BENCHMUTATOR: AGENT MUTATED BENCHMARKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Benchmark data contamination undermines the evaluation of large language models. Transforming the instances of an existing benchmark is a lower-cost alternative to constructing a new one, but current transformation methods have two limitations. First, most are task-specific and do not transfer across benchmarks. Second, their effectiveness is commonly inferred from the performance degradation they induce, although degradation can also result from invalid instances, increased difficulty, or variance across generation runs. We introduce BENCHMUTATOR, a task-adaptive multi-agent framework that calibrates transformation rules from benchmark examples and coordinates specialized agents to mutate inputs, update reference answers, and validate the resulting instances. We evaluate BENCHMUTATOR on six benchmarks spanning code synthesis, program repair, mathematical reasoning, and text-based question answering, using four model families. To control for contamination, we evaluate both base models and models deliberately fine-tuned on benchmark instances. We assess four properties, namely transformation diversity, human-assessed validity, consistency across generation runs, and model response to mutation. BENCHMUTATOR produces broader lexical and semantic variation than existing mutation approaches. A human audit reports 91.7% acceptance of transformed instances, with a near-zero mean shift in perceived difficulty. On GSM8K, BENCHMUTATOR yields lower run-to-run performance variation than the dynamic baselines Self-Evolving and DyVal. Mutation effects vary across tasks, generators, and model families, including for models fine-tuned on benchmark instances. These results indicate that performance degradation alone is not a reliable indicator of contamination mitigation, and that benchmark transformations should be assessed jointly on validity, diversity, and evaluation consistency. BENCHMUTATOR makes this assessment practical across task families.

1 INTRODUCTION

Benchmark scores are useful measures of generalization only when success cannot be explained by prior exposure to the evaluation instances. For large language models (LLMs), benchmark data contamination (BDC), the inclusion of evaluation data in training corpora, undermines this distinction (Sainz et al., 2023; Magar & Schwartz, 2022). Exposure can inflate performance, making it difficult to determine whether a model has learned transferable capabilities or exploits information encountered during training (Deng et al., 2024). One response is to construct new benchmarks. LatestEval, LiveBench, and LiveCodeBench pursue this strategy through evaluation data drawn from recent texts, information sources, and programming contests (Li et al., 2024; White et al.; Jain et al., 2024). However, maintaining fresh benchmarks requires recurring effort to collect suitable material, construct and validate questions and reference answers, and maintain evaluation infrastructure. Replacing evaluation instances also creates a comparability challenge where changes in scores may reflect changes in the benchmark's coverage or difficulty rather than changes in model capability. Moreover, newly released benchmarks may themselves be incorporated into the training data of subsequent models, creating a recurring cycle of benchmark construction, contamination, and replacement (Roberts et al., 2023). Benchmark transformation offers a practical alternative by generating variants of existing evaluation instances while reusing their task specifications, reference solutions, and evaluation procedures (Zhu et al., 2024c; Cao et al., 2024). This reuse offers a path to refreshing evaluation data while retaining a connection to the original benchmark. Yet transformation does not automatically preserve evaluation validity. Superficial edits may leave exploitable patterns intact, whereas more substantial changes may alter the required reasoning, invalidate reference answers or tests, or shift task

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

difficulty (Sun et al., 2025). A second challenge concerns how transformation effectiveness is assessed. Input perturbations have been used to investigate memorization and robustness (Djiré et al., 2025; Wang et al., 2022), but a drop in performance does not, by itself, demonstrate successful contamination mitigation (Sun et al., 2025). The same drop may arise from increased difficulty, an invalid reference answer, or a model's sensitivity to unfamiliar formulations. Benchmark transformation therefore requires evaluating both the quality of the transformed instances and the consistency and interpretation of the model responses they induce. This motivates the question that drives this paper. *How can benchmark transformation adapt to different tasks while producing diverse, valid variants that support consistent evaluation?*

In this paper, we introduce BENCHMUTATOR, a multi-agent framework that derives benchmark-specific transformation strategies from examples without requiring a predefined transformation taxonomy. In the **calibration phase**, an *Analysis Agent* identifies task constraints from a benchmark subset, and a *Rule Agent* proposes transformation strategies that are refined through a calibration probe. In the **application phase**, a *Transformation Agent* instantiates the calibrated rules for each sample, an *Output Update Agent* revises reference answers and tests when necessary, and task-specific checkers assess candidate validity. Calibration establishes a shared set of transformation constraints before full-benchmark generation, while instance-level application allows those strategies to accommodate individual problems.

We evaluate BENCHMUTATOR on six benchmarks spanning code generation, program repair, mathematical reasoning, and text-based question answering, using four target model families (DeepSeek, Gemma, Llama, and Qwen). In an audit of 30 distinct instances assessed by two annotator pools, acceptance reaches 91.7% across pool-level majority judgments, with a mean perceived difficulty shift of +0.04 on a scale from -2 to +2. We additionally evaluate models deliberately fine-tuned on benchmark instances to study transformation under controlled additional exposure. This experiment measures the effects of known exposure without assuming that the original models' pretraining data were contamination-free. Transformation effects vary across tasks, generators, and target model families, including after this fine-tuning. These findings caution against interpreting transformation sensitivity as a model-independent measure of memorization and motivate assessing contamination-related behavior jointly with validity, diversity, and evaluation consistency. Our primary contributions are as follows. **(i)** A task-adaptive transformation framework. BENCHMUTATOR combines benchmark-level rule calibration with instance-level transformation and coordinated answer and test updates, supporting multiple task families without a hand-coded transformation taxonomy. **(ii)** An empirical assessment of transformation quality. Across six benchmarks and four model families, we compare transformation approaches along complementary dimensions of diversity, validity, and performance consistency. **(iii)** A contamination-resistance analysis. Experiments with benchmark-fine-tuned models characterize variation in transformation effects across model families and the limits of performance degradation as evidence of contamination mitigation.

2 BACKGROUND

BDC Detection. BDC detection methods approach contamination through membership inference and memorization analysis. Membership inference uses signals such as model confidence, loss, or neighborhood comparisons to estimate whether an instance appeared during training (Shokri et al., 2017; Yeom et al., 2018; Mattern et al., 2023). Memorization analysis examines behaviors such as training-text reproduction or responses to targeted prompts (Carlini et al., 2023; Golchin & Surdeanu, 2024). The reliability of these signals depends on model access and evaluation assumptions, limiting their interpretation as definitive evidence of contamination (Duan et al., 2024). Benchmark transformation provides a complementary perspective by examining how performance changes when evaluation instances are modified. However, transformation sensitivity may also reflect robustness or task difficulty; it does not uniquely identify memorization (Wang et al., 2022; Sun et al., 2025).

Mitigation Strategies. Existing approaches refresh evaluation data through three broad strategies.
Fresh-data collection. LatestEval, LiveBench, and LiveCodeBench construct evaluations from recent material to reduce overlap with model training data (Li et al., 2024; White et al.; Jain et al., 2024). Maintaining freshness requires recurring collection and validation, while released instances may enter subsequent training corpora (Roberts et al., 2023).
Synthetic generation. Frameworks such as BenchAgents automate benchmark construction through

Approach	Description	Task				
		Program Repair	Code Synthesis	General Knowledge	Math Reasoning	MCQ
CleanEval (Zhu et al., 2024c)	Uses neural paraphrase and back-translation to generate semantically equivalent reformulations of contaminated benchmark samples, filtering candidates by a semantic quality score.	✗	✗	✓	✓	✓
ITD (Zhu et al., 2024b)	Detects leaked benchmark samples at inference time and rewrites them to reduce contamination-induced performance inflation, without changing task difficulty.	✗	✗	✓	✓	✓
CodeMorph (Rao et al., 2025)	Applies semantics-preserving structural code transformations to existing benchmark problems to mitigate data leakage in LLM assessment.	✓	✗	✗	✗	✗
DyCodeEval (Chen et al., 2025)	Uses multiple agents to dynamically rewrite the natural-language context of code reasoning problems, generating semantically equivalent variants without altering core logic, to benchmark Code LLMs under contamination risks.	✗	✓	✗	✗	✗
DyVal (Zhu et al., 2024a)	Dynamically generates graph-structured reasoning problems with controllable complexity for maths, logical reasoning, and algorithmic tasks, avoiding reliance on static benchmark instances.	✗	✗	✗	✓	✗
Self-Evolving (Wang et al., 2024)	A multi-agent framework that applies six reframing operations to existing benchmark instances, generating evolved variants that probe diverse sub-abilities and robustness to data noise.	✗	✗	✗	✓	✗
CodeCleaner (Cao et al., 2024)	A toolkit that applies diverse semantics-preserving code transformations to training and evaluation data to mitigate data contamination in code LLM assessment.	✓	✗	✗	✗	✗
BENCHMUTATOR	Agent-selected, task-adaptive transformation: each sample is assigned a rule from a task-specific set, checker-gated for correctness.	✓	✓	✓	✓	✓

Table 1: Task coverage of prior benchmark transformation approaches versus BENCHMUTATOR. Marks for prior work reflect the task scope stated in the cited papers.

coordinated agents (Butt et al., 2024). Generation reduces dependence on newly collected source material, but synthetic origin alone does not establish freedom from contamination or comparability with existing evaluations (Sun et al., 2025).

Benchmark transformation. Transformation methods reuse existing instances, modifying their wording or structure while aiming to preserve the intended task. Approaches range from paraphrasing and code transformations to agentic evolution (Zhu et al., 2024c; Cao et al., 2024; Wang et al., 2024). Their main challenge is to introduce meaningful variation without invalidating reference answers or altering difficulty (Sun et al., 2025). Table 1 summarizes individual approaches and their task coverage.

BENCHMUTATOR follows the transformation strategy, deriving benchmark-specific rules through calibration and instantiating them per sample. It combines this adaptation with answer and test updates and task-specific validation, as detailed in Section 3.

3 BENCHMUTATOR

BENCHMUTATOR transforms benchmarks through two phases. Calibration derives benchmark-specific transformation rules from a sampled subset, and application instantiates those rules for individual examples. This separates rule design from execution without requiring a predefined transformation taxonomy. Figure 1 summarizes the pipeline.

3.1 PHASE 1: CALIBRATION

Sampling and task analysis. We determine the calibration sample size using Cochran's finite-population formula (Cochran & Cochran, 1977) as a sampling heuristic that accounts for benchmark size while limiting calibration cost. The required sample size approaches a constant as the benchmark grows. We additionally limit the calibration subset to at most 400 instances to control computational

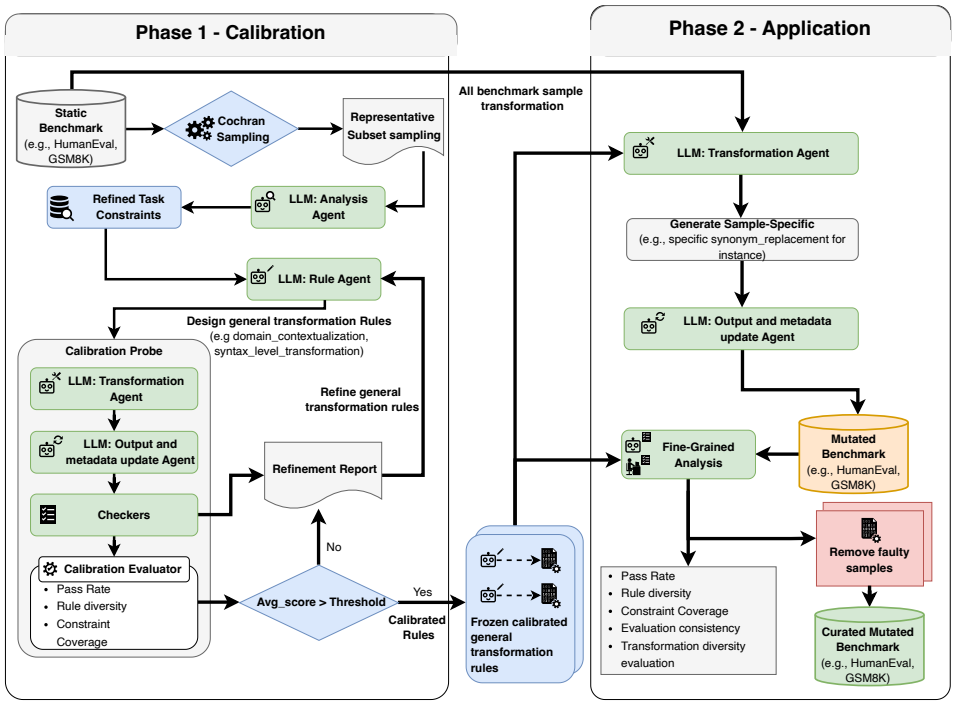

Figure 1: **Overview of the BENCHMUTATOR pipeline.** Phase 1 (calibration) derives general transformation rules from a statistically representative sample via the Cochran formula, refining them through a calibration probe with automatic feedback. Phase 2 (application) applies frozen agents to every benchmark instance, with the Output Update Agent ensuring input/output/test coherence.

cost. The **Analysis Agent** uses the sampled instances to identify explicit task constraints and define the Task Input Space (TIS), which comprises inputs satisfying the task requirements. For example, a program repair task requires a buggy program as input, so a natural-language passage without a program to repair falls outside its TIS.

Rule synthesis and refinement. Given these constraints and the TIS, the **Rule Agent** proposes reusable transformation strategies, such as meaning-preserving synonym replacement or contextual reframing. For example, for a code generation benchmark, the Rule Agent might produce rules such as SYNONYM_REPLACEMENT (replacing key verbs and nouns with meaning-preserving synonyms), DOMAIN_CONTEXTUALIZATION (adding a non-technical scenario frame), or PHRASING_RESTRUCTURING (reordering sentence clauses while preserving all semantic content). Then the Transformation Agent takes the SYNONYM_REPLACEMENT and instantiates it as REWORDING_OCCURRENCE_REMOVAL for a specific string manipulation problem, with a documented mapping of replaced terms and a complexity preservation rationale. A calibration probe applies the transformation, output-update, and checking procedure described in Section 3.2 to the sampled instances. If the checker pass rate is below 0.8, a report of the failures is returned to the Rule Agent for revision. The resulting calibrated rules are held fixed during application.

3.2 PHASE 2: APPLICATION

Transformation and output alignment. For each benchmark instance, the **Transformation Agent** instantiates one or more calibrated rules to produce a modified input. It records the applied changes and the rationale for preserving task complexity and retaining or updating the reference answer. The **Output Update Agent** then revises reference outputs and tests when required, such as after changes to numerical values or function signatures. Otherwise, the original outputs and tests are retained.

Task-specific checking. A **Checker** assesses each candidate according to its task modality. For code synthesis, it executes the reference solution against the original or updated tests. For program

repair, it runs the original test suite to check that the transformed program remains faulty. For natural language tasks, an LLM judge assesses whether the reference answer is correct for the modified input. Rejected candidates are returned to the Transformation Agent for revision; an outer loop retries the application phase up to three times.

4 EXPERIMENT SETUP

Research Questions. We structure our evaluation around four research questions. ① *Transformation diversity*: does BENCHMUTATOR produce transformations that are more diverse than those of prior approaches? ② *Mutated-benchmark coherence*: do BENCHMUTATOR's mutated benchmarks remain coherent and preserve the difficulty of the reference benchmark? ③ *Performance shift*: how does model performance shift under transformation, for both BENCHMUTATOR and competing approaches, and how consistent is this shift across independent runs of the same generation process? ④ *Contamination resistance*: to what extent does BENCHMUTATOR disclose the effect of data contamination?

Tasks and Datasets considered. We evaluate on six static benchmarks spanning four task types: *math reasoning* (GSM8K (Cobbe et al., 2021)), *code synthesis* (HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021)), *general knowledge* (TruthfulQA (Lin et al., 2021) and LatestEval (Li et al., 2024), denoted BBC-latest in tables and figures) and *program repair* (QuixBugs (Lin et al., 2017)).

Models Considered. We evaluate the benchmarks generated by BENCHMUTATOR using four target models, DeepSeek-V3.2 (Liu et al., 2025), Gemma-2 9B (Team et al., 2024), Llama-3.1 8B (Dubey et al., 2024), and Qwen-2.5 7B (Qwen et al., 2025). We use DeepSeek-V3.2 and Mistral 24B (Rastogi et al., 2025) as the underlying models for the agents, selecting generators of different sizes to examine how transformation quality varies with generator choice.

Baselines. We compare BENCHMUTATOR with transformation and generation methods discussed in Section 2. DyCodeEval (Chen et al., 2025) and Self-Evolving-Benchmark (Wang et al., 2024) transform the same source instances as BENCHMUTATOR, on HumanEval and MBPP for the former and on GSM8K for the latter, enabling direct comparisons of transformation quality. DyVal (Zhu et al., 2024a) and BenchAgents (Butt et al., 2024) provide additional comparisons with dynamically generated benchmarks. We also include lexical perturbations, code-identifier renaming, and back-translation as transformation baselines.

5 RESULTS

For each dataset and applicable mutation method, we generate three benchmark variants in separate runs. We evaluate each target model on the original instances and their corresponding mutations to measure performance changes and consistency across runs.

5.1 GENERATED BENCHMARK DIVERSITY

We measure diversity (the variation introduced by transformation, both relative to the original instances and among the generated instances) along three complementary axes: ① *lexical distance* (TF-IDF cosine distance between an original and its transformation); ② *semantic distance* (sentence-embedding, with text-embedding-3-small, cosine between the same pair); and ③ *self-BLEU diversity* across the set of mutated instances produced for each dataset (Montahaei et al., 2019). Figure 2 shows the distribution of these metrics for each system and dataset. We observe the following. BenchAgent's high TF-IDF (0.8–1.0) and embedding (0.4–1.0) distances are an artifact of the absent anchor. This is because it generates benchmarks from scratch, therefore, its pairs are inherently dissimilar regardless of transformation quality, making these scores non-comparable with systems that mutate real instances. DyVal's near-zero scores (TF-IDF 0.00, embedding 0.01, self-BLEU 0.13) reflect its design and should be read as a property of its operator, not a deficiency. Among methods that mutate real instances, BENCHMUTATOR consistently achieves the widest diversity ranges. On HumanEval, at least one BENCHMUTATOR generator outperforms DyCodeEval on both TF-IDF (0.97/0.87 vs. 0.64) and embedding (0.78/0.90 vs. 0.54); on GSM8K, it outperforms Self-Evolving (0.89 vs. 0.67 lexical) and DyVal; on BBC-latest, it exceeds the rule-based transformations (0.94–1.00 vs. 0.68). Critically, self-BLEU scores of 0.86/0.89 on HumanEval and 0.95/0.94 on GSM8K confirm that

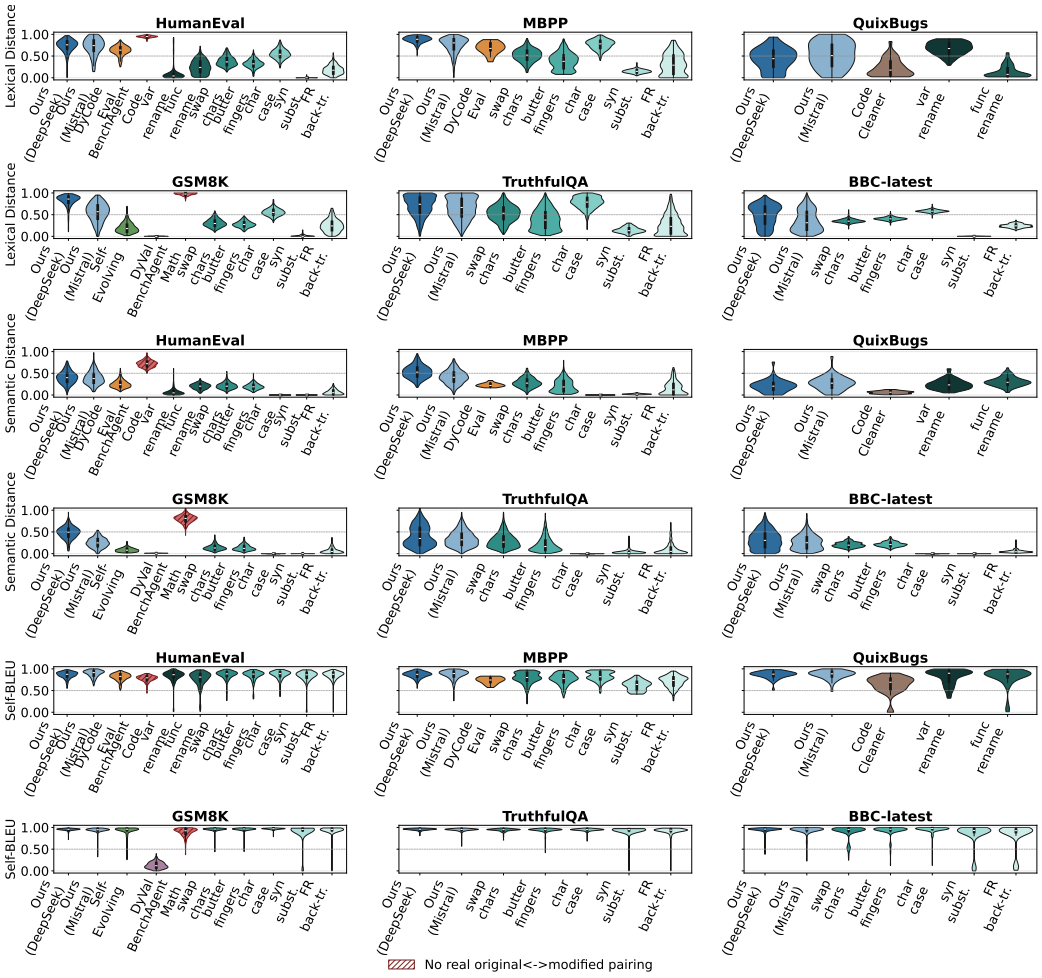

Figure 2: Diversity by dataset and metric, restricted to systems that mutate a real original item for that dataset. BenchAgent Code/Math (hatched, HumanEval and GSM8K respectively) are shown for reference only.

successive BENCHMUTATOR runs produce meaningfully different transformations of the same source instance rather than near-identical outputs. The only tie is TruthfulQA's lexical axis, where rule-based transformations also reach the full 0–1 range. In addition, we observe that the choice of generator is task-dependent. Specifically, on MBPP, DeepSeek's lexical range is 0.36 against Mistral's full 0–1 range, while both peak on TruthfulQA.

Finding (RQ1)

BENCHMUTATOR produces wider lexical and semantic diversity ranges than all competing systems that share a real anchor, and higher self-BLEU set diversity than methods that generate from scratch. BenchAgent's diversity scores are inflated by the absent anchor; DyVal's near-zero scores reflect its graph-based design.

5.2 VALIDATING GENERATED BENCHMARK

We audit transformation quality through a human annotation study on BENCHMUTATOR's own transformations: two independently-recruited annotator pools (A, B) each judged the same 30 items, stratified across the two generators (DeepSeek, Mistral) and five benchmarks, against an 8-point rubric (well-formed, difficulty vs. original, answer-correct, tests-consistent where applicable, no-leak,

accept/reject). Ten annotators contributed 300 labels in total. Results are presented in Tables 2 and 3.

	N	Acceptance	Well-formed	No-leak	Diff. shift
Overall	60	91.7%	91.7%	93.3%	$+0.04$
HumanEval	12	83.4%	83.4%	91.7%	$+0.06$
MBPP	12	100%	100%	100%	$+0.10$
GSM8K	12	100%	100%	91.7%	$+0.04$
TruthfulQA	12	91.7%	91.7%	100%	$+0.06$
BBC-latest	12	83.4%	83.4%	83.4%	-0.08
Pool A	30	93.3%	93.3%	96.7%	-0.07
Pool B	30	90%	90%	90%	$+0.10$

Table 2: Human acceptance of BENCHMUTATOR's mutated items (majority verdict per item), all 10 annotators. Diff. shift is the mean difficulty rating vs. the original on a -2 (much easier) to $+2$ (much harder) scale. Fleiss $\kappa = 0.212$ overall (0.116 pool A, 0.235 pool B).

Generator	HumanEval	MBPP	GSM8K	TruthfulQA	BBC-latest
DeepSeek	100%	100%	100%	100%	83.3%
Mistral	66.7%	100%	100%	83.3%	83.3%

Table 3: Acceptance per generator $\times$ benchmark, $n = 6$ per cell

Overall, 91.7% of transformations were accepted as well-formed and no-leak, with a mean difficulty shift of only $+0.04$ across all items, confirming that the calibration mechanism preserves task difficulty. MBPP and GSM8K achieve perfect acceptance and no-leak rates under both generators; BBC-latest is the weakest benchmark across both criteria (83.4%), likely because paraphrasing open-domain news passages is harder to constrain than structured code or math problems. At the generator level, DeepSeek achieves perfect acceptance on four of five benchmarks, whereas Mistral degrades notably on HumanEval (66.7%) and TruthfulQA (83.3%), consistent with the wider diversity gap observed for DeepSeek on MBPP in Section 5.1. The low inter-annotator agreement (Fleiss $\kappa = 0.212$, fair) is itself informative rather than a study defect. Acceptance and well-formedness are the two most subjective criteria. Annotators disagree primarily on borderline phrasings that are technically valid but stylistically unusual whereas the more objective no-leak criterion reaches $\kappa = 0.31$ in isolation. This pattern suggests that the hard correctness constraints enforced by the Validation Agent are reliably met, while residual disagreement reflects annotator preferences over style rather than substantive transformation failures.

Finding (RQ2)

BENCHMUTATOR transformations are accepted as well-formed and answer-preserving by human annotators at a 91.7% rate overall, with near-zero mean difficulty shift ($+0.04$), confirming that the calibration mechanism preserves task structure.

5.3 EFFECTS OF BENCHMARK TRANSFORMATION ON PERFORMANCE AND CONSISTENCY

We evaluate each target model on every $\langle \text{original}, \text{mutated} \rangle$ pair and report both the mean score ratio and the run-to-run consistency (standard deviation across runs). High variance across runs indicates a key failure mode. It implies that two benchmarks produced by the same strategy may induce very different performance drops, undermining any attribution to contamination. Results are presented in Table 4. We observe that rule-based surface perturbations (character swap, case change, typo injection) yield broadly stable mean scores but with high run-to-run variance (e.g. swap character: $\pm 0.12/\pm 0.21$ on HumanEval). Synonym substitution is the clearest outlier, dropping mutated scores to 0.39 from 0.59 on HumanEval ($\pm 0.15/\pm 0.18$), likely because substitutions alter function semantics rather than surface form. BenchAgent produces near-zero mutated scores (0.16 on HumanEval, 0.23 on GSM8K) with the highest variance (± 0.25, ± 0.08), consistent with the diversity analysis showing it generates instances without reference to the originals. BENCHMUTATOR exhibits meaningful shifts on code tasks with substantially lower variance. On HumanEval, DeepSeek achieves 0.53 ± 0.18 on the original benchmark and 0.66 ± 0.11 on the mutated benchmark and Mistral

	HumanEval		MBPP		QuixBugs		GSM8K		TruthfulQA		BBC-latest	
System	Original	Mutated	Original	Mutated	Original	Mutated	Original	Mutated	Original	Mutated	Original	Mutated
Swap character	0.56 ± 0.12	0.61 ± 0.21	0.39 ± 0.09	0.38 ± 0.09	–	–	0.81 ± 0.11	0.70 ± 0.12	0.43 ± 0.10	0.42 ± 0.08	0.42 ± 0.02	0.44 ± 0.02
Change char case	0.55 ± 0.12	0.59 ± 0.19	0.39 ± 0.09	0.39 ± 0.08	–	–	0.81 ± 0.11	0.73 ± 0.15	0.43 ± 0.11	0.41 ± 0.10	0.42 ± 0.02	0.44 ± 0.02
Typo swap	0.54 ± 0.10	0.60 ± 0.20	0.39 ± 0.09	0.36 ± 0.08	–	–	0.81 ± 0.10	0.73 ± 0.15	0.45 ± 0.10	0.40 ± 0.07	0.42 ± 0.02	0.38 ± 0.02
Synonym substitution	0.59 ± 0.15	0.39 ± 0.18	0.39 ± 0.09	0.39 ± 0.08	–	–	0.78 ± 0.15	0.78 ± 0.13	0.45 ± 0.02	0.42 ± 0.08	0.42 ± 0.02	0.45 ± 0.02
French back translation	0.59 ± 0.13	0.60 ± 0.17	0.40 ± 0.09	0.27 ± 0.05	–	–	0.81 ± 0.11	0.63 ± 0.09	0.47 ± 0.11	0.43 ± 0.07	0.48 ± 0.00	0.49 ± 0.00
Function renaming	0.55 ± 0.13	0.59 ± 0.19	–	–	0.18 ± 0.05	0.12 ± 0.06	–	–	–	–	–	–
Variable renaming	0.58 ± 0.16	0.56 ± 0.20	–	–	0.18 ± 0.06	0.12 ± 0.07	–	–	–	–	–	–
DyCodeEval	0.57 ± 0.12	0.68 ± 0.18	0.62 ± 0.12	0.75 ± 0.00	–	–	–	–	–	–	–	–
CodeCleaner	–	–	–	–	0.15 ± 0.06	0.12 ± 0.06	–	–	–	–	–	–
Self-Evolving	–	–	–	–	–	–	0.78 ± 0.24	0.61 ± 0.21	–	–	–	–
DyVal	–	–	–	–	–	–	0.61 ± 0.15	0.55 ± 0.13	–	–	–	–
BenchAgent - Code	–	0.16 ± 0.25	–	–	–	–	–	–	–	–	–	–
BenchAgent - Math	–	–	–	–	–	–	–	0.23 ± 0.08	–	–	–	–
BENCHMUTATOR (DeepSeek)	0.53 ± 0.18	0.66 ± 0.11	0.49 ± 0.04	0.83 ± 0.07	0.15 ± 0.03	0.12 ± 0.10	0.93 ± 0.07	0.86 ± 0.13	0.45 ± 0.07	0.55 ± 0.09	0.51 ± 0.10	0.71 ± 0.08
BENCHMUTATOR (Mistral)	0.50 ± 0.22	0.41 ± 0.16	0.50 ± 0.06	0.62 ± 0.07	0.15 ± 0.13	0.12 ± 0.10	0.93 ± 0.07	0.85 ± 0.10	0.45 ± 0.08	0.45 ± 0.06	0.49 ± 0.10	0.46 ± 0.06

Table 4: **Performance shift between original and mutated benchmarks.** *Original* denotes evaluation on the unmodified benchmark instances, whereas *Mutated* denotes evaluation on the corresponding transformed instances. Across the 12 BENCHMUTATOR generator–benchmark comparisons, transformation increases the mean score in 6 cases, leaves it unchanged in 2. A "–" indicates that the corresponding transformation method is not applicable to that benchmark or task; for example, function renaming is applicable to HumanEval and QuixBugs, but not to benchmarks that do not expose a function-level program representation. BenchAgent has no directly paired original benchmark and is therefore reported only for the generated benchmark. We report here the averages across the four model families. Per model result is in Table 7 in the Appendix.

0.50 ± 0.22 and 0.41 ± 0.16 respectively. On GSM8K, both generators preserve original performance more closely, reflecting the affine-transform constraint on math transformations. BENCHMUTATOR's consistency (± 0.07–± 0.13) compares favorably with that of Self-Evolving (± 0.24–± 0.21) and DyVal (± 0.15–± 0.13) on GSM8K. A recurring pattern across all methods is that newer and larger models absorb transformations with smaller drops, suggesting that transformation sensitivity reflects model capability as much as benchmark exposure.

Finding (RQ3)

BENCHMUTATOR produces more consistent performance shifts than Self-Evolving and DyVal on GSM8K, with stronger shifts on code tasks than math tasks. Our results shows that transformation sensitivity reflects model capability as much as benchmark exposure.

5.4 CONTAMINATION RESISTANCE

To measure the contamination effect, we follow the controlled pipeline proposed by Sun et al. (2025). We first fine-tune the base model on the original benchmark instances to force contamination, and then we evaluate the clean base model and contaminated model on the mutated benchmark. Contamination resistance (CR) is the normalized Hamming distance between the question-level evaluation vectors of the clean and contaminated models on the mutated benchmark. A high CR score indicates that question-level answers remain stable after contamination, meaning the transformation successfully neutralizes the contamination advantage. In this study, Qwen, Llama, and Gemma are fine-tuned with full-weight fine-tuning at a learning rate of 2×10^{-5} for 3 epochs. Due to the large size of Deepseek, we did not fine-tune it. Results are presented in Table 5. BENCHMUTATOR (DeepSeek) achieves CR scores of 0.656–0.986 on HumanEval and 0.741–0.884 on GSM8K across all three model families, consistently above rule-based baselines, and generalizes to text tasks (TruthfulQA: 0.784–0.884; BBC-latest: 0.772–0.791). Qwen scores are anomalously high (0.986 on HumanEval and MBPP), suggesting a stronger tendency towards verbatim reproduction. Character-level baselines achieve moderate CR (0.613–0.845 on HumanEval). We also observe that CR varies substantially across model families under identical transformations. Specifically, newer and larger models remain robust regardless of fine-tuning, while older and smaller models show higher question-level divergence under any perturbation.

	HumanEval			MBPP			QuixBugs			GSM8K			TruthfulQA			BBC-latest		
System	Llama	Gemma	Qwen	Llama	Gemma	Qwen	Llama	Gemma	Qwen	Llama	Gemma	Qwen	Llama	Gemma	Qwen	Llama	Gemma	Qwen
Swap character	0.758	0.684	0.717	0.765	0.726	0.798	–	–	–	0.693	0.661	0.640	0.634	0.516	0.699	0.694	0.740	0.731
Change char case	0.708	0.682	0.684	0.801	0.713	0.801	–	–	–	0.640	0.649	0.629	0.622	0.501	0.675	0.677	0.786	0.733
Typo swap	0.845	0.671	0.357	0.797	0.752	0.841	–	–	–	0.675	0.652	0.620	0.668	0.568	0.765	0.752	0.755	0.750
Synonym substitution	0.750	0.679	0.772	1.000	1.000	1.000	–	–	–	0.625	0.615	0.615	0.607	0.464	0.768	0.718	0.782	0.735
French back translation	0.743	0.682	0.763	0.803	0.777	0.815	–	–	–	0.667	0.719	0.608	0.731	0.632	0.745	0.692	0.755	0.776
Function renaming	0.645	0.710	0.690	–	–	–	0.788	0.764	0.743	–	–	–	–	–	–	–	–	
Variable renaming	0.676	0.754	0.684	–	–	–	0.694	0.722	0.738	–	–	–	–	–	–	–	–	
DyCodeEval	0.613	0.658	0.658	0.732	0.695	0.878	–	–	–	–	–	–	–	–	–	–	–	–
CodeCleaner	–	–	–	–	–	–	0.671	0.714	0.600	–	–	–	–	–	–	–	–	–
Self-Evolving	–	–	–	–	–	–	–	–	–	0.744	0.767	0.778	–	–	–	–	–	–
DyVal	–	–	–	–	–	–	–	–	–	0.664	0.665	0.802	–	–	–	–	–	–
BENCHMUTATOR (DeepSeek)	0.656	0.663	0.986	0.759	0.671	0.986	0.819	0.841	0.648	0.741	0.683	0.772	0.784	0.820	0.884	0.772	0.785	0.791
BENCHMUTATOR (Mistral)	0.794	0.828	0.824	0.794	0.722	0.773	0.672	0.797	0.750	0.716	0.734	0.688	0.755	0.799	0.773	0.782	0.774	0.765

Table 5: **Contamination Resistance per transformation method**. Higher values indicates that the transformation effectively neutralizes the contamination advantage of the original benchmark. BENCHMUTATOR (DeepSeek) consistently achieves the highest contamination resistance across benchmarks and model families, confirming that resistance reflects model capability as much as contamination exposure.

Finding (RQ4)

BENCHMUTATOR achieves competitive contamination resistance across three model families, consistently outperforming rule-based baselines. Newer, larger models show lower question-level divergence regardless of contamination status, indicating that resistance cannot be used as a universal threshold.

5.5 TRANSFORMATION COST

In this section, we make a comparative cost evaluation across the different transformation approaches against MBPP while considering the generation of 500 code synthesis samples with BenchAgent. For this, we used a single NVIDIA RTX A6000 (48 GB) rented at \$0.33/hr [1], with Mistral Small 24B served via vLLM. BENCHMUTATOR, DyCodeEval and BenchAgent used this model while French back-translation uses m2m100 (a dedicated translation model).

Method	token or s/item	\$/1k items
DyCodeEval	108.5 tok	\$0.08
BenchAgent	505.8 tok	\$0.36
French back translation	15.4 s	\$0.705
BENCHMUTATOR **(Ours)**	503.2 tok	\$0.36

Table 6: GPU cost estimation of the different mutated benchmarks on an NVIDIA RTX A6000 (\$0.33/hr)

We observe that BENCHMUTATOR and BenchAgent are cost-equivalent per item, despite their fundamentally different generation strategies. Second, DyCodeEval is $4.6\times$ cheaper than BENCHMUTATOR, however, as shown in Table 1, it can only be applied to the conde synthesis task. The French back translation is the most expensive option because of the dedicated translation model m2m100's (Dhole et al., 2023). The other rule-based transformation strategies don't involve the usage of GPU.

6 CONCLUSION

We introduced BENCHMUTATOR, a multi-agent framework that infers task-aware transformation rules from a representative benchmark sample and applies them without a predefined taxonomy. Across six benchmarks and four model families, BENCHMUTATOR produces wider diversity than all competing methods that mutate real instances, achieves a 91.7% human-judged acceptance rate with near-zero difficulty shift, maintains lower run-to-run variance, and achieves competitive contamination resistance. CR nevertheless varies across model families under identical mutations. Newer and larger models remain robust regardless of contamination status, so transformation sensitivity reflects model capability as much as benchmark exposure.

[1] https://www.runpod.io/pricing

AI USE STATEMENT

We used generative AI tools to assist with spelling, and grammar, as well as bug localization and correction during the development of BENCHMUTATOR. AI-based generation of mutated benchmarks is a core component of our research methodology and is described in the paper. A subset of the generated instances was also evaluated by human annotators, as reported in Section 5.2.

REPRODUCIBILITY STATEMENT

The data used in this research are derived from publicly available sources. To ensure transparency and reproducibility, the source code, datasets, and usage instructions are publicly accessible in our anonymized repository at https://anonymous.4open.science/r/BenchMutator-2AFE/.

REFERENCES

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.

Natasha Butt, Varun Chandrasekaran, Neel Joshi, Besmira Nushi, and Vidhisha Balachandran. BENCHAGENTS: Automated Benchmark Creation with Agent Interaction, October 2024. URL http://arxiv.org/abs/2410.22584. arXiv:2410.22584 [cs].

Jialun Cao, Songqiang Chen, Wuqi Zhang, Hau Ching Lo, and Shing-Chi Cheung. CODECLEANER: Elevating Standards with A Robust Data Contamination Mitigation Toolkit, November 2024. URL http://arxiv.org/abs/2411.10842. arXiv:2411.10842 [cs].

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying Memorization Across Neural Language Models, 2023.

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.

Simin Chen, Pranav Pusarla, and Baishakhi Ray. Dynamic Benchmarking of Reasoning Capabilities in Code Large Language Models Under Data Contamination, June 2025. URL http://arxiv.org/abs/2503.04149. arXiv:2503.04149 [cs].

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

William Gemmell Cochran and William G Cochran. *Sampling techniques*, volume 3. wiley New York, 1977.

Chunyuan Deng, Yilun Zhao, Xiangru Tang, Mark Gerstein, and Arman Cohan. Investigating data contamination in modern benchmarks for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8698–8711, 2024.

Kaustubh Dhole, Varun Gangal, Sebastian Gehrmann, Aadesh Gupta, Zhenhao Li, Saad Mahamood, Abinaya Mahadiran, Simon Mille, Ashish Shrivastava, Samson Tan, Tongshang Wu, Jascha Sohl-Dickstein, Jinho Choi, Eduard Hovy, Ondřej Dušek, Sebastian Ruder, Sajant Anand, Nagender Aneja, Rabin Banjade, Lisa Barthe, Hanna Behnke, Ian Berlot-Attwell, Connor Boyle, Caroline Brun, Marco Antonio Sobrevilla Cabezudo, Samuel Cahyawijaya, Emile Chapuis, Wanxiang Che, Mukund Choudhary, Christian Clauss, Pierre Colombo, Filip Cornell, Gautier Dagan, Mayukh Das, Tanay Dixit, Thomas Dopierre, Paul-Alexis Dray, Suchitra Dubey, Tatiana Ekeinhor, Marco Di Giovanni, Tanya Goyal, Rishabh Gupta, Louanes Hamla, Sang Han, Fabrice Harel-Canada, Antoine Honoré, Ishan Jindal, Przemysław Joniak, Denis Kleyko, Venelin Kovatchev, Kalpesh Krishna,

Ashutosh Kumar, Stefan Langer, Seungjae Ryan Lee, Corey James Levinson, Hualou Liang, Kaizhao Liang, Zhexiong Liu, Andrey Lukyanenko, Vukosi Marivate, Gerard de Melo, Simon Meoni, Maxine Meyer, Afnan Mir, Nafise Sadat Moosavi, Niklas Meunnighoff, Timothy Sum Hon Mun, Kenton Murray, Marcin Namysl, Maria Obedkova, Priti Oli, Nivranshu Pasricha, Jan Pfister, Richard Plant, Vinay Prabhu, Vasile Pais, Libo Qin, Shahab Raji, Pawan Kumar Rajpoot, Vikas Raunak, Roy Rinberg, Nicholas Roberts, Juan Diego Rodriguez, Claude Roux, Vasconcellos Samus, Ananya Sai, Robin Schmidt, Thomas Scialom, Tshephisho Sefara, Saqib Shamsi, Xudong Shen, Yiwen Shi, Haoyue Shi, Anna Shvets, Nick Siegel, Damien Sileo, Jamie Simon, Chandan Singh, Roman Sitelew, Priyank Soni, Taylor Sorensen, William Soto, Aman Srivastava, Aditya Srivatsa, Tony Sun, Mukund Varma, A Tabassum, Fiona Tan, Ryan Teehan, Mo Tiwari, Marie Tolkiehn, Athena Wang, Zijian Wang, Zijie Wang, Gloria Wang, Fuxuan Wei, Bryan Wilie, Genta Indra Winata, Xinyu Wu, Witold Wydmanski, Tianbao Xie, Usama Yaseen, Michael Yee, Jing Zhang, and Yue Zhang. NL-Augmenter: A Framework for Task-Sensitive Natural Language Augmentation. *Northern European Journal of Language Technology*, 9, 2023. doi: 10.3384/nejlt.2000-1533.2023.4725. URL https://aclanthology.org/2023.nejlt-1.5/. Place: Linköping, Sweden Publisher: Linköping University Electronic Press.

Albérick Euraste Djiré, Abdoul Kader Kaboré, Earl T Barr, Jacques Klein, and Tegawendé F Bissyandé. Memorization or interpolation? detecting llm memorization through input perturbation analysis. *arXiv preprint arXiv:2505.03019*, 2025.

Michael Duan, Anshuman Suri, Niloofar Mireshghallah, Sewon Min, Weijia Shi, Luke Zettlemoyer, Yulia Tsvetkov, Yejin Choi, David Evans, and Hannaneh Hajishirzi. Do membership inference attacks work on large language models? *arXiv preprint arXiv:2402.07841*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

Shahriar Golchin and Mihai Surdeanu. Time Travel in LLMs: Tracing Data Contamination in Large Language Models, February 2024. URL http://arxiv.org/abs/2308.08493. arXiv:2308.08493 [cs].

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code, 2024. URL https://arxiv.org/abs/2403.07974.

Yucheng Li, Frank Guerin, and Chenghua Lin. LatestEval: Addressing Data Contamination in Language Model Evaluation through Dynamic and Time-Sensitive Test Construction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):18600–18607, March 2024. ISSN 2374-3468. doi: 10.1609/aaai.v38i17.29822. URL https://ojs.aaai.org/index.php/AAAI/article/view/29822. Number: 17.

Derrick Lin, James Koppel, Angela Chen, and Armando Solar-Lezama. Quixbugs: a multi-lingual program repair benchmark set based on the quixey challenge. In *Proceedings Companion of the 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, SPLASH Companion 2017, pp. 55–56, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450355148. doi: 10.1145/3135932.3135941. URL https://doi.org/10.1145/3135932.3135941.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods, 2021.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Inbal Magar and Roy Schwartz. Data contamination: From memorization to exploitation. *arXiv preprint arXiv:2203.08242*, 2022.

Justus Mattern, Fatemehsadat Mireshghallah, Zhijing Jin, Bernhard Schölkopf, Mrinmaya Sachan, and Taylor Berg-Kirkpatrick. Membership Inference Attacks against Language Models via Neighbourhood Comparison, August 2023. URL http://arxiv.org/abs/2305.18462. arXiv:2305.18462 [cs].

Ehsan Montahaei, Danial Alihosseini, and Mahdieh Soleymani Baghshah. Jointly measuring diversity and quality in text generation models, 2019. In *NAACL 2019 workshop (NeuralGen 2019)*, volume 5, 2019.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Hongzhou Rao, Yanjie Zhao, Wenjie Zhu, Ling Xiao, Meizhen Wang, and Haoyu Wang. CodeMorph: Mitigating Data Leakage in Large Language Model Assessment, June 2025. URL http://arxiv.org/abs/2506.17627. arXiv:2506.17627 [cs].

Abhinav Rastogi, Albert Q Jiang, Andy Lo, Gabrielle Berrada, Guillaume Lample, Jason Rute, Joep Barmentlo, Karmesh Yadav, Kartik Khandelwal, Khyathi Raghavi Chandu, et al. Magistral. *arXiv preprint arXiv:2506.10910*, 2025.

Manley Roberts, Himanshu Thakur, Christine Herlihy, Colin White, and Samuel Dooley. To the cutoff... and beyond? a longitudinal perspective on llm data contamination. In *The Twelfth International Conference on Learning Representations*, 2023.

Oscar Sainz, Jon Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. Nlp evaluation in trouble: On the need to measure llm data contamination for each benchmark. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10776–10787, 2023.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pp. 3–18. IEEE, 2017. URL https://ieeexplore.ieee.org/abstract/document/7958568/.

Yifan Sun, Han Wang, Dongbai Li, Gang Wang, and Huan Zhang. The emperor's new clothes in benchmarking? a rigorous examination of mitigation strategies for llm benchmark data contamination. In *Forty-second International Conference on Machine Learning*, 2025.

Gemma Team, Morgane Riviere, Shreya Pathak, et al. Gemma 2: Improving Open Language Models at a Practical Size, October 2024. URL http://arxiv.org/abs/2408.00118. arXiv:2408.00118.

Shiqi Wang, Zheng Li, Haifeng Qian, Chenghao Yang, Zijian Wang, Mingyue Shang, Varun Kumar, Samson Tan, Baishakhi Ray, Parminder Bhatia, Ramesh Nallapati, Murali Krishna Ramanathan, Dan Roth, and Bing Xiang. ReCode: Robustness Evaluation of Code Generation Models, December 2022. URL http://arxiv.org/abs/2212.10264. arXiv:2212.10264 [cs].

Siyuan Wang, Zhuohan Long, Zhihao Fan, Zhongyu Wei, and Xuanjing Huang. Benchmark Self-Evolving: A Multi-Agent Framework for Dynamic LLM Evaluation, February 2024. URL http://arxiv.org/abs/2402.11443. arXiv:2402.11443 [cs].

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, et al. Livebench: A challenging, contamination-limited llm benchmark. In *The Thirteenth International Conference on Learning Representations*.

Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st computer security foundations symposium (CSF)*, pp. 268–282. IEEE, 2018.

Kaijie Zhu, Jiaao Chen, Jindong Wang, Neil Zhenqiang Gong, Diyi Yang, and Xing Xie. DyVal: Dynamic Evaluation of Large Language Models for Reasoning Tasks, March 2024a. URL http://arxiv.org/abs/2309.17167. arXiv:2309.17167 [cs].

Qin Zhu, Qinyuan Cheng, Runyu Peng, Xiaonan Li, Ru Peng, Tengxiao Liu, Xipeng Qiu, and Xuanjing Huang. Inference-Time Decontamination: Reusing Leaked Benchmarks for Large Language Model Evaluation. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 9113–9129, Miami, Florida, USA, November 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.532. URL https://aclanthology.org/2024.findings-emnlp.532/.

Wenhong Zhu, Hongkun Hao, Zhiwei He, Yun-Ze Song, Jiao Yueyang, Yumeng Zhang, Hanxu Hu, Yiran Wei, Rui Wang, and Hongyuan Lu. Clean–eval: Clean evaluation on contaminated large language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 835–847, 2024c.

A APPENDIX

A.1 PERFORMANCE SHIFT PER MODEL

Table 7 details the results of Section 5.3 by reporting score and consistency for each model family (Gemma, Llama, Qwen, DeepSeek) separately. Three patterns emerge from the per-family breakdown.

	HumanEval		MBPP		QuixBugs		GSM8K		TruthfulQA		BBC-latest	
Method	Original	Mutated	Original	Mutated	Original	Mutated	Original	Mutated	Original	Mutated	Original	Mutated
Gemma												
Typo swap	0.65 ± 0.11	0.79 ± 0.12	0.51 ± 0.11	0.49 ± 0.10	–	–	0.67 ± 0.10	0.56 ± 0.13	0.44 ± 0.10	0.41 ± 0.09	0.44 ± 0.05	0.43 ± 0.10
Change char case	0.63 ± 0.12	0.83 ± 0.15	0.52 ± 0.09	0.54 ± 0.10	–	–	0.67 ± 0.08	0.59 ± 0.14	0.44 ± 0.10	0.45 ± 0.11	0.44 ± 0.05	0.43 ± 0.10
French back translation	0.63 ± 0.12	0.83 ± 0.15	0.54 ± 0.13	0.48 ± 0.12	–	–	0.66 ± 0.12	0.50 ± 0.14	0.44 ± 0.10	0.41 ± 0.08	0.45 ± 0.08	0.39 ± 0.08
Swap character	0.65 ± 0.11	0.78 ± 0.13	0.49 ± 0.15	0.59 ± 0.10	–	–	0.67 ± 0.08	0.54 ± 0.16	0.46 ± 0.10	0.43 ± 0.10	0.44 ± 0.10	0.44 ± 0.15
Synonym substitution	0.63 ± 0.12	0.36 ± 0.18	0.52 ± 0.11	0.49 ± 0.12	–	–	0.58 ± 0.12	0.60 ± 0.17	0.44 ± 0.10	0.44 ± 0.13	0.44 ± 0.10	0.43 ± 0.15
Function renaming	0.63 ± 0.11	0.40 ± 0.13	–	–	0.12 ± 0.00	0.12 ± 0.00	–	–	–	–	–	–
Variable renaming	0.62 ± 0.12	0.37 ± 0.13	–	–	0.14 ± 0.00	0.09 ± 0.03	–	–	–	–	–	–
DyCodeEval	0.72 ± 0.12	0.72 ± 0.22	0.60 ± 0.11	0.54 ± 0.15	–	–	–	–	–	–	–	–
CodeCleaner	–	–	–	–	0.09 ± 0.00	0.03 ± 0.00	–	–	–	–	–	–
Self-Evolving	–	–	–	–	–	–	0.85 ± 0.11	0.51 ± 0.32	–	–	–	–
DyVal	–	–	–	–	–	–	0.62 ± 0.12	0.61 ± 0.11	–	–	–	–
BenchAgent Code	–	0.18 ± 0.24	–	–	–	–	–	–	–	–	–	–
BenchAgent Math	–	–	–	–	–	–	–	0.18 ± 0.10	–	–	–	–
BENCHMUTATOR (DeepSeek)	0.69 ± 0.18	0.72 ± 0.14	0.58 ± 0.01	0.82 ± 0.12	0.12 ± 0.00	0.12 ± 0.00	0.94 ± 0.11	0.91 ± 0.12	0.46 ± 0.11	0.51 ± 0.13	0.45 ± 0.03	0.63 ± 0.17
BENCHMUTATOR (Mistral)	0.68 ± 0.22	0.48 ± 0.24	0.60 ± 0.04	0.63 ± 0.15	0.14 ± 0.00	0.09 ± 0.03	0.96 ± 0.11	0.93 ± 0.11	0.46 ± 0.11	0.43 ± 0.12	0.44 ± 0.04	0.43 ± 0.11
Llama												
Typo swap	0.43 ± 0.13	0.56 ± 0.15	0.33 ± 0.09	0.26 ± 0.05	–	–	0.81 ± 0.08	0.67 ± 0.10	0.40 ± 0.10	0.33 ± 0.10	0.45 ± 0.10	0.35 ± 0.08
Change char case	0.43 ± 0.11	0.56 ± 0.13	0.31 ± 0.12	0.31 ± 0.06	–	–	0.80 ± 0.07	0.70 ± 0.13	0.38 ± 0.10	0.43 ± 0.08	0.45 ± 0.10	0.35 ± 0.12
French back translation	0.43 ± 0.12	0.31 ± 0.19	0.32 ± 0.10	0.21 ± 0.06	–	–	0.80 ± 0.11	0.60 ± 0.11	0.38 ± 0.10	0.35 ± 0.10	0.43 ± 0.11	0.39 ± 0.11
Swap character	0.43 ± 0.11	0.59 ± 0.15	0.31 ± 0.08	0.29 ± 0.09	–	–	0.79 ± 0.08	0.67 ± 0.11	0.40 ± 0.10	0.39 ± 0.10	0.45 ± 0.10	0.37 ± 0.10
Synonym substitution	0.43 ± 0.12	0.33 ± 0.19	0.32 ± 0.11	0.25 ± 0.07	–	–	0.71 ± 0.10	0.71 ± 0.10	0.38 ± 0.10	0.38 ± 0.10	0.44 ± 0.10	0.36 ± 0.12
Function renaming	0.43 ± 0.12	0.44 ± 0.13	–	–	0.29 ± 0.15	0.20 ± 0.07	–	–	–	–	–	–
Variable renaming	0.42 ± 0.12	0.44 ± 0.13	–	–	0.29 ± 0.15	0.15 ± 0.07	–	–	–	–	–	–
DyCodeEval	0.41 ± 0.13	0.48 ± 0.14	0.62 ± 0.00	0.83 ± 0.06	–	–	–	–	–	–	–	–
CodeCleaner	–	–	–	–	0.18 ± 0.00	0.20 ± 0.00	–	–	–	–	–	–
Self-Evolving	–	–	–	–	–	–	0.72 ± 0.42	0.77 ± 0.23	–	–	–	–
DyVal	–	–	–	–	–	–	0.58 ± 0.02	0.56 ± 0.12	–	–	–	–
BenchAgent Code	–	0.16 ± 0.22	–	–	–	–	–	–	–	–	–	–
BenchAgent Math	–	–	–	–	–	–	–	0.18 ± 0.08	–	–	–	–
BENCHMUTATOR (DeepSeek)	0.33 ± 0.22	0.56 ± 0.06	0.43 ± 0.04	0.76 ± 0.04	0.20 ± 0.02	0.25 ± 0.07	0.94 ± 0.10	0.84 ± 0.13	0.40 ± 0.11	0.53 ± 0.14	0.40 ± 0.10	0.65 ± 0.17
BENCHMUTATOR (Mistral)	0.27 ± 0.18	0.28 ± 0.12	0.40 ± 0.09	0.55 ± 0.06	0.29 ± 0.07	0.19 ± 0.19	0.95 ± 0.11	0.87 ± 0.13	0.38 ± 0.11	0.44 ± 0.12	0.39 ± 0.12	0.40 ± 0.13
Qwen												
Typo swap	0.71 ± 0.12	0.54 ± 0.10	0.48 ± 0.07	0.40 ± 0.05	–	–	0.82 ± 0.06	0.84 ± 0.10	0.42 ± 0.10	0.44 ± 0.10	0.45 ± 0.03	0.62 ± 0.08
Change char case	0.71 ± 0.12	0.56 ± 0.10	0.44 ± 0.07	0.55 ± 0.04	–	–	0.88 ± 0.15	0.82 ± 0.10	0.44 ± 0.10	0.44 ± 0.10	0.46 ± 0.08	0.65 ± 0.10
French back translation	0.69 ± 0.12	0.43 ± 0.10	0.48 ± 0.08	0.28 ± 0.03	–	–	0.90 ± 0.08	0.84 ± 0.10	0.38 ± 0.10	0.48 ± 0.14	0.56 ± 0.10	0.41 ± 0.14
Swap character	0.71 ± 0.12	0.56 ± 0.10	0.48 ± 0.09	0.45 ± 0.11	–	–	0.85 ± 0.12	0.86 ± 0.11	0.36 ± 0.10	0.43 ± 0.11	0.45 ± 0.10	0.39 ± 0.10
Synonym substitution	0.69 ± 0.14	0.48 ± 0.10	0.47 ± 0.07	0.58 ± 0.06	–	–	0.88 ± 0.07	0.88 ± 0.11	0.40 ± 0.10	0.47 ± 0.14	0.45 ± 0.10	0.45 ± 0.10
Function renaming	0.71 ± 0.13	0.70 ± 0.10	–	–	0.12 ± 0.09	0.09 ± 0.03	–	–	–	–	–	–
Variable renaming	0.70 ± 0.13	0.70 ± 0.10	–	–	0.09 ± 0.09	0.09 ± 0.00	–	–	–	–	–	–
DyCodeEval	0.74 ± 0.14	0.78 ± 0.14	0.90 ± 0.12	0.92 ± 0.05	–	–	–	–	–	–	–	–
CodeCleaner	–	–	–	–	0.09 ± 0.00	0.06 ± 0.00	–	–	–	–	–	–
Self-Evolving	–	–	–	–	–	–	0.66 ± 0.44	0.40 ± 0.41	–	–	–	–
DyVal	–	–	–	–	–	–	0.67 ± 0.01	0.05 ± 0.12	–	–	–	–
BenchAgent Code	–	0.17 ± 0.28	–	–	–	–	–	–	–	–	–	–
BenchAgent Math	–	–	–	–	–	–	–	0.32 ± 0.12	–	–	–	–
BENCHMUTATOR (DeepSeek)	0.74 ± 0.14	0.76 ± 0.15	0.51 ± 0.06	0.78 ± 0.06	0.08 ± 0.04	0.03 ± 0.05	0.91 ± 0.08	0.82 ± 0.12	0.39 ± 0.11	0.48 ± 0.14	0.46 ± 0.11	0.68 ± 0.14
BENCHMUTATOR (Mistral)	0.74 ± 0.18	0.50 ± 0.12	0.53 ± 0.06	0.62 ± 0.05	0.03 ± 0.03	0.09 ± 0.02	0.92 ± 0.07	0.78 ± 0.11	0.38 ± 0.10	0.40 ± 0.13	0.47 ± 0.12	0.45 ± 0.13
DeepSeek												
Typo swap	0.37 ± 0.04	0.51 ± 0.43	0.24 ± 0.09	0.29 ± 0.12	–	–	0.94 ± 0.12	0.85 ± 0.23	0.54 ± 0.10	0.42 ± 0.01	0.34 ± 0.10	0.33 ± 0.08
Change char case	0.43 ± 0.13	0.41 ± 0.38	0.29 ± 0.08	0.16 ± 0.12	–	–	0.89 ± 0.09	0.81 ± 0.26	0.46 ± 0.14	0.32 ± 0.11	0.33 ± 0.15	0.55 ± 0.17
French back translation	0.61 ± 0.52	0.45 ± 0.17	0.26 ± 0.05	0.11 ± 0.01	–	–	0.88 ± 0.11	0.58 ± 0.11	0.68 ± 0.14	0.48 ± 0.04	0.48 ± 0.24	0.77 ± 0.24
Swap character	0.45 ± 0.14	0.51 ± 0.46	0.28 ± 0.04	0.19 ± 0.06	–	–	0.93 ± 0.09	0.73 ± 0.09	0.50 ± 0.10	0.43 ± 0.08	0.34 ± 0.22	0.56 ± 0.16
Synonym substitution	0.61 ± 0.22	0.39 ± 0.25	0.25 ± 0.07	0.24 ± 0.07	–	–	0.95 ± 0.11	0.93 ± 0.14	0.58 ± 0.22	0.39 ± 0.05	0.35 ± 0.22	0.56 ± 0.22
Function renaming	0.43 ± 0.16	0.82 ± 0.40	–	–	0.19 ± 0.12	0.12 ± 0.16	–	–	–	–	–	–
Variable renaming	0.58 ± 0.27	0.73 ± 0.44	–	–	0.20 ± 0.22	0.15 ± 0.10	–	–	–	–	–	–
DyCodeEval	0.41 ± 0.09	0.74 ± 0.22	0.36 ± 0.12	0.71 ± 0.26	–	–	–	–	–	–	–	–
CodeCleaner	–	–	–	–	0.24 ± 0.19	0.19 ± 0.24	–	–	–	–	–	–
Self-Evolving	–	–	–	–	–	–	0.89 ± 0.01	0.76 ± 0.12	–	–	–	–
DyVal	–	–	–	–	–	–	0.57 ± 0.45	0.98 ± 0.17	–	–	–	–
BenchAgent Code	–	0.13 ± 0.26	–	–	–	–	–	–	–	–	–	–
BenchAgent Math	–	–	–	–	–	–	–	0.24 ± 0.02	–	–	–	–
BENCHMUTATOR (DeepSeek)	0.36 ± 0.18	0.60 ± 0.09	0.44 ± 0.05	0.96 ± 0.06	0.20 ± 0.08	0.12 ± 0.06	0.93 ± 0.12	0.87 ± 0.15	0.55 ± 0.05	0.68 ± 0.05	0.73 ± 0.16	0.88 ± 0.16
BENCHMUTATOR (Mistral)	0.31 ± 0.30	0.38 ± 0.16	0.47 ± 0.05	0.68 ± 0.02	0.14 ± 0.42	0.11 ± 0.16	0.89 ± 0.13	0.82 ± 0.05	0.58 ± 0.00	0.53 ± 0.13	0.66 ± 0.12	0.56 ± 0.13

Table 7: **Performance shift between original and mutated benchmarks, by target model family.** *Original* denotes evaluation on the unmodified benchmark instances, whereas *Mutated* denotes evaluation on the corresponding transformed instances generated by each mutation method. Each entry reports the mean score $\pm$ standard deviation across independent benchmark-generation runs. The four blocks correspond to the target models Gemma, Llama, Qwen, and DeepSeek; these models are evaluated on both the original and mutated benchmarks and are distinct from the generator model used by BENCHMUTATOR. For BENCHMUTATOR, DeepSeek and Mistral denote the underlying generator model used within the mutation pipeline. A "–" denotes a mutation–benchmark combination for which the mutation method is not applicable to that task. BenchAgent does not provide a directly paired original benchmark and is therefore reported only in the *Mutated* column.

Model capability modulates mutation sensitivity. The performance drop under mutation varies substantially across model families for the same strategy. On HumanEval, synonym substitution drops Gemma from 0.63 to 0.36 but leaves Qwen nearly stable ($0.69/0.48$) and causes only a moderate drop for Llama ($0.43/0.33$). On GSM8K, Self-Evolving drops Gemma to 0.51 but *increases* Llama's

score (0.62/0.77), a pattern inconsistent with any contamination-based explanation and indicative of capability differences in handling rephrased problems. This confirms the finding of Section 5.3 that transformation sensitivity is a function of model capability as much as benchmark exposure.

BENCHMUTATOR is consistent across model families. BENCHMUTATOR (DeepSeek) produces positive or near-neutral score shifts on code tasks for all four families, with lower run-to-run variance than rule-based baselines in most cells. The most notable exception is Llama on HumanEval, where the original score (0.33) is already low, limiting the room for a meaningful shift. On GSM8K, all four families show consistent preservation of original performance (0.91–0.96 original, 0.82–0.91 mutated), confirming that the affine-transform constraint successfully bounds difficulty change.

DeepSeek as a target model. The DeepSeek family block shows higher variance than other families on several rule-based methods (e.g. swap character on HumanEval: ± 0.46 mutated; French back-translation on HumanEval: ± 0.68 mutated), and anomalous scores above 1.0 on BBC-latest for multiple baselines. These results should be interpreted cautiously given the dual role of DeepSeek as both generator agent and target model family, and the BBC-latest anomalies likely reflect a scoring artifact rather than genuine performance.

A.2 PERFORMANCE VARIATION DURING FINE-TUNING

As mentionned in the section 5.4, we fine-tune each target model on the original benchmarks and track performance across training. Table 8 reports scores for BENCHMUTATOR, the rule-based and agentic baselines, both before fine-tuning (*Base Orig*, *Base Mut*) and after three epochs of fine-tuning on the original benchmark (*FT Orig*, *FT Mut*).

	HumanEval				MBPP				QuixBugs				GSM8K				TruthfulQA				BBC-latest			
System	Base Orig	Base Mut	FT Orig	FT Mut	Base Orig	Base Mut	FT Orig	FT Mut	Base Orig	Base Mut	FT Orig	FT Mut	Base Orig	Base Mut	FT Orig	FT Mut	Base Orig	Base Mut	FT Orig	FT Mut	Base Orig	Base Mut	FT Orig	FT Mut
Gemma																								
Typo swap	0.65	0.79	0.693	0.521	0.51	0.49	0.590	0.461	–	–	–	–	0.67	0.67	0.721	0.639	0.41	0.41	0.841	0.552	0.44	0.43	0.427	0.417
Change char case	0.63	0.83	0.693	0.562	0.52	0.54	0.590	0.553	–	–	–	–	0.67	0.67	0.721	0.651	0.45	0.45	0.840	0.685	0.44	0.43	0.422	0.410
French back translation	0.63	0.83	0.500	0.500	0.54	0.48	0.580	0.494	–	–	–	–	0.66	0.66	0.715	0.663	0.41	0.41	0.843	0.547	0.45	0.39	0.422	0.412
Swap character	0.65	0.78	0.693	0.617	0.49	0.59	0.585	0.574	–	–	–	–	0.67	0.67	0.715	0.663	0.43	0.43	0.840	0.633	0.44	0.44	0.422	0.420
Synonym substitution	0.63	0.36	0.693	0.500	0.52	0.49	0.300	0.300	–	–	–	–	0.58	0.58	0.633	0.717	0.44	0.44	0.750	0.714	0.44	0.43	0.425	0.420
Function renaming	0.63	0.40	0.700	0.652	–	–	–	–	0.12	0.12	0.500	0.371	–	–	–	–	–	–	–	–	–	–	–	–
Variable renaming	0.62	0.37	0.677	0.618	–	–	–	–	0.14	0.09	0.404	0.371	–	–	–	–	–	–	–	–	–	–	–	–
DyCodeEval	0.72	0.72	0.659	0.704	0.6	0.54	0.300	0.300	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
CodeCleaner	–	–	–	–	–	–	–	–	0.09	0.03	0.457	0.229	–	–	–	–	–	–	–	–	–	–	–	–
Self-Evolving	–	–	–	–	–	–	–	–	–	–	–	–	0.85	0.51	0.838	0.509	–	–	–	–	–	–	–	–
DyVal	–	–	–	–	–	–	–	–	–	–	–	–	0.62	0.61	0.508	0.446	–	–	–	–	–	–	–	–
BENCHMUTATOR (DeepSeek)	0.69	0.72	0.728	0.634	0.58	0.82	0.616	0.687	0.12	0.12	0.383	0.276	0.94	0.91	0.931	0.887	0.46	0.51	0.354	0.183	0.45	0.63	0.529	0.585
BENCHMUTATOR (Mistral)	0.68	0.48	0.717	0.603	0.60	0.63	0.656	0.772	0.14	0.09	0.591	0.337	0.96	0.93	0.931	0.930	0.46	0.43	0.343	0.214	0.44	0.43	0.532	0.585
Llama																								
Typo swap	0.43	0.56	0.555	0.314	0.33	0.26	0.527	0.328	–	–	–	–	0.81	0.67	0.819	0.650	0.33	0.33	0.743	0.403	0.45	0.35	0.532	0.515
Change char case	0.43	0.56	0.548	0.300	0.31	0.31	0.500	0.333	–	–	–	–	0.80	0.70	0.825	0.667	0.43	0.43	0.743	0.563	0.45	0.35	0.530	0.520
French back translation	0.43	0.58	0.300	0.300	0.32	0.21	0.511	0.285	–	–	–	–	0.80	0.60	0.837	0.656	0.35	0.35	0.745	0.432	0.43	0.39	0.532	0.508
Swap character	0.43	0.59	0.555	0.321	0.31	0.29	0.511	0.387	–	–	–	–	0.79	0.67	0.837	0.650	0.39	0.39	0.742	0.495	0.45	0.37	0.532	0.527
Synonym substitution	0.43	0.33	0.548	0.307	0.32	0.25	0.511	0.333	–	–	–	–	0.71	0.71	0.808	0.621	0.38	0.38	0.714	0.643	0.44	0.36	0.532	0.522
Function renaming	0.43	0.44	0.548	0.535	–	–	–	–	0.29	0.15	0.743	0.564	–	–	–	–	–	–	–	–	–	–	–	–
Variable renaming	0.42	0.44	0.535	0.521	–	–	–	–	0.29	0.15	0.714	0.529	–	–	–	–	–	–	–	–	–	–	–	–
DyCodeEval	0.41	0.48	0.553	0.554	0.62	0.83	0.53	0.39	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
CodeCleaner	–	–	–	–	–	–	–	–	0.18	0.20	0.714	0.371	–	–	–	–	–	–	–	–	–	–	–	–
Self-Evolving	–	–	–	–	–	–	–	–	–	–	–	–	0.72	0.77	0.886	0.398	–	–	–	–	–	–	–	–
DyVal	–	–	–	–	–	–	–	–	–	–	–	–	0.58	0.56	0.363	0.377	–	–	–	–	–	–	–	–
BENCHMUTATOR (DeepSeek)	0.33	0.56	0.487	0.287	0.43	0.76	0.420	0.559	0.20	0.25	0.319	0.451	0.94	0.84	0.915	0.885	0.40	0.53	0.278	0.192	0.40	0.65	0.435	0.475
BENCHMUTATOR (Mistral)	0.27	0.28	0.462	0.287	0.40	0.55	0.420	0.529	0.29	0.19	0.345	0.302	0.95	0.87	0.897	0.852	0.38	0.44	0.299	0.202	0.39	0.40	40299000	0.475
Qwen																								
Typo swap	0.71	0.54	0.821	0.448	0.48	0.40	0.550	0.485	–	–	–	–	0.82	0.84	0.856	0.739	0.44	0.44	0.570	0.285	0.45	0.41	0.442	0.445
Change char case	0.71	0.56	0.828	0.400	0.44	0.55	0.550	0.512	–	–	–	–	0.88	0.82	0.850	0.739	0.44	0.44	0.572	0.394	0.46	0.43	0.445	0.447
French back translation	0.69	0.54	0.400	0.400	0.48	0.28	0.555	0.442	–	–	–	–	0.90	0.84	0.856	0.727	0.48	0.48	0.569	0.339	0.56	0.41	0.445	0.442
Swap character	0.71	0.56	0.828	0.545	0.48	0.45	0.560	0.496	–	–	–	–	0.85	0.86	0.844	0.733	0.43	0.43	0.571	0.391	0.45	0.39	0.445	0.455
Synonym substitution	0.69	0.48	0.828	0.414	0.47	0.58	0.200	0.200	–	–	–	–	0.88	0.88	0.842	0.779	0.47	0.47	0.464	0.393	0.45	0.45	0.445	0.450
Function renaming	0.71	0.70	0.821	0.786	–	–	–	–	0.09	0.02	0.743	0.683	–	–	–	–	–	–	–	–	–	–	–	–
Variable renaming	0.70	0.70	0.819	0.731	–	–	–	–	0.09	0.00	0.701	0.529	–	–	–	–	–	–	–	–	–	–	–	–
DyCodeEval	0.74	0.78	0.776	0.818	0.90	0.92	0.200	0.200	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
CodeCleaner	–	–	–	–	–	–	–	–	0.06	0.00	0.743	0.343	–	–	–	–	–	–	–	–	–	–	–	–
Self-Evolving	–	–	–	–	–	–	–	–	–	–	–	–	0.66	0.40	0.876	0.576	–	–	–	–	–	–	–	–
DyVal	–	–	–	–	–	–	–	–	–	–	–	–	0.67	0.05	0.545	0.482	–	–	–	–	–	–	–	–
BENCHMUTATOR (DeepSeek)	0.74	0.76	0.949	0.646	0.51	0.78	0.715	0.730	0.08	0.03	0.880	0.476	0.91	0.82	0.952	0.934	0.39	0.48	0.355	0.237	0.46	0.68	0.517	0.527
BENCHMUTATOR (Mistral)	0.74	0.50	0.853	0.668	0.53	0.62	0.670	0.774	0.03	0.09	0.901	0.254	0.92	0.78	0.988	0.983	0.38	0.40	0.362	0.165	0.47	0.45	0.522	0.527

Table 8: **Performance before and after fine-tuning on original benchmarks, per target model family.** *Base Orig* and *Base Mut* are scores on the original and mutated benchmarks before any fine-tuning. *FT Orig* and *FT Mut* are scores after 3 epochs of fine-tuning on the original benchmark instances.

Fine-tuning effect and gap widening. We observe that for across all model families, *FT Orig* is consistently higher than *Base Orig*, confirming the contamination induction effectiveness. Crucially, the gap between *FT Orig* and *FT Mut* widens after fine-tuning in most cells, confirming that the contaminated model becomes more sensitive to mutation: Self-Evolving on GSM8K for Llama shows the most extreme case (0.886 vs. 0.398, a gap of 0.488 against -0.05 before fine-tuning), and BENCHMUTATOR (DeepSeek) on HumanEval for Qwen reaches 0.949 original vs. 0.646 mutated.

Disclosure varies by task and system. On TruthfulQA, *FT Orig* rises sharply (Llama under swap character: $0.33 \rightarrow 0.743$) while *FT Mut* does not follow, indicating mutations effectively disrupt memorised surface patterns. BENCHMUTATOR maintains a controlled and meaningful gap on most benchmarks and model families, consistent with the contamination resistance scores reported in Section 5.4.

A.3 CROSS-REPLICATE SIMILARITY

As stated in Section 5, we generate three mutated benchmarks for each system and dataset considered. In this section, we compute the pairwise TF-IDF cosine similarity between them, yielding a 3×3 similarity matrix per cell as illustrated in the Figure 3.

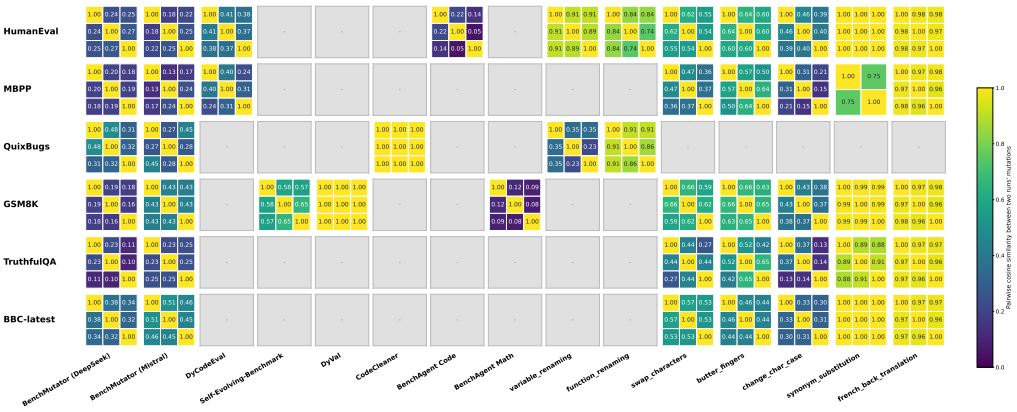

Figure 3: Pairwise cosine similarity between the TF-IDF representations of the three independent mutation runs, per system and dataset.

Deterministic rule-based transformations (swap_characters, change_char_case, butter_fingers, french_back_translation) produce near-perfect cross-replicate similarity (≥ 0.97) across all datasets, confirming they explore a single fixed point in the benchmark neighbourhood rather than a region. Synonym substitution similarly collapses to perfect similarity on GSM8K (0.99) and BBC-latest (1.00). CodeCleaner on QuixBugs reaches 1.00, consistent with its deterministic operators, while BenchAgent Math converges to a narrow generative mode on GSM8K (0.09–0.12) despite nominally sampling freely.

In contrast, BENCHMUTATOR produces substantially lower cross-replicate similarity than all baselines, confirming that agent-selected rules generate genuinely different transformations across runs. On HumanEval, off-diagonal values reach 0.24–0.27 (DeepSeek) and 0.18–0.25 (Mistral); on GSM8K, 0.16–0.19 (DeepSeek) and 0.43 (Mistral). DyCodeEval occupies a middle ground on HumanEval (0.38–0.41), more variable than rule-based systems but less so than BENCHMUTATOR.

A.4 AGENT CONFIGURATION

As reminder, BENCHMUTATOR's pipeline is a fixed sequence of five LLM agents, each a thin wrapper around a shared `LLMClient` with its own system prompt and sampling temperature (Table 9). The ANALYSISAGENT refines the task's constraints and input/output space from a sample of originals; the TRANSFORMATIONRULEAGENT designs general mutation rules from that analysis, calibrated against a probe set (Section 4) and refined via the FEEDBACKSYNTHESISAGENT when the probe pass rate falls below 0.5; the TRANSFORMATIONAGENT applies a rule to one sample; the VALIDATIONAGENT checks the result against task constraints and complexity; the OUTPUTUPDATEAGENT determines whether the expected output and correctness checks need updating to match the mutated input. Every agent responds in a fixed JSON schema, parsed and validated downstream.

The generator model is swapped per experiment (`deepseek/deepseek-v3.2` or `mistralai/mistral-small-24b-instruct-2501`, both via OpenRouter) while every other agent setting is held fixed, isolating the generator as the only varying factor when comparing BENCHMUTATOR (DeepSeek) against BENCHMUTATOR (Mistral) throughout the paper. Sampling of probe/full-benchmark items uses a Cochran finite-population formula (confidence 0.80, margin of error 0.10, capped at 400 items); the orchestrator allows up to 3 rule-refinement iterations and 1 retry per sample.

Agent	Role	Temp.	System prompt
AnalysisAgent	Refines task constraints and input/output space definitions from sampled examples.	0.3	Refine the task constraints and input-space definition based on the analysis of the provided sampled examples and the task description, taking into consideration memorization-pattern disclosure.
TransformationRuleAgent	Designs general, strong mutation rules; refines them from calibration feedback.	0.5–0.8	Design general transformation rules. You can deeply restructure the problem so that a model that has memorized the original sample cannot recognize or reuse the memorized answer.
FeedbackSynthesisAgent	Synthesises checker failures into rule-refinement advice.	0.3	Identify systematic failure patterns and give actionable advice so TransformationRuleAgent can produce better rules. Consider Technical Correctness and Mutation Quality.
TransformationAgent	Applies one rule to one sample, producing the modified input.	0.8	Analyse the given sample and modify its input according to the provided transformation rule, while ensuring the modified input conforms to the task's input space definition.
OutputUpdateAgent	Determines whether the expected output/tests need updating post-mutation.	0.3	Determine whether the expected output and any correctness checks still hold after a transformation is applied, or if they need updating to match the modified input.

Table 9: The five pipeline agents, their role, LLM sampling temperature, and the task-specific instruction from each system prompt

A.5 Task Constraints

Each task definition given to the AnalysisAgent (Section A.4) has four parts: a natural-language description, a list of *initial* constraints the mutated input must satisfy (refined into the enhanced constraints of Section 3.1 during calibration), an input-space definition, and an output-space definition. Four tasks are predefined and reused verbatim across benchmarks; three benchmarks (GSM8K, TruthfulQA, BBC-latest) use a custom task definition instead, since they are not code tasks. Table **??** gives all four parts for all seven.

Task	Field	Definition
Code Synthesis	Description	Given a problem description, generate Python code that solves the problem; additionally generate modified versions of the problem description similar in reasoning and problem-solving approach.
	Initial constraints	• Preserve the core reasoning and solution approach. • May modify the problem's context. • May modify all discriminative details that let the problem be recognised (e.g. variable/function names). • Must not contain the associated code, only the problem statement. • Updated tests, if any, must follow assert function_name(input) == expected_output format with at least 5 cases.
	Input space	A textual problem description stating the problem to solve in Python; modified inputs should share the same problem statement even under a different context.
	Output space	Python code that solves the provided problem description.
Program Repair	Description	Given a code snippet that contains a bug, repair the code to fix the bug; additionally generate modified versions of the buggy code snippet.
	Initial constraints	• May add bugs that alter the main purpose deducible from the function name. • May add vulnerabilities or other syntax inconsistencies. • May add unrelated code while preserving non-functional aspects. • Must preserve a genuine, test-detectable bug – whatever tests accompany the mutated sample, the mutated buggy program must still fail at least one of them, and only the corrected solution passes all of them.
	Input space	Faulty code that doesn't do what it's supposed to do – including functional bugs, logical bugs, syntax errors, and security vulnerabilities.
	Output space	Valid code that does what it's supposed to do; kept the same as the original output.
MMLU	Description	Given a multiple-choice question, answer it correctly; additionally generate modified versions of the question similar in reasoning and problem-solving approach.
	Initial constraints	• Must remain a valid multiple-choice question with the answer among the options. • Option order may be shuffled. • Incorrect options may be replaced with plausible alternatives. • The question may be rephrased but must test the same knowledge.
	Input space	A multiple-choice question with four options (A, B, C, D); modified inputs should test the same concept with different phrasing or option ordering.
	Output space	The correct answer option letter (A, B, C, or D) for the modified question.
GSM8K	Description	Given a math problem, reason about the problem and provide the answer.
	Initial constraints	• Preserve the core reasoning and problem-solving approach. • May use an affine function on the problem data and answer to keep the problem solvable. • Must not contain the associated answer, only the problem statement.
	Input space	A math problem statement.
	Output space	The answer to the math problem in number format; the reasoning process is optional.
TruthfulQA	Description	A short factual question is given, and the task is to provide a concise, truthful answer to the question.
	Initial constraints	• The correct answer is unchanged. • The question stays answerable from the same underlying fact. • The context or language of the question may change, but the core fact being asked about must remain the same. • Must not hint at or state the answer inside the question.
	Input space	A short natural-language factual question.
	Output space	A concise, truthful answer to the question.
BBC-latest	Description	A short reading-comprehension question is given along with a news passage, and the task is to provide a concise, truthful answer to the question.
	Initial constraints	• The question can differ but must still be answerable from the passage. • The passage can be paraphrased, but the core facts must remain the same. • Must not remove information needed to answer the question. • Must not include the answer in the passage or question.
	Input space	A news passage followed by a question about it.
	Output space	The answer to the question, grounded in the passage.

Table 10: Task definitions provided to the AnalysisAgent, verbatim from the pipeline configuration. Initial constraints are the starting list handed to the pipeline, not the enhanced constraints the AnalysisAgent later derives from them (Section 3.1).