Diff-XYZ: A Benchmark for Evaluating Diff Understanding

Evgeniy Glukhov
JetBrains Research
Amsterdam, the Netherlands
evgeniy.glukhov@jetbrains.com

Michele Conti
JetBrains Research
Amsterdam, the Netherlands
michele.conti@jetbrains.com

Egor Bogomolov
JetBrains Research
Amsterdam, the Netherlands
egor.bogomolov@jetbrains.com

Yaroslav Golubev
JetBrains Research
Belgrade, Serbia
yaroslav.golubev@jetbrains.com

Alexander Bezzubov
JetBrains Research
Amsterdam, the Netherlands
alexander.bezzubov@jetbrains.com

Abstract

Reliable handling of code diffs is central to agents that edit and refactor repositories at scale. We introduce *Diff-XYZ*, a compact benchmark for code–diff understanding with three supervised tasks: *apply* (old code + diff $\rightarrow$ new code), *anti-apply* (new code – diff $\rightarrow$ old code), and *diff generation* (new code – old code $\rightarrow$ diff). Instances in the benchmark are triples $\langle$*old code*, *new code*, *diff*$\rangle$ drawn from real commits in CommitPackFT, paired with automatic metrics and a clear evaluation protocol. We use the benchmark to do a focused empirical study of the unified diff format and run a cross-format comparison of different diff representations. Our findings reveal that different formats should be used depending on the use case and model size. For example, representing diffs in search-replace format is good for larger models in the diff generation scenario, yet not suited well for diff analysis and smaller models. The Diff-XYZ benchmark is a reusable foundation for assessing and improving diff handling in LLMs that can aid future development of diff formats and models editing code. The dataset is published on HuggingFace Hub: https://huggingface.co/datasets/JetBrains-Research/diff-xyz.

1 Introduction

Modern code-capable language models increasingly interact with repositories by both analyzing and generating code diffs. In tasks such as issue resolution [5; 17], CI build repair [6], or bug repair [13], the model's task is to produce a patch for an existing codebase that is then evaluated for correctness. Tasks like commit message generation treat diffs as an input that a model should analyze to generate a summary [11; 29].

There exists a multitude of formats for representing code diffs. Classical diff formats include the (i) normal format – changes without surrounding context, (ii) context format – changes with surrounding

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Deep Learning for Code in the Agentic Era.

lines, and (iii) unified format[1] – a compact, context-carrying variant with `@@` hunk headers and +/- prefixes (see the example in Figure 6). These three formats are standardized in GNU diffutils [21] and remain the basis for machine-applicable patches. In modern tooling, `git diff` produces a Git patch similar to unified diff but with additional Git-specific headers.

While benchmarks such as SWE-bench evaluate patches in the unified diff format, the choice of the diff representation used by an LLM can influence the quality of generations and their cost. For example, Aider reports that switching from a search/replace scheme to unified diffs cuts unhelpful outputs and raises benchmark scores [1]. OpenAI trains GPT-4.1 to generate and apply diffs in a recommended *V4A* patch format, and reports large gains on Aider's polyglot diff benchmark [2; 23]. Existing works [25; 27; 28] report using different approaches, such as working with unified diffs, performing rewrites on specified ranges, supporting search-replace formats that specify anchored replacements.

Despite this centrality, current evaluations make it hard to isolate the effect of diff format from other factors like retrieval or tool use. Wang et al. [26] analyze SWE-bench and highlight that there exist distinct failure modes: a patch may fail because it is syntactically malformed, because it does not apply to the intended context, or because it applies but does not fix the issue. This shows that patch outcomes depend on more than just representation, making it difficult to isolate and evaluate the impact of diff formats within such end-to-end benchmarks.

To enable studies targeting diff representations, we introduce Diff-XYZ, a focused lightweight benchmark for exploring how well models work with various diff formats. Our benchmark allows varying the diff representation while keeping the rest of the context fixed. Diff-XYZ consists of three synthetic tasks that represent three possible problems of finding an unknown in the equation *diff* $=$ *new code* $-$ *old code*. We hypothesize that solving complex generation and analysis tasks involving code diffs requires first mastering simpler variants of the problem. These include cases where two elements of the equation *new code* $-$ *old code* $=$ *diff* are provided and the third must be inferred. Rewriting it as $X - Y = Z$, we obtain the following list of tasks.

X. *Apply Task — new code* is unknown. This tests format obedience and character-level fidelity. The model must analyze a diff in a given format and realize the exact edits, including whitespace and ordering.

Y. *Anti-Apply Task — old code* is unknown. This probes invertibility and losslessness of the chosen format as processed by the model. The model must reconstruct deletions and replacements precisely and align modified regions back to their original spans. This is a strict operational check of diff understanding that goes beyond surface copying.

Z. *Diff Generation Task — diff* is unknown. This measures reliable diff synthesis: correct formatting and minimal, parseable edits. The outcome is directly relevant to code agents and patch-based evaluations where the system must emit a patch that tools can apply and reviewers can inspect.

First, we apply Diff-XYZ to analyze how well proprietary and open-source LLMs work with the most widespread unified diff format. Proprietary models consistently outperform open-source models, with Claude 4 Sonnet and GPT-4.1 achieving the highest scores across the tasks; GPT-4.1 is more sensitive to a task prompt and tends to emit V4A diff format by default. For open models, scores improve as model size grows, yet performance is still short of strong results, especially for smaller open models. Diff-XYZ can help by providing focused, reproducible checks that highlight where formats and prompts make the most difference. This analysis clarifies model behavior on the benchmark tasks and establishes strong reference results. Building on these findings, we then compare alternative edit representations: udiff-based formats work best for Apply and Anti-Apply, search-replace excels for Diff Generation, but for smaller models modified udiff variants perform best.

2 Data Collection

We construct *Diff-XYZ*, a compact benchmark of 1,000 real-world code edits derived from the CommitPackFT dataset [22], a large-scale corpus of open-source commits with paired before/after code and commit metadata. Each instance in Diff-XYZ is a triple $\langle$*old code*, *new code*, *diff*$\rangle$, where

[1]Throughout the work, we use *unified diff* or *udiff* for all the diffs that use the unified diff syntax.

diff is computed between the two code versions. This dataset serves as the common basis for all tasks presented in the next section.

To ensure diversity and quality, we apply systematic filtering and sampling. We retain only commits that change a single file and exclude binary files, generated code, vendor directories, and trivial whitespace-only changes. Changed files must contain at least 40 lines and no more than 1,000 lines in at least one version of the code (old or new). To balance edit complexity, we stratify commits by the number of changed hunks and change size (added + removed lines), targeting a 50/50 split between single-hunk and multi-hunk edits and a 40/40/20 distribution of small, medium, and large edits, defined by the 40th and 80th percentiles of change size (added + removed lines) within each bucket. In practice, edits with at most 7 changed lines are considered small (40th percentile), those with 8-24 lines medium (40th-80th percentile), and those with more than 24 lines large. We also cap the number of examples per repository at 5 to encourage repository diversity.

The final dataset covers five programming languages, with 200 examples each: Python, JavaScript, and Java, which are widely used and appear in most existing benchmarks, and Kotlin and Rust, which are comparatively underrepresented but actively used in distinct contexts.

Most edits include both additions and deletions (81.5%), while a minority are add-only (16.3%) or delete-only (2.2%). In total, Diff-XYZ spans 891 unique repositories.

3 Tasks Description

3.1 Apply Task and Anti-Apply Task

Apply is a straightforward task: given the old code and the diff, generate the new code. It can be viewed as a special case of code generation task with all the needed information being already in the prompt, yet the model has to correctly interpret the diff to solve it. If an LLM is capable of understanding the diff and copying, we expect it to solve the task accurately, with larger models solving it near-perfectly. In case a model scores low on the Apply task, it may require additional fine-tuning to work with diffs or an adaptation of the diff format. Note that Apply is a natural simplification of the recently introduced problem of smart paste or fuzzy diff application with LLMs [12], *i.e.,* generate a new code state after applying a patch in some insufficient format.

Anti-Apply is the problem complementary to the Apply task: given the new code and the diff, generate the old code. Similar to the Apply task, all the data required to solve it is in the prompt. In case there is a significant difference between the Apply and Anti-Apply scores for a model, it is a sign of model's overfitting on one of the tasks.

We evaluate performance on these tasks with slightly modified standard metrics. To calculate the following metrics for code snippets, we remove all the lines that contain only whitespace characters.

- *Stripped Exact Match* — **EM** — is 1 when two processed code snippets are exactly the same, 0 otherwise.
- *Stripped Intersection over Union for lines* — **IoU** — ratio of unique lines in intersection to unique lines in union for two processed code snippets.

3.2 Diff Generation Task

Diff Generation Task is formulated as follows: given the old code and the new code, generate the diff in the specified format. This task is different from the first two, as it does not require generating long code sequences but rather shorter strictly formatted diffs.

The choice of format is crucial for the Diff Generation Task, as the model may be more or less aware of the given format. The unified diff format is the most common format used by GitHub and SWE-bench, and LLMs have higher chances of seeing it during pretraining. This may be not the case for other formats, so we always add a format description in the system prompt.

Unlike Apply and Anti-Apply, diff calculation is not uniquely defined. Even for the same code change, different but valid diffs can be produced, depending on factors such as the number of context lines. Because of that, we cannot apply similarity metrics like EM or IoU to the Diff Generation task. To overcome this problem, we evaluate EM and IoU for the *new code* and the code that is the

result of applying the generated diff to the *old code*. Because diff application is a strict, all-or-nothing procedure, EM and IoU after application only capture exact correctness when the generated diff applies successfully. To measure partial correctness even when application fails, we also compute F1-scores over the sets of added and deleted lines [20].

Finally, the standard unified diff hunk header contains line numbers, which are used to resolve uncertainties. However, it can be challenging to generate a hunk header before the hunk itself, and such uncertainties represent less than 1% of the dataset. For this reason, we ignore hunk headers during patch application when they are not needed.

Here is the formal definition for metrics that we use to evaluate the Diff Generation task.

- *Parsing Rate* and *Applying Rate* — the fraction of generated diffs that can be parsed, and the fraction that can be successfully applied to the old code.
- *EM* and line-level *IoU* after application —– EM and IoU values of the new code and the code after applying diff to the old code, 0 if the generated diff cannot be parsed.
- *F1-score on Addition Lines* — **F1+** — F1-score between the set of added lines in the generated diff and those in the reference.
- *F1-score on Deletion Lines* — **F1–** — F1-score between the set of deleted lines in the generated diff and those in the reference.

4 Unified Diff Evaluation

In this section, we provide an extensive evaluation for the unified diff format on Diff-XYZ. The results are organized in two subsections: proprietary models and open-source models.

All models are evaluated with a fixed user prompt for each task (see Appendix A.2). We vary the system prompt in two ways:

1. **w/o format** —– a generic system prompt ("You are a helpful assistant.") with no explicit description of the edit representation.
2. **w/ format** — a system prompt that explicitly defines the unified diff format (see Figure 2 in Appendix A.1)

4.1 Proprietary Models

Proprietary LLMs are usually more powerful than open-source ones [3; 10; 16; 23], and since the tasks are conceptually simple, we expect the best models to achieve near-perfect scores.

Table 1 shows the results for the Apply and Anti-Apply tasks on the Diff-XYZ dataset. Claude 4 Sonnet outperforms all models, making almost no mistakes on Apply under either system prompt, though it shows some quality drop on Anti-Apply. GPT-4.1 also performs very well across both tasks, but its results vary more across prompts. For some models (*e.g.,* GPT-4.1), quality drops on Apply when moving from the *w/o format* to the *w/ format* system prompt, while the scores on Anti-Apply remain unaffected. Such inconsistencies may indicate overfitting: the model focuses on the diff description and outputs a diff instead of code (see Figure 10 in Appendix B).

Although diff application is a conceptually simple task, only the strongest proprietary models solve it perfectly. Open-source models, and even some smaller proprietary ones, still make mistakes. This confirms that the benchmark is well-calibrated: it exposes meaningful differences in models' ability to interpret diff syntax, align edits with code, and produce the correct target version exactly.

Table 2 shows the results for the Diff Generation task. Here, the effect of the system prompt is much stronger: explicitly describing the format narrows the model's choices and reduces the likelihood of switching to alternative diff formats. This effect is especially noticeable for GPT 4.1, which, without explicit instructions, frequently outputs diffs in V4A, the format it was trained to produce, rather than unified diff (see Figure 11 in Appendix B).

We observe that the Apply Rate is nearly identical to the IoU across models, indicating that whenever a model produces an applicable diff, it is usually very close to the expected result.

Table 1: Results of closed models on the Apply and Anti-Apply tasks on Diff-XYZ.

Model	Prompt	Apply		Anti-Apply	
		EM	IoU	EM	IoU
GPT 4o	w/o format	0.83	0.97	0.88	0.97
GPT 4o	w/format	0.87	0.98	0.89	0.98
GPT 4o-mini	w/o format	0.70	0.96	0.57	0.85
GPT 4o-mini	w/format	0.05	0.12	0.34	0.49
GPT 4.1	w/o format	0.92	0.98	0.93	0.98
GPT 4.1	w/format	0.81	0.86	0.95	0.98
GPT 4.1 mini	w/o format	0.90	0.99	0.86	0.97
GPT 4.1 mini	w/format	0.90	0.98	0.85	0.97
GPT 4.1 nano	w/o format	0.38	0.85	0.03	0.63
GPT 4.1 nano	w/format	0.29	0.58	0.00	0.08
Claude 4 Sonnet	w/o format	0.95	0.99	0.87	0.90
Claude 4 Sonnet	w/format	0.96	0.99	0.87	0.89
Gemini 2.5 Flash	w/o format	0.91	0.97	0.77	0.85
Gemini 2.5 Flash	w/format	0.93	0.98	0.81	0.85

Table 2: Results of closed models on the Diff Generation task on Diff-XYZ.

Model	Prompt	EM	IoU	f1+	f1–	Apply Rate	Parsing Rate
GPT 4o	w/o format	0.34	0.50	0.72	0.51	0.52	0.89
GPT 4o	w/format	0.33	0.53	0.79	0.52	0.55	0.98
GPT 4o-mini	w/o format	0.17	0.38	0.65	0.36	0.40	0.92
GPT 4o-mini	w/format	0.16	0.27	0.60	0.33	0.28	0.90
GPT 4.1	w/o format	0.34	0.35	0.34	0.25	0.35	0.44
GPT 4.1	w/format	0.76	0.78	0.76	0.52	0.79	0.99
GPT 4.1 mini	w/o format	0.11	0.12	0.08	0.08	0.12	0.15
GPT 4.1 mini	w/format	0.60	0.64	0.52	0.33	0.78	0.65
GPT 4.1 nano	w/o format	0.25	0.34	0.26	0.14	0.35	0.47
GPT 4.1 nano	w/format	0.21	0.27	0.18	0.09	0.28	0.53
Claude 4 Sonnet	w/o format	0.64	0.66	0.73	0.57	0.67	0.80
Claude 4 Sonnet	w/format	0.85	0.89	0.92	0.71	0.89	1.00
Gemini 2.5 Flash	w/o format	0.69	0.79	0.82	0.61	0.80	0.93
Gemini 2.5 Flash	w/format	0.66	0.76	0.80	0.59	0.77	0.90

4.2 Open Source Models

To analyze how model size affects diff understanding, we evaluate the Qwen2.5-Coder series [15], a family of open-source, code-focused LLMs ranging from 0.5B to 32B parameters. Qwen2.5-Coder models achieve near state-of-the-art results on standard code generation and reasoning benchmarks among open-source models, making them a strong testbed for scaling analysis on Diff-XYZ. Tables 3 and 4 report results for the Apply/Anti-Apply and Diff Generation tasks, respectively.

We observe a clear scaling trend: performance improves steadily with model size. On Apply and Anti-Apply, reliable results emerge around the 7B scale, with the largest Qwen Coder approaching GPT-4o. In contrast, none of the open-source models achieve comparable performance on Diff Generation. This gap suggests that handling diff syntax and formatting requires substantially more capacity than simply applying edits. It may also help explain why smaller models perform poorly on complex downstream benchmarks such as SWE-bench, where correctly generating patches is critical.

Table 3: Results of the Qwen Coder family of models on the Apply and Anti-Apply tasks on Diff-XYZ with *w/format* system prompt.

Model	Apply		Anti-Apply	
	EM	IoU	EM	IoU
Qwen2.5-Coder-0.5B-Instruct	0.00	0.39	0.00	0.46
Qwen2.5-Coder-1.5B-Instruct	0.11	0.39	0.05	0.70
Qwen2.5-Coder-3B-Instruct	0.36	0.81	0.15	0.71
Qwen2.5-Coder-7B-Instruct	0.59	0.94	0.64	0.93
Qwen2.5-Coder-14B-Instruct	0.82	0.97	0.82	0.97
Qwen2.5-Coder-32B-Instruct	0.85	0.98	0.86	0.98

Table 4: Results of the Qwen Coder family of models on the Diff Generation task on Diff-XYZ with *w/format* system prompt.

Model	EM	IoU	f1+	f1–	Apply Rate	Parsing Rate
Qwen2.5-Coder-0.5B-Instruct	0.00	0.00	0.01	0.02	0.01	0.23
Qwen2.5-Coder-1.5B-Instruct	0.01	0.04	0.07	0.10	0.04	0.69
Qwen2.5-Coder-3B-Instruct	0.00	0.12	0.18	0.15	0.15	0.72
Qwen2.5-Coder-7B-Instruct	0.03	0.17	0.29	0.24	0.19	0.70
Qwen2.5-Coder-14B-Instruct	0.14	0.35	0.47	0.37	0.38	0.92
Qwen2.5-Coder-32B-Instruct	0.24	0.47	0.61	0.50	0.50	0.99

5 Diff Format Exploration

While the unified diff format is the most widely used, it is not the only way to represent edits. Different formats impose different structural constraints and tokenization patterns, which can affect how easily models generate, parse, and apply them. To probe these effects, we compare several representations side by side on Diff-XYZ. For each format, the system prompt included both a description of the format and one example (see Appendix A.3).

We evaluate the following formats:

1. *udiff*: the standard unified diff (Figure 6).
2. *udiff-h*: unified diff with a relaxed hunk header, written as `@@ ... @@` (Figure 7).
3. *udiff-l*: unified diff with verbose line markers `ADD`, `DEL`, `CON` instead of the single-character `+`, `-`, and a leading space (Figure 8).
4. *search-replace*: a sequence of edits where each *search* substring is replaced by a *replace* substring, following Aider [1] and Li et al. [18] (Figure 9).

We include *udiff* and *search-replace* because they are widely used. The two *udiff* variants address practical generation issues: *udiff-h* avoids committing to exact line numbers before the hunk body is produced, and *udiff-l* reduces ambiguity and token collisions by replacing single-character markers with explicit tags.

The results are presented in Table 5. For Diff Generation, *search-replace* is a strong default for most larger models, while *udiff-l* achieves the best scores for smaller models, a result that is both consistent and unexpected. Another unexpected finding is that *udiff-h* is typically much worse than *udiff*, despite only relaxing the hunk header and making a minimal change in format. For Apply and Anti-Apply, however, *search-replace* underperforms relative to more structured formats, highlighting a trade-off between ease of generating edits and faithfulness of application. These results suggest that while search–replace is attractive for generation, it is a poor fit for tasks where faithful application matters. This may also affect downstream tasks such as commit message generation, which rely on diffs as structured inputs. For such tasks, explicit udiff variants remain the safer and more reliable choice.

We also observe that the most effective representation depends on both model scale and task step (generation vs. application). We hypothesize three interacting causes:

Table 5: Results on Diff-XYZ for various diff formats. For each model, the best-performing format is highlighted per task: Apply, *Anti-Apply*, and **Diff Generation**. If a format is best in multiple tasks, the corresponding styles are combined (e.g., **udiff**).

Model	Diff Format	Apply	Anti-Apply	Diff Generation		
		EM	EM	EM	f+	f–
GPT 4o	udiff	0.86	0.85	0.43	0.89	0.83
	udiff-h	0.85	0.89	0.03	0.20	0.61
	udiff-l	0.82	0.88	0.03	0.11	0.25
	search-replace	0.57	0.60	0.74	0.93	0.89
GPT 4.1	udiff	0.90	0.88	0.77	0.92	0.91
	udiff-h	0.92	0.93	0.06	0.25	0.61
	udiff-l	0.91	0.88	0.06	0.21	0.39
	search-replace	0.57	0.56	0.95	0.97	0.94
GPT 4.1-mini	udiff	0.89	0.81	0.73	0.87	0.84
	udiff-h	0.86	0.85	0.03	0.28	0.62
	udiff-l	0.86	0.83	0.04	0.10	0.14
	search-replace	0.57	0.55	0.90	0.94	0.94
GPT 4.1-nano	**udiff**	0.37	0.02	0.51	0.81	0.78
	udiff-h	0.32	0.01	0.12	0.47	0.49
	udiff-l	0.44	0.04	0.09	0.10	0.11
	search-replace	0.33	0.01	0.05	0.07	0.07
Claude 4 Sonnet	udiff	0.95	0.82	0.82	0.95	0.94
	udiff-h	0.95	0.87	0.02	0.13	0.31
	udiff-l	0.95	0.93	0.15	0.34	0.55
	search-replace	0.57	0.48	0.94	0.97	0.96
Gemini 2.5 Flash	udiff	0.90	0.34	0.73	0.94	0.92
	udiff-h	0.91	0.54	0.10	0.22	0.28
	udiff-l	0.79	0.07	0.17	0.32	0.26
	search-replace	0.57	0.53	0.88	0.96	0.95
Qwen2.5-Coder-0.5B	*udiff*	0.01	0.01	0.00	0.04	0.02
	udiff-h	0.01	0.01	0.03	0.18	0.18
	udiff-l	0.00	0.01	0.23	0.57	0.42
	search-replace	0.00	0.00	0.00	0.04	0.22
Qwen2.5-Coder-1.5B	udiff	0.16	0.04	0.01	0.07	0.09
	udiff-h	0.16	0.05	0.20	0.45	0.42
	udiff-l	0.11	0.03	0.38	0.49	0.49
	search-replace	0.20	0.07	0.19	0.35	0.38
Qwen2.5-Coder-3B	udiff	0.38	0.17	0.01	0.20	0.22
	udiff-h	0.41	0.20	0.02	0.16	0.21
	udiff-l	0.34	0.13	0.36	0.54	0.55
	search-replace	0.28	0.25	0.14	0.30	0.22
Qwen2.5-Coder-7B	*udiff*	0.58	0.65	0.05	0.39	0.31
	udiff-h	0.57	0.64	0.01	0.17	0.28
	udiff-l	0.57	0.62	0.07	0.17	0.18
	search-replace	0.50	0.50	0.28	0.62	0.58
Qwen2.5-Coder-32B	*udiff*	0.84	0.87	0.23	0.73	0.63
	udiff-h	0.83	0.86	0.00	0.25	0.67
	udiff-l	0.83	0.82	0.03	0.18	0.39
	search-replace	0.57	0.53	0.68	0.92	0.86

1. **Local vs. global constraints.** *search-replace* avoids format-level global constraints such as predicting line numbers in hunk headers (e.g., `@@ -a,b +c,d @@`), matching hunk lengths, and preserving the number of context lines. Each replacement stands on its own, so an error in one edit does not invalidate the rest. Larger models are better at locating distinctive anchors and ordering small, local edits, which increases parse/apply success.
2. **Marker collisions.** Small models may confuse single-character udiff markers (`+`, `-`, leading space) with ordinary code characters. Replacing them with explicit tags (`ADD`/`DEL`/`CON`) in *udiff-l* makes the control tokens unambiguous and rare; the model mainly needs to decide, for each line, which tag to use and then emit the line, an easier decision for weaker models.
3. **Header scaffolding & distribution shift.** Standard udiff hunk headers (`@@ -a,b +c,d @@`) provide numeric anchors and implicit ordering cues. Even though our application step does not rely on these numbers, their presence appears to help models structure the patch: they act as scaffolding that encourages hunks to be segmented and emitted in file order. In *udiff-h*, this scaffold is removed, which not only introduces a distribution shift relative to pretraining corpora (most public diffs include numbers) but may also increase the frequency of hunks being emitted out of order (e.g., an edit to line 120 preceding one to line 90). Both effects make the resulting patches less reliable, even though the format change is minimal.

6 Limitations and Future Work

The tasks in Diff-XYZ are simplified proxies rather than full downstream applications. Establishing a quantitative link between performance on these tasks and outcomes in real systems such as commit message generation, automated bug fixing, or code review is left for future work. As a result, our findings should not be interpreted as direct predictions of production performance.

All experiments use non-reasoning LLMs under greedy decoding. Evaluation of tool use, multi-step reasoning, sampling strategies, or best-of-n decoding, is an important question that is out of scope of this work. The reported numbers therefore reflect single-pass behavior and likely represent conservative estimates rather than achievable upper bounds.

The set of edit representations we study is limited to the most popular ones. Some alternatives remain unexplored, including richer tree- or AST-based patches, structured search-replace variants, and error-tolerant or partially specified formats. Note, that they can be integrated into Diff-XYZ without modifying the benchmark.

7 Related Work

Code generation benchmarks. A long line of benchmarks evaluate LLMs on code generation from natural language. Chen et al. [9] and Austin et al. [4] introduced HumanEval and MBPP, which remain the most widely used, with extensions such as HumanEval+ and MBPP+ [19], HumanEval-XL [24], and multilingual variations such as MultiPL-E [7]. BigCodeBench [30] emphasizes more complex and library-heavy tasks. These benchmarks primarily test function-level synthesis and correctness, but do not address how edits are represented or applied.

Editing and issue-resolution benchmarks. More recent work has evaluated language models in editing settings. Cassano et al. [8] introduces instruction-following edits, and Guo et al. [14] evaluate debugging, polishing and translation tasks. At a larger scale, Jimenez et al. [17] proposed SWE-bench, which tasks models with resolving real GitHub issues by producing patches that apply and pass tests. These benchmarks reflect realistic workflows, but include many factors, from retrieval and long context reasoning to semantic correctness and patch formatting. These benchmarks capture realistic workflows, including patch application, but their evaluation mixes retrieval, long-context reasoning, semantic correctness, and patch formatting — making it difficult to isolate the role of edit representation.

Positioning. Despite this breadth, existing benchmarks all assume a fixed diff format, typically unified diff. In reality, there are multiple representations in use: normal diff, context diff, unified diff, OpenAI's V4A diff format [23], and search/replace schemes, including anchored replacements used by some agents. Format is therefore a real variable, not a constant. Aider's benchmarks [2]

are notable for reporting differences between formats (whole file rewrite, search/replace, unified diff), but their evaluation was motivated by tool design choices rather than a systematic study of edit representations.

Our benchmark takes a complementary approach: it decomposes the end-to-end code editing pipeline problem and focuses specifically on edit representation in isolation. By holding task context fixed, it enables us to measure how models handle alternative formats under controlled conditions. This design also makes evaluation lightweight and cheap: unlike agentic frameworks such as SWE-bench, it requires no repository setup or execution harness, yet still targets a core capability that directly impacts downstream systems.

8 Conclusion

We introduced Diff-XYZ, a benchmark for evaluating LLMs' ability to handle code diffs across multiple formats. The tasks isolate core subproblems that arise in downstream systems while remaining simple enough to serve as controlled probes for more complex workflows.

Our main contribution is a set of focused tasks with clearly specified inputs and targets, paired with automatic metrics. This framework offers a reproducible setting for studying model behavior in diff-centric workflows. Possible extensions include connecting the benchmark to downstream tasks such as commit message generation, exploring corrupted or partial diff application, and incorporating structured code editing.

We also establish baselines for the unified diff format by measuring how LLMs process udiff inputs directly. Frontier models perform strongly on the provided tasks, suggesting the value of introducing more challenging instances within the same framework.

Beyond udiff, we compare several edit representations across a range of models. The resulting trade-offs can guide representation design. Future work will address the observed flaws and iterate toward formats that improve faithfulness and applicability.

References

[1] Aider. Unified diffs make GPT-4 turbo 3x less lazy, 2024. URL `https://aider.chat/docs/unified-diffs`.

[2] Aider. Aider polyglot benchmark, 2024. URL `https://aider.chat/2024/12/21/polyglot.html#the-polyglot-benchmark`.

[3] Anthropic. System card: Claude Opus 4 & Claude Sonnet 4, 2025. URL `https://www-cdn.anthropic.com/4263b940cabb546aa0e3283f35b686f4f3b2ff47.pdf`.

[4] 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.

[5] Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. *arXiv preprint arXiv:2505.20411*, 2025.

[6] Egor Bogomolov, Aleksandra Eliseeva, Timur Galimzyanov, Evgeniy Glukhov, Anton Shapkin, Maria Tigina, Yaroslav Golubev, Alexander Kovrigin, Arie Van Deursen, Maliheh Izadi, et al. Long code arena: a set of benchmarks for long-context code models. *arXiv preprint arXiv:2406.11612*, 2024.

[7] Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, et al. Multipl-e: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 49(7):3675–3691, 2023.

[8] Federico Cassano, Luisa Li, Akul Sethi, Noah Shinn, Abby Brennan-Jones, Anton Lozhkov, Carolyn Jane Anderson, and Arjun Guha. Can it edit? evaluating the ability of large language models to follow code editing instructions. In *Conference on Language Modeling (COLM)*, 2024.

[9] 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.

[10] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[11] Aleksandra Eliseeva, Yaroslav Sokolov, Egor Bogomolov, Yaroslav Golubev, Danny Dig, and Timofey Bryksin. From commit message generation to history-aware commit message completion. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 723–735. IEEE, 2023.

[12] Simone Forte and Marcus Revaj. Smart Paste for context-aware adjustments to pasted code, 2024. URL `https://research.google/blog/smart-paste-for-context-aware-adjustments-to-pasted-code/`.

[13] Gregory Gay and René Just. Defects4j as a challenge case for the search-based software engineering community. In *International Symposium on Search Based Software Engineering*, pages 255–261. Springer, 2020.

[14] Jiawei Guo, Ziming Li, Xueling Liu, Kaijing Ma, Tianyu Zheng, Zhouliang Yu, Ding Pan, Yizhi Li, Ruibo Liu, Yue Wang, et al. Codeeditorbench: Evaluating code editing capability of large language models. *arXiv preprint arXiv:2404.03543*, 2024.

[15] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[16] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[17] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024.

[18] Hongwei Li, Yuheng Tang, Shiqi Wang, and Wenbo Guo. Patchpilot: A stable and cost-efficient agentic patching framework. *arXiv preprint arXiv:2502.02747*, 2025.

[19] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36:21558–21572, 2023.

[20] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008. ISBN 0521865719.

[21] Jim Meyering and Paul Eggert. GNU diffutils, 2010. URL `https://www.gnu.org/software/diffutils/`.

[22] Niklas Muennighoff, Qian Liu, Armel Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro von Werra, and Shayne Longpre. Octopack: Instruction tuning code large language models. *arXiv preprint arXiv:2308.07124*, 2023.

[23] OpenAI. Introducing GPT-4.1 in the API, 2025. URL `https://openai.com/index/gpt-4-1/`.

[24] Qiwei Peng, Yekun Chai, and Xuhong Li. Humaneval-xl: A multilingual code generation benchmark for cross-lingual natural language generalization. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 8383–8394, 2024.

[25] Sergey Vakhreev. Refact.ai is now the #1 open-source AI agent on SWE-bench, 2025. URL `https://refact.ai/blog/2025/open-source-sota-on-swe-bench-verified-refact-ai/`.

[26] You Wang, Michael Pradel, and Zhongxin Liu. Are "solved issues" in SWE-bench really solved correctly? An empirical study. *arXiv preprint arXiv:2503.15223*, 2025.

[27] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE): 801–824, 2025.

[28] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

[29] Yuxia Zhang, Zhiqing Qiu, Klaas-Jan Stol, Wenhui Zhu, Jiaxin Zhu, Yingchen Tian, and Hui Liu. Automatic commit message generation: A critical review and directions for future work. *IEEE Transactions on Software Engineering*, 50(4):816–835, 2024.

[30] Terry Yue Zhuo, Vu Minh Chien, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations*, 2025.

A Inference Details

A.1 System prompts

```
You are a helpful assistant.
```

Figure 1: *w/o format* System Prompt

```
You are a helpful assistant.

When referred to unified diff format, the formatting must be
as follows:

Do NOT include or start with Git headers like diff -git
...  or index .....  Use POSIX unified diff with headers
-- <old> and +++ <new>, hunks @@ -old_start,old_count
+new_start,new_count @@.  Prefix context with space, removals
-, additions +.  1-based numbering, LF newlines, 1 context
lines.  New file:  -- /dev/null.  Deleted file:  +++
/dev/null.
```

Figure 2: *w/format* System Prompt

A.2 Task Prompts

````
You need to write a code that is a result of applying the
following diff in unified diff format to the following code
snippet:


Diff:
{diff}


Code:
{old_code}


Use triple backtick formatting for you answer (e.g.,
```python...```).
````

Figure 3: Prompt Template for the Apply Task

````
You are given a code snippet that results from applying a
unified diff.  Your task is to reconstruct the original
version of the code before the diff was applied.


Diff:
{diff}


Code After Applying the Diff:
{new_code}


Use triple backtick formatting for you answer (e.g.,
```python...```).
````

Figure 4: Prompt Template for the Anti-Apply Task

```
You need to write a diff in unified diff format that
transforms code snippet 1 to code snippet 2:


Code Snippet 1:
{old_code}


Code Snippet 2:
{new_code}


Use triple backtick formatting for you answer (e.g.,
```diff...```).
```

Figure 5: Prompt Template for the Diff Generation Task

A.3 Examples of Diff Formats

```
@@ -1,2 +1,4 @@
+import math
+
 def calculate_area(radius):
-    return 3.14159 * radius * radius
+    return math.pi * radius * radius
@@ -4,2 +6,0 @@
-def old_function():
-    return "deprecated"
```

Figure 6: Example of a diff in *udiff* format. This example is included in system prompt for diff exploration.

```
@@ ... @@
+import math
+
 def calculate_area(radius):
-    return 3.14159 * radius * radius
+    return math.pi * radius * radius
@@ ... @@
-def old_function():
-    return "deprecated"
```

Figure 7: Example of a diff in *udiff-h* format. This example is included in system prompt for diff exploration.
```

```
@@ -1,2 +1,4 @@
ADD import math
ADD
CON def calculate_area(radius):
DEL     return 3.14159 * radius * radius
ADD     return math.pi * radius * radius
@@ -4,2 +6,0 @@
DEL def old_function():
DEL     return "deprecated"
```

Figure 8: Example of a diff in *udiff-l* format. This example is included in system prompt for diff exploration.

```
<<<<<<< SEARCH
    return 3.14159 * radius * radius
=======
    return math.pi * radius * radius
>>>>>>> REPLACE

<<<<<<< SEARCH

def old_function():
    return "deprecated"
=======
>>>>>>> REPLACE
```

Figure 9: Example of a diff in *search-replace* format. This example is included in system prompt for diff exploration.

B Typical Errors

```
--- a/src/main.rs
+++ b/src/main.rs
@@ -20,5 +20,15 @@
  let mut num = 0;
+ let mut prev = 0;

  for c in roman_number.chars() {
-  num += roman_char_to_decimal(c);
+
+  let digit = roman_char_to_decimal(c);
+
+  num += digit;
+
+  if prev != 0 && digit > prev {
+   num -= 2 * prev;
+  }
+
+  prev = digit;
  }
```

Figure 10: Typical incorrect answer for GPT 4.1 with *w/format* system prompt in *Apply Task*. Diff instead of new code is generated for more than 10% of data points. For these data points, answer with *w/o format* system prompt is correct, and answer for Anti-Apply is correct with either system prompt.

```
--- snippet1.py 2024-06-13 12:00:00.000000000 +0000
+++ snippet2.py 2024-06-13 12:00:00.000000000 +0000
@@ setup(
-    classifiers=[
-        'Development Status :: 4 - Beta',
+    classifiers=[
+        'Development Status :: 5 - Production/Stable',
```

```
--- old
+++ new
@@ -18,7 +18,7 @@
     packages=['drupdates', 'drupdates.tests'],
     classifiers=[
-        'Development Status :: 4 - Beta',
+        'Development Status :: 5 - Production/Stable',
         'Environment :: Console',
         'Intended Audience :: Developers',
         'Intended Audience :: System Administrators',
```

Figure 11: **Top Box.** Typical incorrect answer for GPT 4.1 with *w/o format* system prompt in *Diff Generation Task*. **Bottom Box.** Answer with *w/format* system prompt. Incorrect diff is in V4A format instead of the unified diff format. Such answers are generated for more than 50% of data points.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Main claims in the abstract and introduction are an accurate summary of the results presented in the paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The separate "Limitations" section is in the paper. Additionally, local assumptions are highlighted in corresponding sections.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: Paper does not include theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: To the best of our knowledge, we have provided all the details necessary to reproduce the paper. If some steps are not reproducible, we are ready to provide additional details upon request.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Dataset with instructions is provided in supplementary materials.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Experimental setting is presented in the paper, with some details in Appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We do not provide any information such as error bars, confidence intervals, or statistical significance tests since it is not of main significance for the paper. We did all the necessary calculations and we are ready to provide the results upon request.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: The number of resources needed for all the experimental runs is not provided, however, it can be derived from all the provided details.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Paper conforms with NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: There is no societal impact of the work performed.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We followed the citation information provided by the authors of the datasets and models we use. We only use permissively licensed data in our work.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The documentation file for the dataset is provided.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.