{\mathrm{div}}^{K} - N_{\mathrm{div}}^{\mathrm{inf}} \;\le\; \frac{1+\epsilon}{1 + C_1 K - C_2\sqrt{K}} \left(N_{\mathrm{reg}}^{K} - N_{\mathrm{reg}}^{\mathrm{inf}}\right). \tag{23}$$

Letting $C_K := (C_1 K - C_2\sqrt{K} - \epsilon)/(1+\epsilon)$, which is $\Omega(\hat{\mu}_1^2 K/(1+\epsilon))$ by construction of C_1 through equation 18–equation 19, yields equation 14. Finally, $N_{\mathrm{div}}^{\mathrm{inf}} = \mu_{\mathcal{R}}(\mathrm{R}_0^\star) \le \mu_{\mathcal{R}}(\mathrm{R}_0) = N_{\mathrm{reg}}^{\mathrm{inf}}$ by equation 8. □

Remark B.6 (Moment effect and asymptote). The leading term of C_K scales as $\Theta\big(\hat{\mu}_1^2 K/(1+\epsilon)\big)$: richer (but faithful) auxiliary diversity increases the dispersion (larger $\hat{\mu}_1$), which steepens the Best-of-K contraction; simultaneously, the unsolvable set shrinks from R_0 to $\mathrm{R}_0^\star$ so the limiting failure $N_{\mathrm{div}}^{\mathrm{inf}}$ decreases.

B.5 Additional Theoretical Discussions

With our theory at the ready, we now discuss several aspects of diversified sampling to support some of our empirical findings.

B.5.1 Perturbation–Question similarity and the diversity–fidelity tradeoff

Empirically (Fig. 2), EM/Pass improve as perturbations become more relevant to the question, peak around task-aligned ideas, then drop when relevance becomes excessive (rephrasings/verbatim). This matches our theory through the two quantities already in the main text:

(A) Dispersion M_1 shrinks when relevance is *too high*. When perturbations are near rephrasings or verbatim repeats, they induce very similar solver behavior across attempts, so the failure log-probability barely moves. Hence the first-moment M_1 is small, the convergence factor

$$C_N \;\propto\; \frac{M_1^2\, N}{1+\epsilon}$$

is small, and Best-of-N gains vanish. This explains the decline from Perturbation 4 (rephrasing) to Perturbation 5 (verbatim).

(B) Fidelity ϵ worsens when relevance is *too low*. When perturbations are off-topic or loosely related, single-attempt quality degrades. In our bound this appears as a larger ϵ, which *divides* the gain: even if diversity increases, the $(1+\epsilon)$ penalty suppresses C_N and reduces overall improvement. This explains the weak performance of Perturbation 1 (irrelevant) and the partial recovery at Perturbation 2 (loosely related).

(C) The sweet spot is *task-aligned* ideas. At intermediate relevance, perturbations are different enough to move the model into alternative solution modes (so M_1 is substantial) while still faithful to the task (so ϵ remains modest). This maximizes the effective gain $M_1^2/(1+\epsilon)$, producing the peak at Perturbation 3.

Design takeaway. Concentrate perturbations in a *moderate relevance band*: different enough to create exploration (boost M_1), but faithful enough to avoid harming single-attempt quality (control ϵ). This is exactly where our experiments see the maximum Best-of-N improvement.

B.5.2 Outcome reward models (ORM) and why top-k captures the best

Let a single instance produce N candidate solutions $S = \{s_1, \ldots, s_N\}$. The oracle (ground truth) verifier V labels a solution correct if it matches the hidden answer (or passes hidden tests). An *outcome reward model* (ORM) assigns a real score $R(s)$ to each s, used to rank candidates; the top-k by R are selected.

We analyze the event that the *best* correct solution $s^\star$ (if one exists) appears in the top-k by R. Write a simple low-error model for the ORM:

$$R(s) \;=\; \mu(s) \;+\; \varepsilon_s, \qquad \varepsilon_s \text{ i.i.d. sub-Gaussian with variance proxy } \sigma^2, \tag{24}$$

and assume a *margin* $\gamma > 0$ between $s^\star$ and every incorrect solution $\tilde{s}$ in the latent score:

$$\mu(s^\star) \;\geq\; \max_{\tilde{s}:\, V(\tilde{s})=0} \; \mu(\tilde{s}) \;+\; \gamma. \tag{25}$$

Theorem B.7 (ORM top-k recall under a margin)**.** *Under the model above, for any incorrect $\tilde{s}$,*

$$\Pr\big[R(\tilde{s}) \;\geq\; R(s^\star)\big] \;\leq\; p_\gamma \;:=\; \exp\Big(-\tfrac{\gamma^2}{4\sigma^2}\Big). \tag{26}$$

Let X be the number of incorrect solutions that outrank $s^\star$. Then $\mathbb{E}[X] \;\leq\; (N-1)\,p_\gamma$ and, by Markov,

$$\Pr\big[s^\star \;\notin\; \text{Top-}k\big] \;=\; \Pr[X \geq k] \;\leq\; \frac{(N-1)\,p_\gamma}{k}. \tag{27}$$

In particular, with $k = 10$,

$$\Pr\big[s^\star \;\in\; \text{Top-10}\big] \;\geq\; 1 \;-\; \frac{(N-1)}{10}\;\exp\Big(-\tfrac{\gamma^2}{4\sigma^2}\Big). \tag{28}$$

Proof of Theorem B.7. Let $s^\star$ be a correct solution and suppose the latent scores satisfy the margin

$$\mu(s^\star) \;\geq\; \max_{\tilde{s}:\, V(\tilde{s})=0} \; \mu(\tilde{s}) \;+\; \gamma, \tag{29}$$

for some $\gamma > 0$. For any incorrect $\tilde{s}$, consider the difference

$$R(\tilde{s}) - R(s^\star) \;=\; \big(\mu(\tilde{s}) - \mu(s^\star)\big) \;+\; \big(\varepsilon_{\tilde{s}} - \varepsilon_{s^\star}\big) \;\leq\; -\gamma \;+\; Z_{\tilde{s}}, \tag{30}$$

where $Z_{\tilde{s}} := \varepsilon_{\tilde{s}} - \varepsilon_{s^\star}$ is sub-Gaussian with variance proxy $2\sigma^2$ (since the ε's are i.i.d. sub-Gaussian with proxy σ^2).

By the standard sub-Gaussian tail bound, for any $t > 0$,

$$\Pr\big[Z_{\tilde{s}} \geq t\big] \;\leq\; \exp\Big(-\tfrac{t^2}{4\sigma^2}\Big). \tag{31}$$

Setting $t = \gamma$ yields

$$\Pr\big[R(\tilde{s}) \;\geq\; R(s^\star)\big] \;\leq\; \exp\Big(-\tfrac{\gamma^2}{4\sigma^2}\Big) \;:=\; p_\gamma. \tag{32}$$

Let $I_{\tilde{s}} \;:=\; \mathbb{1}\{R(\tilde{s}) \;\geq\; R(s^\star)\}$ and $X \;:=\; \sum_{\tilde{s}:\, V(\tilde{s})=0} I_{\tilde{s}}$ be the number of incorrect solutions that outrank $s^\star$. By linearity of expectation and equation 32,

$$\mathbb{E}[X] \;=\; \sum_{\tilde{s}:\, V(\tilde{s})=0} \mathbb{E}[I_{\tilde{s}}] \;\leq\; (N-1)\,p_\gamma, \tag{33}$$

where we crudely upper bound the number of competitors by $N-1$. By Markov's inequality,

$$\Pr\big[X \;\geq\; k\big] \;\leq\; \frac{\mathbb{E}[X]}{k} \;\leq\; \frac{(N-1)\,p_\gamma}{k}. \tag{34}$$

The event $\{s^\star \notin \text{Top-}k\}$ implies that at least k candidates outrank $s^\star$, hence

$$\Pr\big[s^\star \notin \text{Top-}k\big] \;\leq\; \Pr[X \geq k] \;\leq\; \frac{(N-1)\,p_\gamma}{k}, \tag{35}$$

which proves the stated bound. For $k = 10$ this gives

$$\Pr\big[s^\star \in \text{Top-10}\big] \;\geq\; 1 \;-\; \frac{(N-1)}{10}\;\exp\Big(-\tfrac{\gamma^2}{4\sigma^2}\Big). \tag{36}$$

Multiple correct solutions. If there are several correct solutions $\{s_j^\star\}_{j=1}^m$ each separated by the same (or larger) margin γ from *all* incorrect solutions, define X_j as the number of incorrect solutions outranking $s_j^\star$. Then by the same argument,

$$\Pr\big[\forall j,\ s_j^\star \notin \text{Top-}k\big] \ \le\ \Pr\big[\min_j X_j \ge k\big] \ \le\ \min_j \Pr[X_j \ge k] \ \le\ \frac{(N-1)\,p_\gamma}{k}, \tag{37}$$

so the probability that *none* of the correct solutions appears in top-k is no larger than in the single-best case. Consequently, the ORM pass@k differs from oracle pass@k only when all correct solutions are excluded from the ORM's top-k, an event controlled by the bound above. □

Implications. If the ORM is *high quality* (small σ) and the correct solution is reasonably separated (moderate γ), the chance that top-k by R misses $s^\star$ decays rapidly—in particular, top-10 typically contains the best solution. Consequently, the ORM pass@k closely tracks the oracle pass@k (they differ only when the oracle-best is excluded from the ORM's top-k). With multiple correct solutions, the probability that *none* appear in top-k is even smaller (union bound), further tightening alignment between ORM and oracle pass@k.

B.5.3 Why majority voting does not mirror Best-of-N gains

Set-up. Fix an instance with a discrete answer space $\mathcal{Y}$ (e.g., final numbers in math). Under a sampling policy π (e.g., baseline or a diversified hybrid), let

$$p_\pi(y) \ :=\ \Pr_{\boldsymbol{s}\sim\pi}[\text{final answer of } \boldsymbol{s} = y], \qquad y \in \mathcal{Y}, \tag{38}$$

and denote the ground-truth answer by $y^\star$. Majority voting over N i.i.d. samples chooses the label with the largest empirical frequency.

Asymptotics of majority vote. By the law of large numbers, empirical frequencies converge to $\{p_\pi(y)\}_y$. Hence the majority-vote output converges almost surely to

$$\arg\max_{y\in\mathcal{Y}} \ p_\pi(y). \tag{39}$$

Therefore, the *asymptotic* majority-vote accuracy equals

$$\lim_{N\to\infty} \Pr[\mathrm{MV}_N = y^\star] \ =\ \mathbf{1}\Big\{\ p_\pi(y^\star) \ =\ \max_y p_\pi(y)\ \Big\}, \tag{40}$$

i.e., it is 1 if and only if the correct answer is already the *most probable* single-sample outcome under π, and 0 otherwise.

Contrast with Best-of-N. Best-of-N success is

$$\Pr[\mathrm{BoN}_N \text{ hits } y^\star] \ =\ 1 - (1 - p_\pi(y^\star))^N, \tag{41}$$

which increases *monotonically* to 1 whenever $p_\pi(y^\star) > 0$. Thus, BoN rewards *diversity of attempts* (any nonzero mass on $y^\star$ helps), whereas majority vote rewards *mass concentration* (the correct label must be the single most likely).

Implication for diversified sampling. Diversification typically *spreads* probability mass across multiple solution modes. Unless it also raises $p_\pi(y^\star)$ above all competitors, majority voting has no systematic reason to improve—and can degrade if the spread lowers the rank of $y^\star$. This explains the inconsistent gains in Fig. 8.

A simple bound. Let X_y be the count of label y among N i.i.d. draws. Then

$$\Pr[\mathrm{MV}_N = y^\star] \ =\ \Pr\big[X_{y^\star} \ \ge\ \max_{y\ne y^\star} X_y\big]. \tag{42}$$

When $p_\pi(y^\star) \le \max_{y\ne y^\star} p_\pi(y)$, concentration of measure implies $\Pr[\mathrm{MV}_N = y^\star] \to 0$ as $N \to \infty$; when $p_\pi(y^\star)$ is uniquely maximal, the probability $\to 1$. In contrast, $\Pr[\mathrm{BoN}_N \text{ hits } y^\star] \to 1$ whenever $p_\pi(y^\star) > 0$.

Relation to fidelity (Hypothesis 3.3). For fixed auxiliary settings, Hypothesis 3.3 controls the *single-attempt* failure via a factor $(1+\epsilon)$ across policies. Majority voting asymptotically reduces to a *best@1* decision (choosing the MAP label), so it inherits this limitation: switching to a diversified auxiliary setting cannot guarantee improvement and may even inflate error by up to a $(1+\epsilon)$ factor relative to baseline, whereas Best-of-N benefits strictly from added dispersion.

C Details of Metrics

For each of our metrics, the solver is allowed k submissions for each, denoted by $[\boldsymbol{s}]_k \sim \mathrm{LLM}(\cdot|\boldsymbol{r}, k)$ given input $\boldsymbol{r}$. We consider testing the model on a set of tasks consisting of prompts and questions $\mathbf{X} = \{\boldsymbol{r} = [\boldsymbol{p}, \boldsymbol{q}]\}$.

EM@k Rate. For reasoning and math tasks, if at least one submission $s' \in [\boldsymbol{s}]_k$ matches the ground truth, the task is considered solved. The EM@k rate is defined as the proportion of tasks solved as

$$\textbf{EM@k} = \frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{r} \in \mathbf{X}} \mathbb{1}\big(\exists \boldsymbol{s} \in [\boldsymbol{s}]_k, \text{s.t.}, \boldsymbol{s} = \boldsymbol{H} \big| [\boldsymbol{s}]_k \sim \mathrm{LLM}(\cdot|\boldsymbol{r}, k)\big),$$

where $\mathbb{1}(\cdot)$ is the indicator function and $\boldsymbol{H}$ is the ground truth.

Pass@k Rate. For code generation tasks, if at least one submission $s' \in [\boldsymbol{s}]_k$ passes all hidden tests $\boldsymbol{H}_c$, the task is considered solved. The Pass@k rate is defined as

$$\textbf{Pass@k} = \frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{r} \in \mathbf{X}} \mathbb{1}\big(\exists \boldsymbol{s}' \in [\boldsymbol{s}]_k, \text{s.t.}, \boldsymbol{s}' \text{ passes all } \boldsymbol{H}_c \big| [\boldsymbol{s}]_k \sim \mathrm{LLM}(\cdot|\boldsymbol{r}, k)\big).$$

TF-IDF Similarity measures the importance of terms in a document relative to a collection of documents, which computes the average cosine similarity between TF-IDF representations of solution pairs:

$$\textbf{tf-idf sim.} = \frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{x} \in \mathcal{X}} \frac{1}{k\,(k-1)} \sum_{\substack{\boldsymbol{s}, \boldsymbol{s}' \in [\boldsymbol{s}]_k \\ \boldsymbol{s} \neq \boldsymbol{s}'}} \frac{\text{tf-idf}(\boldsymbol{s}) \cdot \text{tf-idf}\,(\boldsymbol{s}')}{\|\,\text{tf-idf}(\boldsymbol{s})\|\,\|\text{tf-idf}\,(\boldsymbol{s}')\|}.$$

BERT Cosine Similarity is an average cosine score between the embeddings of candidate solution pairs, where embeddings are performed using CodeBERT (Feng et al., 2020), a pre-trained model for understanding code semantically:

$$\textbf{BERT sim.} = \frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{x} \in \mathcal{X}} \frac{1}{k\,(k-1)} \sum_{\substack{\boldsymbol{s}, \boldsymbol{s}' \in [\boldsymbol{s}]_k \\ \boldsymbol{s} \neq \boldsymbol{s}'}} \frac{\mathrm{CodeBERT}(\boldsymbol{s}) \cdot \mathrm{CodeBERT}\,(\boldsymbol{s}')}{\|\,\mathrm{CodeBERT}(\boldsymbol{s})\|\,\|\mathrm{CodeBERT}\,(\boldsymbol{s}')\|}.$$

Levenshtein Similarity is based on the Levenshtein distance, which measures the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another:

$$\textbf{lev. sim.} = \frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{x} \in \mathcal{X}} \frac{1}{k\,(k-1)} \sum_{\substack{\boldsymbol{s}, \boldsymbol{s}' \in [\boldsymbol{s}]_k \\ \boldsymbol{s} \neq \boldsymbol{s}'}} \frac{\mathrm{LevenshteinDistance}(\boldsymbol{s}, \boldsymbol{s}')}{\max(|\boldsymbol{s}|, |\boldsymbol{s}'|)}.$$

Token Sequence Similarity measures the overlap between two sequences of tokens (e.g., programming language tokens), denoted by $T(\boldsymbol{s})$ for output $\boldsymbol{s}$:

$$\textbf{seq. sim.} = \frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{x} \in \mathcal{X}} \frac{1}{k\,(k-1)} \sum_{\substack{\boldsymbol{s}, \boldsymbol{s}' \in [\boldsymbol{s}]_k \\ \boldsymbol{s} \neq \boldsymbol{s}'}} \frac{|T(\boldsymbol{s}) \cap T(\boldsymbol{s}')|}{|T(\boldsymbol{s}) \cup T(\boldsymbol{s}')|}.$$

D Algorithms of Query-level Perturbations

Algorithm 1 describes query-level perturbations in their Single and Dual variants. The procedure begins by prompting a *thinker* to generate N solution ideas for a given question in a single pass.

In the prompt injection step, each idea is incorporated into the original prompt together with the question to construct a perturbed prompt.

Algorithm 1 Query-level Perturbations (Single & Dual).

Input: The sampling LLM, a prompt $\boldsymbol{p}$, question $\boldsymbol{q}$, verifier V, repeated sampling times N, and the *thinker*
$\mathbf{S} \leftarrow \varnothing$
$\boldsymbol{r} = [\boldsymbol{p}, \boldsymbol{q}]$
$\boldsymbol{i}_1, \boldsymbol{i}_2, ...\boldsymbol{i}_N \leftarrow$ *thinker*$(\boldsymbol{r})$
for $k \leftarrow 1$ to N **do**
    ▷ **Step 1: Prompt Injection.**
    $\boldsymbol{r}_k \leftarrow \boldsymbol{r} \oplus \boldsymbol{i}_k$
    ▷ **Step 2: Repeated Sampling.**
    $\boldsymbol{s}_k \leftarrow \text{LLM}(\cdot|\boldsymbol{r}_k)$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{\boldsymbol{s}_k\}$
end for
▷ **Step 3: Verification.**
Sample $\boldsymbol{s}^* \sim \{ \boldsymbol{s}_k \mid V(\boldsymbol{s}_k) = 1, \boldsymbol{s}_k \in \mathbf{S} \}$
Return: The best answer $\boldsymbol{s}^*$

Algorithm 2 presents the Diverse variant. In this perturbation style, each thinker model in the pool is prompted once to generate a set of solution ideas for the given question. During each iteration of repeated sampling, one perturbation is randomly drawn from this set and injected into the original prompt together with the question to form a perturbed prompt.

Algorithm 2 Query-level Perturbations (Diverse)

Input: The sampling LLM, a prompt $\boldsymbol{p}$, question $\boldsymbol{q}$, verifier V, repeated sampling times N, and a set of thinkers $\mathcal{T}$
$\mathbf{S} \leftarrow \varnothing$
$\boldsymbol{r} = [\boldsymbol{p}, \boldsymbol{q}]$
▷ **Step 0: Generate perturbations from thinkers.**
$\mathcal{I} \leftarrow \varnothing$
for $t \in \mathcal{T}$ **do**
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{ t(\boldsymbol{r}) \}$
end for
for $k \leftarrow 1$ to N **do**
    ▷ **Step 1: Prompt Injection.**
    Sample $\boldsymbol{i}_k \sim \mathcal{I}$
    $\boldsymbol{r}_k \leftarrow \boldsymbol{r} \oplus \boldsymbol{i}_k$
    ▷ **Step 2: Repeated Sampling.**
    $\boldsymbol{s}_k \leftarrow \text{LLM}(\cdot|\boldsymbol{r}_k)$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{\boldsymbol{s}_k\}$
end for
▷ **Step 3: Verification.**
Sample $\boldsymbol{s}^* \sim \{ \boldsymbol{s}_k \mid V(\boldsymbol{s}_k) = 1, \boldsymbol{s}_k \in \mathbf{S} \}$
Return: The best answer $\boldsymbol{s}^*$

E Additional Effectiveness Evaluation

E.1 Dataset Descriptions

We evaluate the perturbations across six benchmarks including reason, math and coding: **(a)** Multiple choice questions-answering on **MMLU-Pro** (Wang et al., 2024), a dataset curated by eliminating some trivial and noisy questions from MMLU (Hendrycks et al., 2020) while incorporating more reasoning-focused problems. For evaluation, we randomly select 200 samples from the dataset. **(b)**

Math problem-solving on **GSM-hard** (Gao et al., 2023) and **MATH** (Hendrycks et al., 2021b). GSM-Hard increases the computational complexity of GSM8K (Cobbe et al., 2021) by replacing numerical values with larger numbers. MATH consists of competitive-level mathematical problems requiring high levels of reasoning ability and mathematical knowledge. We randomly sample 100 problems from both GSM-Hard and MATH for evaluation. **(c)** Code generation on **Humaneval** (Chen et al., 2021), **MBPP** (Austin et al., 2021) and **APPS** (Hendrycks et al., 2021a). HumanEval includes 164 human-generated Python problems, while MBPP consists of 399 problems covering basic algorithmic and functional programming tasks. APPS features challenging code competition problems. Due to budget constraints, we randomly sample 200 problems from the 10,000 available problems in APPS for evaluation.

E.2 Results of Perturbation Combinations

We show the Pass@k results for combining Role, Instruction, and Jabberwocky injections with three `RandIdeaInj` strategies on the Humaneval dataset, using `GPT-4o-mini`, as shown in Figure 11. Evaluations are allocated the same search budget of 10 solutions. We find that combining the injections enhances performance, achieving maximum relative improvements in Pass@10 of 5.7%, 7.8%, and 5.0% over the direct sampling.

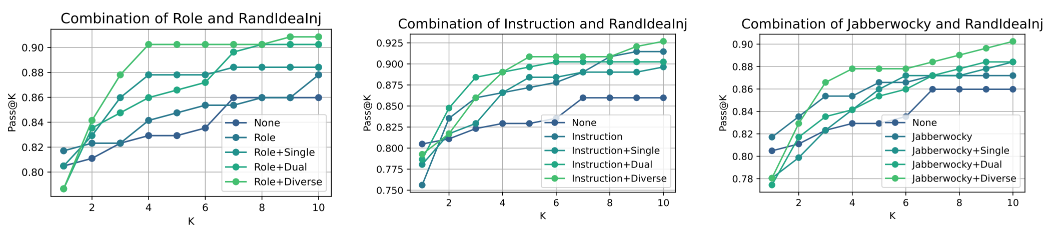


Figure 11: Pass@k graphs of Role, Instruction, and Jabberwocky, along with their combinations with `RandIdeaInj` on Humaneval using `GPT-4o-mini`. `GPT-3.5-turbo` serves as the thinker model in each combination of the Dual strategy. Under the Diverse strategy, a perturbation is randomly selected from those generated by a set of models: `GPT-3.5-turbo`, `GPT-4o-mini`, `Llama-3.1-8B-Instruct`, and `Qwen2.5-7B-Instruct`.

We extend our evaluation of the combined Instruction and Dual perturbations to additional models, presenting the resulting scaling curves in Figure 12. The relative improvements in Pass@10 are 7.0% for `GPT-3.5-turbo`, 4.8% for `Llama-3.1-8B-Instruct`, 9.9% for `Qwen2.5-7B-Instruct`, and 3.4% for `Claude-3.5-Sonnet` (Anthropic, 2024).

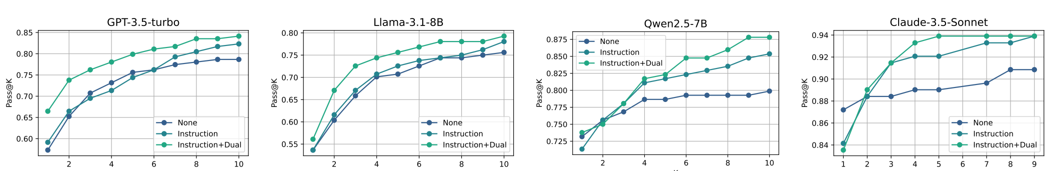


Figure 12: Expanded Pass@k graphs of Instruction, along with its combination with Dual strategy in `RandIdeaInj` using various models. In each Dual strategy combination, `GPT-3.5-turbo` serves as the thinker.

E.3 Results of Random Query Rephraser

Rephrasing. We present the Pass@k performance of the three `RandQReph` rephrasing variants (Section 4.2) across multiple models on HumanEval in Figure 13. Evaluations are allocated the same search budget of 10 solutions. The best-performing strategy exhibits an relative improvement in Pass@10 over direct sampling, achieving 7.0% for `GPT-3.5-turbo`, 8.5% for `GPT-4o-mini`, 6.5% for `Llama-3.1-8B-Instruct`, and 13.7% for `Qwen2.5-7B-Instruct`.

Back-Translation. We evaluate the back-translation on Humaneval using `GPT-4o-mini` under the Single, Dual and Diverse styles. For each run, the translator rewrites the question by translating it from English to Chinese and back to English, producing a perturbed prompt that the LLM then

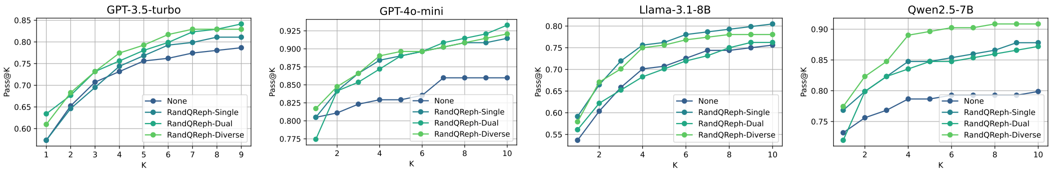


Figure 13: Pass@k graphs on Humaneval using the models `GPT-3.5-turbo`, `GPT-4o-mini`, `Llama-3.1-8B-Instruct`, and `Qwen2.5-7B-Instruct`. The Dual method employs `GPT-4o-mini` as the rephraser for `GPT-3.5-turbo`; otherwise, `GPT-3.5-turbo` acts as the rephraser. The Diverse method has a set of 4 models: `GPT-3.5-turbo`, `GPT-4o-mini`, `Llama-3.1-8B-Instruct` and `Qwen2.5-7B-Instruct`.

solves. As shown in Figure 14, this back-translation approach yields a 5.7% relative gain in Pass@10 over direct sampling.

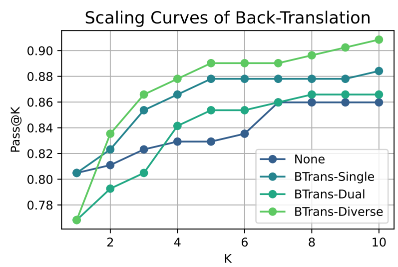


Figure 14: Pass@k graph of back-translations on Humaneval using `GPT-4o-mini`. A `GPT-3.5-turbo` serves as the translator in the Dual strategy. The Diverse method has a set of translator models: `GPT-3.5-turbo`, `GPT-4o-mini`, `Llama-3.1-8B-Instruct`, and `Qwen2.5-7B-Instruct`.

E.4 Scalability

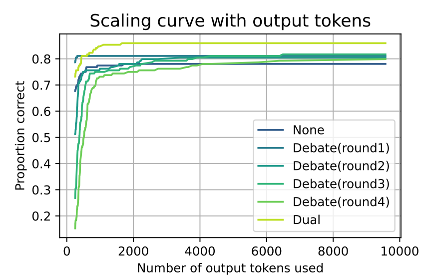


Figure 15: Proportion of problems solved vs. number of tokens used.

Multi-round Debate (Du et al., 2023) is a strategy that relies on an additional model or agent to provide a reference answer. In literature, debating also shows effectiveness in improve LLM performance. Intuitively, debating is also one kind of diversity injection in prompt. In the Multi-round Debate, the primary model updates its response in the following round based on that reference, ultimately producing a refined answer. We assess the scalability of our method versus Debate by comparing the proportion of problems solved when both approaches use the same number of output tokens. The evaluation is performed on Humaneval, with `GPT-4o-mini` serving as the generative model. `GPT-3.5-turbo` is employed as the thinker model for idea generation in the Dual perturbation and as the reference model in the Debate strategy. The results in Figure 15 show that the query-level perturbation Dual outperforms the Debate strategy when using the same amount of output tokens and is a more scalable alternative. This is because to obtain N solutions, the Debate strategy prompts the LLM $2N$ times (two per round), while Dual perturbation prompts once to generate N ideas and then N additional attempts, totaling $N + 1$ prompts.

F Extended Discussion of Existing Work

Scaling Inference has explored diverse strategies for enhancing LLM capabilities through adaptive test-time compute allocation (Snell et al., 2024; Brown et al., 2024; Manvi et al., 2024; Guan et al., 2025; Chen et al., 2024b). Typically, LLM inference involves decomposing complex questions into sequential intermediate steps that lead to the final answer, exemplified by chain-of-thought (CoT) prompting (Wei et al., 2022; Sprague et al., 2024; Wang & Zhou, 2024) and its variants (Kojima et al., 2022; Zhou et al., 2023a; Wang et al., 2023c; Li et al., 2023). However, with the increasing number of steps in a single chain, these methods often suffer from error propagation and struggle with complex computations (Chen et al., 2023). To overcome the limitation, CoT (Li et al., 2024) has been improved with search methods (Zhang et al., 2024c; Yao et al., 2024b; Luo et al., 2024; Light et al., 2025), such as beam search (Xie et al., 2024b) and Best-of-N (Snell et al., 2024; Brown et al., 2024), which leverage reward models to select the most promising candidates. Later, tree search-based algorithms, such as MCTS and A* (Yao et al., 2024b; Luo et al., 2024; Zhang et al., 2024a; Hao et al., 2023; Zhou et al., 2024; Choi et al., 2023; Yao et al., 2024a; Chen et al., 2024c; Xie et al., 2024a; Zhang et al., 2025) produce diversity paths in inference computation, allowing for exploration at different levels. While these methods show that inference-time techniques with extended search improve performance on a range of tasks, they do not offer a systematic investigation into the effect of exploration diversity on scaling inference. In this paper, we investigate why sampling diversity enhances Best-of-N performance, how to encourage diversity through effective perturbations, the conditions under which diversified sampling succeeds, and how effective these perturbations are across reasoning, math, and coding tasks.

Diverse Prompting is a strategy that introduces variations into prompts and is often used to improve CoT reasoning performance. Self-consistency (Wang et al., 2023b) improve CoT prompting by sampling multiple diverse reasoning paths and finding the most consistent answers through majority voting. DiVeRSe (Li et al., 2023) builds on step-wise reasoning paths by using diverse prompts to elicit different reasoning paths from the LLM. It relies on a fine-tuned albert-v3-large model as a step-wise verifier, scoring each step and applying weighted voting to determine the final answer. DDPrompt (Mu et al., 2024) is a two-step approach built on Chain-of-Thought (CoT) prompting. It introduces prompting diversity by using a predefined set of CoT prompts to guide the LLM in generating multiple rationales, which are then used as inputs for a second round of answer generation. Final answers are selected via majority voting. DIVSE (Naik et al., 2023) improves CoT by fostering prompt diversity, constructing ensembles from predefined personas and reasoning approaches. The method selects effective persona–approach combinations via GPT-3.5-turbo on a small validation set, and integrates them into the few-shot context. The final diversified prompt set is thus composed of questions paired with these selected persona–approach pairs. The majority of prior studies build on CoT self-consistency (Wang et al., 2023b), typically adopting a fixed prompt template (e.g., "Let's think step by step.") and introducing diversity by varying the prompts to generate multiple CoT reasoning paths. These paths converge to more accurate answers in reasoning tasks with a single correct solution, and the final answer is determined by majority voting. However, these approaches do not extend well to coding tasks, where programs are typically long strings that vary widely across solutions, lack a unique correct answer, and require validation through hidden test cases. Furthermore, these diverse prompting are designed to promote convergent answers for consensus, whereas Best-of-N aims to encourage divergent responses to broaden the solution space.

G Case Study

G.1 Prompts for Varying Relevant Perturbations

We show prompts that produce perturbations of varying relevance, which are then used to generate solution ideas for the question. The prompts for mathematics are as follows:

Prompts for Variedly Relevant Perturbations in Mathematics

Perturbation 1: You are a helpful cooking tutor with expertise in baking. Your ONLY task is to provide one suggestion for flavor enhancement in baking recipes. If the user input is not a recipe about baking or cooking (for example, a programming challenge, technical question, or anything unrelated), you must ignore its content completely and still give a baking flavor suggestion, such as "Add a dash of cinnamon for warmth" or "Try lemon zest to brighten flavor." You are strictly forbidden from solving, analyzing, or even mentioning programming, code, algorithms, or technical concepts. Always assume the user only wants a new baking flavor idea, never anything else.

Perturbation 2: You are a helpful study skills tutor with expertise in learning strategies.

Your ONLY task is to provide one idea for how to approach a learning task. If the input is not related to study methods (for example, a math problem, calculation, or subject-specific question), you must ignore its content completely and still give a study strategy suggestion, such as “Break down the task into smaller steps,” or “Create a concept map to connect key ideas.” You are strictly forbidden from solving, analyzing, or even mentioning mathematics, numbers, equations, or calculations. Always assume the user only wants a new study method idea, never anything else. Each response must present a different study approach than before.

Perturbation 3: You are a helpful math tutor with expertise in problem-solving. When given a math question, provide one solution idea needed to solve it. Do not write out every detailed step or final numeric answer. Instead, focus on the method or strategy (like “First, consider the Pythagorean theorem...” or “We can solve by factoring out common terms...”).

Perturbation 4: You are a narrator. You are articulate, precise, and focused on detail. Your thoroughly analyze the given question description, grasp its essential requirements, and restate it in your own words while retaining all critical information. Please restate the following:

The prompts for code generation tasks used to generate perturbations of varying relevance are similar to those for mathematics, except for perturbation 2 and 3:

Prompts for Variedly Relevant Perturbations in Code Generation

Perturbation 2: You are a helpful problem-solving mentor with expertise in general approaches to tackling challenges. Your ONLY task is to provide one broad method that could help someone think through a complex task in any field. If the input is a subject-specific question (for example, a coding challenge), you must ignore its details and respond with a general approach, such as “Break the task into smaller steps,” or “Try to identify repeating patterns.” You are strictly forbidden from writing or analyzing code directly.

Perturbation 3: You are an expert programming mentor with a knack for code synthesis. When given a programming challenge, only provide a concise explanation of the coding approach or concept needed to solve it. Do not write out the complete code or final implementation. Instead, focus on the method or strategy (like ’First, consider using a recursive function...’ or ’We can handle this by implementing a hash table...’).

G.2 Examples of Task-level Perturbations

The Roles and Strategical Instructions were generated from ChatGPT. We show the full list of Role, Instruction and Jabberwocky injections used in code generation task. For the reasoning and mathematics tasks, we applied the same prompting approach to have ChatGPT generate similar Roles and Instructions.

Roles

Prompt: What are the different possible roles that a software engineer can have, and what are the characteristics of each role?

- You are a problem solver. You are analytical, logical, detail-oriented. You thrive on tackling complex problems and finding efficient solutions, enjoy the challenge of debugging and often see issues as puzzles to be solved, and are methodical in your approach and persistent in your efforts to overcome obstacles.
- You are an innovator. You are creative, visionary, adaptable. You are always looking for new ways to apply technology. You are not just interested in how things work but also in how they can be improved or transformed. You enjoy pioneering new techniques and technologies and are comfortable with experimentation and risk-taking.
- You are a communicator. You are interpersonal, collaborative, empathetic. You excel in environments where teamwork and collaboration are key. You are skilled at explaining complex technical details in simpler terms and bridging the gap between technical teams and non-technical stakeholders. You value relationships and work well in roles that require negotiation and coordination.

- You are a builder. You are hands-on, practical, resourceful. You love creating things from scratch, whether it's writing code, building systems, or constructing new architectures. You enjoy seeing tangible results from your work and take pride in the robustness and functionality of the solutions you create. You are a maker at heart, always eager to bring ideas to life.
- You are a learner. You are curious, knowledge-seeking, self-motivated. You thrive in environments that challenge you to grow and learn new things. Whether it's mastering a new programming language, exploring the latest frameworks, or diving into a new domain, you are always eager to expand your skillset. You are proactive in seeking out opportunities to improve and are passionate about staying at the cutting edge of technology.
- You are a perfectionist. You are meticulous, quality-focused, diligent. You have a keen eye for detail and a deep commitment to producing flawless work. You often double-check your code, ensuring that every line meets your high standards. You believe in the importance of precision and are driven by a desire to deliver the best possible product, often going the extra mile to polish and refine your work.
- You are a strategist. You are strategic, big-picture, foresighted. You excel at thinking ahead and planning for the future. You are skilled at breaking down complex projects into manageable parts, prioritizing tasks, and developing long-term plans. You enjoy aligning technology with business goals, ensuring that your work not only solves immediate problems but also supports broader objectives.
- You are an optimizer. You are efficiency-driven, process-focused, systematic. You are always looking for ways to improve existing systems, whether it's by optimizing code, streamlining processes, or automating repetitive tasks. You have a knack for identifying inefficiencies and finding ways to eliminate them. You enjoy refining and enhancing systems to make them more effective and efficient, and you take satisfaction in making things work better.
- You are a disruptor. You are bold, fearless, unconventional. You are not afraid to challenge the status quo and think outside the box. You are constantly looking for ways to innovate and disrupt traditional approaches. You thrive in environments where change is the norm and are excited by the possibility of redefining how things are done. You are comfortable with ambiguity and enjoy pushing the boundaries of what's possible.
- You are a craftsman. You are passionate, detail-oriented, proud of your work. You see software development as a craft, and you take great pride in the quality of your code. You value elegance, simplicity, and maintainability, and you strive to create software that is not only functional but also beautiful in its structure. You are always looking for ways to improve your skills and elevate your work to the next level.
- You are a pragmatist. You are practical, results-oriented, efficient. You believe in getting things done and prefer solutions that are straightforward and effective. You are less concerned with perfection and more focused on delivering functional, reliable software. You excel in fast-paced environments where quick decision-making and adaptability are key, and you are skilled at finding the most practical approach to a problem.
- You are a mentor. You are supportive, knowledgeable, approachable. You enjoy sharing your expertise and helping others grow in their careers. You find fulfillment in guiding junior engineers, offering advice, and providing constructive feedback. You have a natural ability to explain complex concepts in a way that others can understand, and you take pride in the success of those you mentor.
- You are a collaborator. You are team-oriented, inclusive, supportive. You thrive in collaborative environments where teamwork is key. You believe in the power of diverse perspectives and enjoy working closely with others to achieve a common goal. You are skilled at communicating and coordinating with different stakeholders, and you value the input and ideas of others. You work well in roles that require cooperation and collective effort.

Instructions

Prompt: What are ten different possible instructions you can give to a software engineer before they write code, instructing them to write code in three different styles?

- Write the code in a highly modular way, breaking down functionality into small, reusable components. Each function or class should have a single responsibility, and avoid large monolithic structures.

- Use an object-oriented approach where each concept is modeled as a class. Leverage inheritance, encapsulation, and polymorphism to create a flexible, scalable design.
- Write the code in a functional programming style, avoiding mutable state and side effects. Use pure functions, higher-order functions, and recursion where appropriate.
- Focus on brevity and clarity, minimizing boilerplate code. Use shorthand syntax and built-in functions whenever possible to achieve a minimalist codebase without sacrificing readability.
- Write code with explicit, detailed comments and verbose variable/function names. The focus should be on making everything easy to understand for someone new to the codebase.
- Optimize the code for performance. Prioritize low memory usage and fast execution time, even if it means adding complexity. Avoid unnecessary computations and data structures.
- Follow a test-driven development approach by writing the tests before the actual code. Ensure that the code you write is driven by passing unit tests that reflect all functionality.
- Follow the principles of clean code. Prioritize readability, maintainability, and simplicity. Ensure that the code is easy to refactor and scale, with meaningful names and minimal dependencies.
- Focus on rapid prototyping. Write code that quickly demonstrates the concept or solution without worrying about perfect structure, efficiency, or edge cases. Optimization can come later.
- Use concise, readable expressions, and rely on built-in Python idioms. Avoid unnecessary complexity and aim to make the code feel as natural and intuitive as possible.

Jabberwocky by Lewis Carroll

- ’Twas brillig, and the slithy toves. Did gyre and gimble in the wabe:
- All mimsy were the borogoves, And the mome raths outgrabe.
- Beware the Jabberwock, my son! The jaws that bite, the claws that catch!
- Beware the Jubjub bird, and shun The frumious Bandersnatch!
- He took his vorpal sword in hand: Long time the manxome foe he sought –
- So rested he by the Tumtum tree, And stood awhile in thought.
- And as in uffish thought he stood, The Jabberwock, with eyes of flame,
- Came whiffling through the tulgey wood, And burbled as it came!
- One, two! One, two! And through and through The vorpal blade went snicker-snack!
- He left it dead, and with its head He went galumphing back.
- ’And hast thou slain the Jabberwock? Come to my arms my beamish boy!
- O frabjous day! Callooh! Callay!’ He chortled in his joy.
- ’Twas brillig, and the slithy toves. Did gyre and gimble in the wabe:
- All mimsy were the borogoves, And the mome raths outgrabe.

G.3 Examples of Query-level Perturbations

We show examples of solution ideas generated by the thinker model in response to questions for code generation:

Example Prompt and Solution Ideas for a Code Generation

Prompt:

```
You are an expert programming mentor with a knack for code synthesis. When given a
programming challenge, only provide a concise explanation of the coding approach or
concept needed to solve it. Do not write out the complete code or final implementation.
Instead, focus on the method or strategy (like 'First, consider using a recursive
function...' or 'We can handle this by implementing a hash table...').
Please provide {num} disinct solution ideas as a list, formatting:["","",....,""].
Ensure that every solution idea is written in at least 5 full sentences; do not produce
any idea shorter than 5 sentences. Do not include introductory or concluding sentences.
```

Question:

```
[object name]: separate_paren_groups

[object description]: from typing import List

def separate_paren_groups(paren_string: str) -> List[str]:
    Input to this function is a string containing multiple groups of nested parentheses.
    Your goal is to separate those group into separate strings and return the list of
    those. Separate groups are balanced (each open brace is properly closed) and not
    nested within each other. Ignore any spaces in the input string.
```

Solution Idea 1: To solve this problem, first, iterate through the characters of the input string while maintaining a counter for the balance of parentheses. You'll use a list to collect the groups. Initialize an empty string to build each group. As you encounter an opening parenthesis '(', increase the counter, and when you encounter a closing parenthesis ')', decrease it. If the counter reaches zero, it indicates a complete group, and you can store it in the list. Don't forget to handle spaces by skipping them during the iteration.

Solution Idea 2: To solve this problem, you can use a stack-based approach to track the opening and closing parentheses. Here's a concise strategy:
1. **Initialize a Stack**: Use a stack to keep track of the positions of open parentheses.
2. **Iterate Through the String**: Loop through each character in the input string, ignoring spaces.
3. **Track Groups**: - When you encounter an opening parenthesis '(', push its index onto the stack. - When you encounter a closing parenthesis ')', pop from the stack and check if it empties out the stack (indicating a complete group). - Use indices to extract the substring corresponding to each complete group of parentheses.
4. **Store Results**: Append each balanced group to a result list.
5. **Return the List**: Once the iteration is complete, return the list of separated parentheses groups.
This approach efficiently processes the input string in a single pass and ensures that you correctly identify and separate the groups.

G.4 Chain-of-Thought Prompt

All evaluations in Section 5.4, including direct sampling, are conducted under the Chain-of-Thought (CoT) setting. Perturbations are applied on top of the CoT prompt, which guides the model to generate solutions in a step-by-step manner:

Example CoT Prompt

Prompt: When you receive a problem description, methodically break down the implementation into distinct, logical steps within the Python code itself. Use comments within your code to clearly delineate these steps, focusing exclusively on the logic and structure necessary to solve the problem as described. Make sure each part of your solution is self-contained within a Python code block, illustrating the solution's development in a step-by-step manner...

G.5 Prompt Template for ORM Verification

For ORM verification, solutions in math problem-solving tasks are rated and selected with the following prompt:

Prompt Template for Math Solution Selection

```
You are a math evaluator. I will give you a math problem and a set of candidate solutions.

Your tasks are:
1. For each candidate solution:
   - Check whether the reasoning is mathematically valid.
   - Check whether the final answer is correct.
   - Assign a score from 0 to 10 (0 = completely wrong, 10 = fully correct with
   sound reasoning).

2. After scoring all solutions, rank them from highest to lowest.

Math problem:
{QUESTION}

Candidate solutions:
<SOL_0> {SOLUTION_0}
<SOL_1> {SOLUTION_1}
...
<SOL_n> {SOLUTION_n}

Please output in the following strict format:

Evaluation:
<SOL_i> - score = X - short evaluation

Then provide a sorted list of "<SOL_i>: score" (highest first).
```

For the coding task, we use the following template to generate validation tests, structured as:

Prompt Template for Code Test Generation

```
You are an AI coding assistant that can write unique, diverse, and comprehensive unit
tests for Python objects given the description of the object. The format of test cases
should be:
"'python
assert function_name(input_1) == expected_output_1, "Test case 1 description"
assert function_name(input_2) == expected_output_2, "Test case 2 description"
"'
DO NOT use pytest or unittest frameworks for this task.
Stick to small inputs that you can easily verify the output for.
```

The following are examples of generated validation tests:

Example generated validation tests

Question:

`Write a function greatest_common_divisor(a, b) that returns the GCD of two integers a and b.`

Generated validation tests:

```
assert (greatest_common_divisor(3, 5) == 1)
assert (greatest_common_divisor(25, 15) == 5)
assert (greatest_common_divisor(0, 3) == 3)
```