electronics

MDPI

Article

How the Representation of Retrieved Context Affects In-Context Prompting for Commit Message Generation

Dokyeong An [1] and Geunseok Yang [2,*]

[1] Department of Computer Applied Mathematics, Hankyong National University, Anseong 17579, Republic of Korea; 2023810034@hknu.ac.kr
[2] Department of Computer Applied Mathematics, Computer System Institute, Hankyong National University, Anseong 17579, Republic of Korea
* Correspondence: gsyang@hknu.ac.kr

Abstract

High-quality commit messages are essential software artifacts because they succinctly communicate the intent and scope of code changes, yet large language models (LLMs) often fail to reflect project-specific writing conventions when used in a zero-shot setting without contextual signals. This study investigates not whether retrieval helps, but how the same retrieved example, when represented differently in the prompt, quantitatively changes generation outcomes. We implement a retrieve-then-generate framework where the target commit's diff is used as a query for BM25 (Best Matching 25)-based sparse retrieval over a commit-level database, and the top-1 similar commit is optionally injected as an example context. We compare a no-context condition (K = 0) against a minimal-context condition (K = 1) under three context representations: Diff-only, Message-only, and Diff + Message pair. Using Qwen-7B on 8000 evaluation samples with a fixed prompt skeleton, deterministic decoding, and identical post-processing across conditions, we observe negligible differences at K = 0 (BLEU-4 1.14, ROUGE-L 7.47–7.48, METEOR 4.88–4.91), establishing a stable baseline. At K = 1, the same top-1 retrieved case yields systematically different metric responses depending on how it is represented (Diff-only, Message-only, or Diff + Message), even under an identical prompt skeleton, deterministic decoding, and identical post-processing. This indicates that "context representation" is not a cosmetic formatting choice but a first-class prompt-design variable in retrieval-augmented in-context learning for commit message generation. Accordingly, practitioners should select the representation based on the intended objective (e.g., lexical/style alignment vs. change-intent grounding), rather than assuming a universally optimal format.

Keywords: automated commit message generation; similarity-based example retrieval; in-context prompting; context type analysis; large language models (LLMs)

Check for updates

Academic Editor: Weibin Wu

Received: 1 January 2026
Revised: 27 January 2026
Accepted: 29 January 2026
Published: 2 February 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Modern software development is commonly characterized by frequent code changes driven by large-scale collaboration, short release cycles, and the widespread adoption of continuous integration and deployment (CI/CD). In this setting, commit messages serve as a critical artifact that concisely summarizes the intent and scope of individual code changes. They provide foundational information for a wide range of maintenance activities, including change history tracing, code review, defect analysis, regression diagnosis, rollback decisions, and related tasks. As project size and change frequency grow, commit histories

become increasingly large, and clear single-line commit messages function as an essential means of documentation throughout software evolution.

In practice, empirical studies have reported that commit messages can be overly brief, underspecified, or stylistically inconsistent, which increases the cost of understanding changes and weakens the usefulness of commit logs [1–3].

To alleviate these issues, automatic commit message generation has been actively studied in software engineering. Early work relied on rule-based summarization or information-retrieval-driven template generation, while later approaches became dominated by neural machine translation (NMT) models that treat the code change (diff) as an input sequence and translate it into a natural-language message [4]. Nonetheless, diff text intermixes heterogeneous elements (e.g., file paths, identifiers, code blocks, and added/deleted lines) and exhibits substantial length variability. As a result, straightforward sequence-to-sequence formulations may fail to capture structural properties of code changes and often generalize poorly to change patterns that are relatively rare in training data.

More recently, the emergence of large language models (LLMs) capable of handling both code and natural language has enabled commit message generation in zero-shot or in-context learning settings without fine-tuning [5]. While this paradigm is attractive because it can be applied across projects without retraining, zero-shot generation still frequently fails to reflect project-specific writing rules and stylistic conventions in a stable manner. Fine-tuning can address this limitation, but it requires large training corpora and substantial computational cost, and it is often impractical for promptly incorporating evolving project conventions in operational settings. Thus, there remains a practical need for inference-time methods that can steer output consistency without updating model parameters.

In this context, recent studies have focused on retrieve-then-generate architectures that retrieve past commits similar to the input commit and provide them as prompt examples [6]. By injecting analogous cases as context, the model can indirectly reference project-specific summarization patterns and stylistic conventions. However, several prior studies mainly examine (i) whether examples are included and/or (ii) how performance changes as the number of examples increases [3,7,8].

This study addresses this gap by isolating and analyzing the effect of context representation rather than the mere presence of context. We construct a retrieve-then-generate framework in which the input commit's code change (diff) is used as a query for BM25-based sparse retrieval, and only the top-1 most similar commit is injected as contextual evidence [6,9,10].

Given the same retrieved result, we consider three context representations: Diff-only, which provides only the retrieved commit's code change; Message-only, which provides only the retrieved commit message; and Diff + Message pair, which provides both the retrieved diff and its message. We further contrast a no-context condition (K = 0) with a minimal-context condition that provides only a single top-1 similar example (K = 1) to disentangle the effects of context presence and context type. All experiments are conducted using the Qwen-7B model on 8000 evaluation samples, while keeping the task definition, prompt skeleton, output constraints, decoding strategy, and output normalization rules identical across conditions to ensure that observed differences arise from context representation [11]. We compare generation outcomes using BLEU-4, ROUGE-L, and METEOR [12–14]. Given that commit messages are short and exhibit high surface-form variability, we focus on relative changes across conditions rather than absolute judgments of quality.

Our results show that under the no-context condition (K = 0), performance differences across context types are negligible. In contrast, under the minimal-context condition (K = 1), BLEU-4 increases for all context types relative to zero-shot, while ROUGE-L decreases for all types. METEOR exhibits heterogeneous behavior: it increases under the Message-

only setting, decreases under Diff-only, and changes only modestly under Diff + Message pair. These findings suggest that providing more information in example context does not guarantee consistent improvements across automatic metrics; instead, the type of signal included in the context can structurally differentiate metric responses.

The contributions of this paper are as follows.

- Formulating context representation as an independent factor in retrieval-augmented in-context prompting.
  We systematically compare three context representations (Diff-only, Message-only, and Diff + Message pair) while holding the retrieved example itself constant. This design controls for retrieval quality and example relevance, enabling an isolated analysis of how context representation alone affects commit message generation.
- Quantitatively showing that "type" rather than "amount" of context differentiates metric responses even in a minimal-context regime.
  By contrasting K = 0 with K = 1, we observe that K = 0 yields little separation among context types, whereas K = 1 produces distinct directions and magnitudes of change in BLEU-4, ROUGE-L, and METEOR depending on the context representation. This shows that even a single example can meaningfully shape alignment characteristics when the contextual signal differs in nature.
- Providing empirical evidence for interpreting automatic metrics in the commit message domain.
  We find that Message-only achieves the highest BLEU-4, while Message-only tends to retain relatively higher ROUGE-L than the other context types under K = 1 and yields the largest METEOR increase; Diff-only is associated with a slight METEOR decrease.
- This divergence suggests that automatic metrics may respond more strongly to stylistic and lexical alignment than to semantic adequacy, highlighting the need for careful metric interpretation in commit message generation research.

2. Background

This section positions automatic commit message generation as a subtask of code summarization and reviews prior work that motivates combining large language model (LLM) prompting with retrieval of similar historical examples. In particular, we describe the background of retrieval-augmented prompting approaches proposed in recent literature, where retrieved instances are incorporated into prompts to guide generation. Beyond the general benefit of including retrieved results, we further summarize the prior context relevant to this study's focus: even when the retrieved example is fixed, the representation of that example in the prompt (i.e., what parts are shown and how they are formatted) can influence the model's output behavior and alignment characteristics.

We also discuss the properties and interpretive limitations of automatic evaluation metrics commonly used in commit message generation research. By outlining what these metrics measure and where they may fall short, especially for short texts with high surface-form variability such as commit messages, we clarify the scope of what quantitative comparisons can and cannot conclude. This framing establishes consistent criteria for interpreting the empirical results reported in subsequent sections.

2.1. Commit Message Generation from a Code Summarization Perspective

Git commit messages are concise natural-language summaries that describe the purpose and scope of code changes. Such messages are central to many software maintenance activities, including change history exploration, code review, defect tracking, and regression analysis. Automatic commit message generation can be defined as a natural language

generation task that automates this summarization process, where the goal is to learn the correspondence between code-change information and its textual description.

This task has input characteristics that differ from conventional text summarization. In traditional document summarization, the input is a continuous natural-language document. In commit message generation, by contrast, the input typically consists of code changes in diff format. A diff intermixes heterogeneous token types, such as file paths, function and variable identifiers, code blocks, added and deleted lines, comments, error strings, and other artifacts. Moreover, input length varies substantially with the extent of a change, ranging from a single-line modification to edits spanning multiple files. These properties increase both structural heterogeneity and length variability in the input.

On the output side, commit messages also exhibit characteristics that distinguish them from general-purpose summaries. Even for semantically similar code changes, preferred phrasing can vary across projects or development organizations. Project-level conventions often recur in consistent patterns, including prefixing rules, verb choice, preferred message length, and whether to include module names or issue identifiers. As a result, commit messages for similar changes may appear in diverse surface forms. Accordingly, commit message generation can be framed as a task that jointly requires content fidelity, accurately reflecting the essence of the change, and expression appropriateness, selecting phrasing consistent with project-specific writing conventions.

Prior work has predominantly adopted a sequence transformation perspective, treating the diff as an input sequence and the commit message as an output sequence. Jiang and McMillan proposed a neural machine translation (NMT) approach that generates commit messages from diffs, formalizing the problem as a transformation between code changes and natural-language descriptions [4]. Although many neural models have since been introduced, concerns have persisted regarding the reliability of automatic metric-based comparisons, largely because commit messages are short and exhibit high surface-form variability [1]. Tao et al. empirically showed that automatic metrics such as BLEU and ROUGE may not adequately reflect real quality differences in commit message generation; they reported cases where metric scores did not align with human judgments and emphasized the need for caution when interpreting such metrics [1].

This line of work suggests that, for commit message generation, analyzing directional changes under controlled conditions may be more informative than relying on absolute scores. In practical settings, commit messages are often judged not only by sentence-level linguistic quality but also by consistency with accumulated project conventions. Patterns such as whether prefixes are used, how long messages tend to be, and what information is routinely included are relatively stable at the project level. This implies that model behavior should be examined from multiple perspectives, separating the ability to understand code changes from the ability to produce expressions aligned with project conventions.

Motivated by these observations, we adopt retrieval-augmented prompting that provides similar historical cases as context, and we explicitly compare three ways of representing that context: providing it as a diff, providing it as a commit message, or providing both. This comparison is important because context representation changes the type of information the model can directly reference. Diff-form context exposes structural code-change details, whereas message-form context exposes summarized natural-language expressions and project writing conventions. Such differences may affect what the model prioritizes during generation, and the goal of this study is to analyze these effects systematically.

2.2. Prompting and In-Context Learning

With the advent of Transformer-based large language models (LLMs) pretrained on massive corpora, prompt-based approaches have become a widely adopted paradigm

for performing diverse tasks without updating model parameters [15]. In this paradigm, settings are commonly categorized into zero-shot prompting, where the model is expected to perform a task given only an instruction and the input, and few-shot prompting, where a small number of input–output examples are included in the prompt context. Brown et al. systematically showed that merely providing task examples in the input context can induce models to follow the corresponding input and output formats, even without explicit parameter updates, suggesting that LLMs can infer task behavior from in-context shown examples [5]. Subsequent studies across natural language processing and code generation have repeatedly reported that the number, order, content, and presentation style of in-context examples can significantly influence model outputs. Collectively, these findings motivate viewing prompts not as simple instructions, but as an interface that conveys expected input structure, output formatting, and stylistic patterns. In particular, prompts containing examples are known to steer outputs with respect to sentence length, lexical choice, syntactic structure, and narrative style.

Commit message generation is well suited to in-context learning. Commit messages are typically short, often a single sentence or phrase, and they frequently follow recurring conventions such as verb forms and prefix rules. Moreover, such conventions tend to be relatively stable at the project level. As a result, providing historical commit message instances as examples can encourage the model to produce outputs that more closely match the project's observed patterns. Importantly, however, the training signal conveyed by in-context examples is not determined solely by the presence of examples. Even when the underlying similar case is the same, the type of information included in the prompt can change what cues the model attends to. For example, examples centered on code changes may emphasize structural details and modification scope, whereas examples centered on natural-language messages may more strongly influence expressive properties such as word choice and sentence structure. When both code changes and natural-language descriptions are provided, the prompt simultaneously exposes both information channels.

These observations suggest that the role of in-context learning is shaped not only by the semantic relevance of examples, but also by how examples are represented in the prompt. That is, the information the model can exploit depends on the composition of the contextual channels included in the prompt, not merely on example correctness or similarity. Consequently, even when the retrieved result is held constant, different prompt designs may lead to different output characteristics.

From this perspective, retrieval-augmented prompting is not merely a step that appends similar cases to a prompt. Rather, how a retrieved case is expressed and presented to the model should be treated as a core prompt-design decision. When the representation of the same retrieved result changes, the type and relative weight of cues available to the model may shift, potentially affecting output length, lexical selection, and structural similarity, which are dimensions to which common automatic metrics are sensitive. Motivated by this view, we treat the representation of retrieved context as an independent analysis variable and quantitatively compare its effects in subsequent sections.

To concretely illustrate the output characteristics observed in automatic commit message generation, we additionally present representative examples identified in our evaluation pipeline. When a single-line output constraint is specified in the prompt, the model tends to follow the expected surface format relatively consistently. Nevertheless, even under such formal alignment, we observe cases where the generated message fails to reflect the key change content in the input diff, or where the model produces structural noise (e.g., unintended markup tokens).

Box 1 provides an illustrative example of a well-formed commit message that follows common one-line conventions.

Box 1. Illustrative example of a well-formed single-line commit message used to clarify the target output style.

Good Example

Reference:[DOCS] Document custom routing support for data streams

Generated:chore: Update docs: Remove custom routing support for data streams, clarify index and bulk behavior

Explanation:In this example, the generated commit message preserves the change category implied by the reference message, maintaining it as a documentation-related update. It also directly reflects key concepts associated with the change target. Notably, domain-specific terms such as custom routing and data streams are explicitly retained, indicating that salient cues likely present in the input diff are reproduced at the message level. In addition, the output follows a common one-line commit convention by using a prefix (chore:) and summarizing the change in a single sentence. Taken together, the generated message is aligned with the reference in terms of change category, change target, and formatting, and it includes a minimal yet interpretable set of informational elements sufficient to convey the change intent.

Bad Example

Reference:[COMMUNITY] display github discussions link in issues

Generated:chore: Update ISSUE_TEMPLATE to direct questions to Discuss, Discourse, and Stack Overflow“‘ sql

Explanation:Here, the generated message describes content that is not semantically consistent with the core intent suggested by the reference. While the reference implies a change that displays a GitHub Discussions link on the Issues interface, the generated message instead describes editing ISSUE_TEMPLATE to redirect questions to external forums, which corresponds to a different type of modification. This mismatch weakens the correspondence between the actual changes in the input diff and the produced summary, undermining the message's role as a faithful description of the change. Moreover, the output includes an extraneous code-block marker (“‘ sql) at the end, violating the single-line formatting constraint. This form of structural noise not only harms readability but also indicates that the generation does not reliably satisfy the definition of a commit message as a constrained one-line artifact.

2.3. Retrieval-Augmented Prompting and BM25-Based Retrieval

Retrieval-Augmented Generation (RAG) has been proposed as a framework to address limitations of generation models that rely solely on information encoded in their parameters by retrieving relevant content from an external corpus and incorporating it into the input context [6]. By dynamically providing query-relevant evidence outside the pretrained parameters, RAG aims to enable models to condition their outputs on more concrete and up-to-date context. Retrieval-augmented approaches have been applied across a wide range of tasks, including question answering, summarization, and code generation. In commit message generation, however, the retrieval corpus is not a generic document collection but a set of historical commits, where each instance typically includes both the code change information and the corresponding commit message. The primary goal of retrieval in this setting is to inject past cases similar to the current change into the prompt so that the model can reference the information and stylistic patterns observed in those cases. This mechanism can increase the likelihood that outputs are consistent with project-specific conventions accumulated over time.

In commit message generation, retrieval can serve two conceptually distinct roles. First, by providing examples of similar code changes, retrieval may supply content-oriented cues that support faithful summarization, such as hints about change scope, affected components, or functional intent. Second, by providing examples of similar commit messages, retrieval may supply expression-oriented cues, including style, prefix usage, and lexical preferences. Importantly, even when the retrieved result is held constant, the extent to which these roles are emphasized can depend on how the retrieved case is represented in the prompt.

Both dense and sparse retrieval paradigms can be used for this purpose. Dense retrieval compares queries and documents in an embedding space and can capture semantic similarity, but retrieval outcomes may vary with the choice of embedding model and

indexing design. Sparse retrieval, in contrast, relies on token-level matching and is often simpler to implement with higher reproducibility. BM25 is a widely used sparse retrieval method that scores documents using a weighted function of token overlap and document length normalization [9]. Commit diffs frequently contain repeated surface tokens such as file paths, function and variable names, API calls, and error-message strings. Due to these properties, token-based matching can be effective to a certain degree for identifying similarity between code changes. Moreover, reducing complexity and variability in the retrieval stage is beneficial when the research goal is to isolate the effect of prompt-design choices: if dense retrieval is used, the retrieval result itself may be influenced by multiple design decisions, making it harder to attribute downstream changes to context representation.

For these reasons, this study adopts BM25-based retrieval not to optimize retrieval quality per se, but to analyze how including the same retrieved result in different prompt representations affects model outputs. This choice minimizes variability at the retrieval stage and enables clearer observation of differences induced by context representation within the prompt. At the same time, prompt length constraints are a practical limitation in retrieval-augmented prompting. Including more examples increases the amount of contextual information, but if the input exceeds token limits, parts of the target diff may be truncated or interference among examples may occur. Such effects can prevent the model from adequately attending to the key change information. Therefore, before scaling the number of examples, it is analytically and practically reasonable to first test whether differences in context representation yield meaningful changes even under a minimal-context configuration.

Accordingly, we restrict our analysis to K = 0 and K = 1. The K = 0 condition corresponds to a baseline prompting setting without any retrieved context, whereas K = 1 corresponds to a minimal retrieval-augmented setting in which only a single top-1 similar case is provided. This configuration provides a controlled environment for comparing the effects of retrieval itself and, more importantly, the output characteristics induced by different representations of retrieved context.

2.4. Evaluation Metrics and Interpretive Limitations

Automatic text similarity metrics have been widely used in commit message generation research to compare models and experimental settings at scale. Commonly adopted metrics include BLEU, ROUGE-L, and METEOR, all of which aim to quantify surface-level similarity between a generated message and a reference message. Their appeal lies in providing a consistent, reproducible basis for comparing results across large experiments.

BLEU measures token-level overlap by focusing primarily on n-gram precision between the generated text and the reference, originally proposed for machine translation evaluation and later adopted in commit message generation because it remains computable and convenient even for short outputs. ROUGE-L reflects sequential similarity based on the longest common subsequence between the generated text and the reference, and it is widely used in summarization evaluation. METEOR was introduced to mitigate some brittleness of pure n-gram matching by incorporating stemming, heuristic matching rules, and a combination of precision and recall.

Despite their practicality, interpreting these automatic metrics in the commit message domain entails structural limitations. Commit messages are typically extremely short, and multiple semantically valid phrasings may exist for the same code change. Consequently, surface overlap with a single reference message may fail to adequately represent whether the generated message is truly appropriate as a description of the underlying change. In addition, commit messages frequently include recurring stylistic elements such as prefix choices, verb forms, preferred length, and conventional phrases. These stylistic factors can

have a disproportionate impact on metric scores. For instance, even when two messages are semantically similar, differences in phrasing may yield large changes in n-gram-based scores. When the reference message strongly reflects project-specific conventions, conditions that generate expressions closer to those conventions may systematically receive higher scores under surface-similarity metrics.

Tao et al. analyzed cases in which automatic metric scores diverge from perceived quality in commit message generation and empirically argued that careful interpretation is required [1]. They noted that while such metrics can be useful for relative comparison under controlled settings, they are limited as definitive criteria for judging the quality of individual outputs.

Given this background, we employ BLEU, ROUGE-L, and METEOR as primary evaluation metrics, but we do not interpret their absolute values as direct proxies for real-world commit message quality. Instead, we focus on how scores change in direction and magnitude when prompt design factors, particularly context representation, are varied under identical evaluation conditions. In retrieval-augmented prompting, the representation of contextual information may influence expressive alignment more strongly than content fidelity, and differences in metric sensitivities may therefore lead to heterogeneous response patterns across metrics. Our experiments quantitatively compare how each metric responds when the representation of the same retrieved context changes, providing foundational evidence for interpreting what such score shifts may indicate about underlying changes in output characteristics.

3. Methodology

This section describes the similarity-based retrieval-augmented generation framework used to produce a single-line commit message from a given code change (diff). The goal of this study is to comparatively analyze how generation outcomes change at inference time, without updating model parameters, when similar historical cases retrieved from past commit histories are incorporated as prompt context. In particular, under a retrieval-based prompting setup, we aim to disentangle the effect of context representation from the mere fact that an example is provided, and to isolate how different ways of constructing example context influence the generated outputs.

To this end, we design the methodology around three separable components: dataset construction and preprocessing for commits consisting of code changes (diffs) and commit messages, retrieval procedures for identifying past commits similar to the input diff, and strategies for converting retrieved cases into prompt context. These components are defined independently, and experimental controls are applied to ensure that factors other than context construction do not affect the comparison across conditions.

To ensure fairness across experimental conditions, we use the same large language model and the same prompt skeleton in all settings. We constrain outputs to a single-line commit message, and we keep the decoding strategy and post-processing rules identical across all conditions. Therefore, differences across conditions are limited to whether retrieved context is included and when context is included, which type of information is exposed in the prompt. This design ensures that any observed performance differences arise from differences in context construction rather than model architecture, decoding procedures, or output normalization.

3.1. Framework Overview

We adopt a retrieve-then-generate pipeline. Given a target commit, its code change (diff) is used as the input, and the same diff is also used as a query to retrieve similar cases from a reference database of past commits. The reference database is constructed at the

commit level, where each entry consists of one diff and its corresponding commit message. The retrieval stage is separated from the generation stage, and retrieved results are used solely for constructing example context in the prompt.

Retrieval is performed over a pre-built commit-level index, returning top-ranked cases based on similarity between the input diff and historical diffs. In this study, we use only the single most similar case (top-1) as the candidate example. The retrieval pipeline and returned case are kept identical across all experimental conditions, ensuring that differences in retrieval quality or retrieval targets do not confound comparisons.

Figure 1 summarizes the overall pipeline and experimental variables. We first split the dataset into an evaluation query set (Eval, 8000 samples) and a retrieval reference database (Retrieval DB). For each evaluation sample, we run BM25-based retrieval using the target diff as the query. Depending on the experimental setting (K and context type), the retrieved result is then converted into prompt context to form the model input, after which generation, output normalization, and automatic evaluation are performed.

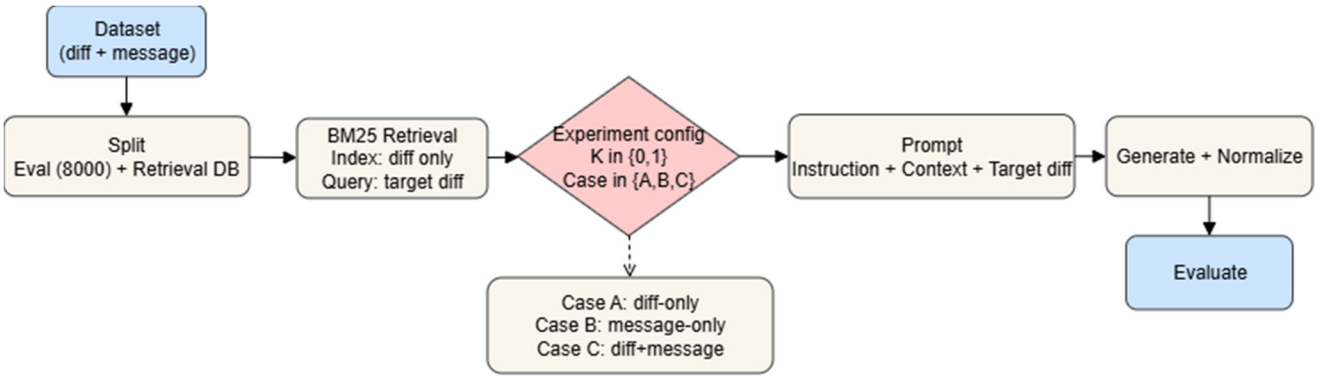


Figure 1. Retrieve-then-generate-based commit message generation pipeline and experimental conditions.

For BM25, we index only diffs (commit messages are not used in retrieval), and the query is fixed to the target commit's diff. The experimental variables are the presence of retrieved context (K = 0 vs. K = 1) and, under K = 1, the representation of the retrieved context (A: Diff-only, B: Message-only, C: Diff + Message). All other factors, including the prompt skeleton, decoding configuration, and post-processing rules, are held constant across conditions. For fair comparison, we also execute the same retrieval pipeline under K = 0, but omit the example-context block from the prompt and do not include the returned case.

The target commit's diff is used as a query to perform BM25 retrieval over the Retrieval DB, and the top-1 similar case is optionally incorporated into the prompt as example context depending on K (K = 0: excluded; K = 1: included). The prompt consists of three blocks: an instruction block, an example context block, and a target input block; across conditions, only the example-context block varies. Under K = 1, the retrieved case is converted into one of three context representations and injected accordingly (A: Diff-only, B: Message-only, C: Diff + Message). The generated output is then passed through the same normalization and post-processing pipeline and used for automatic evaluation.

3.2. Dataset Construction and Preprocessing

We construct our dataset based on MCMD (Multi-programming-language Commit Message Dataset) (version v2, available at https://zenodo.org/records/5025758, accessed on 20 December 2025) [16] and release the processed evaluation split as a public artifact [17]. Each sample is a commit-level pair consisting of a diff as input and its corresponding

commit message as the reference output. Each sample corresponds to a single commit, where the input is the commit's code change represented as a diff text, and the reference output is the commit message. Diffs contain heterogeneous token types such as file paths, identifiers, code lines, and addition/deletion markers, and their lengths vary widely depending on change size. If used as-is, these properties can introduce unnecessary noise and length-induced variance that may affect evaluation outcomes. Therefore, to ensure fair comparisons and stable evaluation, we apply filtering and normalization procedures starting from the collection stage.

To ensure a reasonable level of activity and project maturity, we apply a star-based threshold when selecting repositories. For each selected repository, we extract diffs and commit messages at the commit level and include only English-written messages to maintain linguistic consistency. We exclude commits whose nature differs from intent summarization, such as merge commits and automatically generated commits. In addition, personally identifiable information (PII), including author names and email addresses, is not collected or is removed during data acquisition.

Because automatic metrics can be highly sensitive to small lexical differences in short texts, a large number of low-information messages can cause score fluctuations to be dominated by noise rather than by experimental factors. To mitigate this issue, we remove excessively short messages, single-word boilerplate expressions, and repetitive auto-generated patterns using rule-based filtering. This step is not intended as dataset "cleaning" for model training; rather, it is a quality-control procedure designed to reduce variance in automatic evaluation for controlled comparisons.

Input diffs also require normalization due to their substantial length dispersion. Extremely long diffs can trigger truncation under input length constraints, and the location and extent of truncation may distort cross-condition comparisons. Accordingly, we either exclude samples with excessively large diffs or apply predefined truncation rules to moderate the input length distribution. When truncation is necessary, we prioritize preserving the beginning of the diff so that structural cues such as file headers, file paths, and early change regions are retained as much as possible.

Data leakage is another critical concern in retrieval-based experiments. If identical or near-duplicate changes appear in the retrieval database, retrieval may effectively expose the reference answer, inflating performance estimates. To prevent this, we check duplication based on diffs rather than message text. Specifically, we normalize diffs, compute hash values, and identify identical changes to ensure that duplicate diffs do not appear across splits. We also separate the evaluation set from the retrieval reference database to prevent self-matches, and we enforce that the same identifiers are not returned during retrieval.

All experimental results reported in this study are computed on a total of 8000 evaluation samples, which constitute the query set used for generation and automatic evaluation. The Retrieval DB is composed of commits disjoint from this evaluation set. We consistently report and analyze results with respect to the 8000 evaluation samples.

3.3. Similarity-Based Retrieval Mechanism

To identify historical commits similar to an input diff, we employ BM25-based sparse retrieval. BM25 is a probabilistic ranking model that scores documents based on token overlap with a query, providing a reproducible baseline without requiring pretrained embeddings or additional training. Commit diffs often contain recurring surface tokens such as file paths, class and function names, variable identifiers, and error strings, making them well suited to token-matching retrieval. By keeping the retrieval mechanism simple, we aim to ensure that any downstream differences are attributable to context construction rather than to the choice or complexity of the retrieval model.

We build a commit-level retrieval index. For each indexed commit, the document text consists of the diff only; commit messages are not used during retrieval. For each evaluation sample, the query is the target commit's diff, and BM25 scores are computed based on token overlap between the query and indexed diffs. Retrieved candidates are ranked by score, and in this study we select only the top-1 most similar case as the example candidate.

We define the number of retrieved examples included in the prompt as K and restrict our analysis to K = 0 and K = 1. The K = 0 condition corresponds to a zero-shot setting in which no retrieved context is included in the prompt, whereas K = 1 corresponds to a minimal-context setting in which only the single top-1 similar case is provided. While increasing K can increase contextual information, it also expands prompt length and may introduce interference among examples. To more clearly isolate the effect of context representation, we therefore conduct comparisons under this minimal setting.

The retrieval pipeline and the returned similar case are held constant across all experimental conditions. In other words, retrieval results themselves do not change across context-using conditions; the only difference is how the same retrieved case is exposed in the prompt. Concretely, we construct three context representations from the identical top-1 retrieved case: providing only the retrieved diff (Diff-only), providing only the retrieved commit message (Message-only), and providing both the retrieved diff and message (Diff + Message). This design controls for retrieval variability and enables isolating the effect of context representation on generation outcomes.

3.4. Prompt Design for In-Context Learning

Our prompt design is guided by a single principle: across conditions, only whether retrieved context is provided and how that context is represented should be able to induce differences in the outputs. To satisfy this requirement, we use an identical prompt template skeleton for all experimental conditions and treat the inclusion of retrieved context and its representation format as the only condition variables. The instruction content, output-format constraints, block ordering, decoding configuration, and post-processing rules are held constant across all settings.

The prompt consists of three blocks: an instruction block, an example context block, and a target input block. The instruction block specifies the task and output constraints. The example-context block is included only when K = 1. The target input block contains the current commit's diff and is always placed at the bottom of the prompt. Even when the example-context block is present, the overall layout and field delimiters remain unchanged; the only variation is the type of information populated in the example block.

We consider three example-context representations. Case A provides only the retrieved commit's diff and omits its message. Case B provides only the retrieved commit message and omits the diff. Case C provides both the retrieved diff and the retrieved message for the same retrieved case. Because all three cases use the identical retrieval result, differences arise solely from the composition of information channels exposed in the prompt.

Below we present the prompt template used in our study (Box 2). The template skeleton is identical across conditions, and only the example-context block differs by representation. The template skeleton is identical across conditions, and only the presence of the example-context block and the filled fields vary depending on K and the case type.

3.5. Commit Message Generation

The generation stage in our framework takes a preprocessed code change (diff) as input and instructs a large language model to produce a single-line commit message. In this stage, both input and output formats are strictly fixed so that any differences observed

in subsequent experiments can be attributed to context construction, rather than incidental variation in model settings.

Box 2. Prompt template used for controlled in-context commit message generation (only the example-context block changes across conditions).

```
[System Instruction]
You are a coding assistant. Write a concise commit message for the diff. Rules: Start with a verb. No prefixes. Output ONLY the message.

[Retrieved Context (Few-shot Examples)]
Example 1:
Diff: {Retrieved_Diff_1}
Message: {Retrieved_Message_1}

[Target Task]
Target Diff:{Target_Diff_Content}
Write the commit message.
```

We normalize the input diff to encourage the model to focus on the change content itself. Metadata that may appear in raw diffs, such as commit hashes, author information, timestamps, and file permission details, is removed because it is weakly related to change summarization while increasing input length. The input is instead constructed around the modified file paths and the actual changed code lines. Addition and deletion markers ($+/-$) are preserved so that the directionality of changes remains explicit. Because diff length dispersion can cause the input to exceed length constraints, we apply the same maximum length limit across all conditions. When truncation is required, we truncate from the tail so that the beginning of the diff is preferentially preserved. The head of a diff often contains file headers, modified paths, and the early portion of the main change, which helps reduce the loss of salient cues under a fixed budget.

Given the normalized diff, the model generates a single-line commit message. Generation is performed with an identical decoding configuration across all conditions to ensure that the same input yields the same output behavior under the same settings, and outputs are constrained to a single line. If the model produces line breaks, we keep only the first line. We further remove artifacts such as labels (e.g., "Commit message:"), surrounding quotation marks, and unnecessary whitespace to standardize the final message format. These normalization rules are not designed to favor any particular condition; rather, they are applied uniformly to enforce a consistent artifact definition and to ensure comparability across conditions.

Below, we provide an illustrative example showing how an input diff containing structural changes is summarized into a single-line message (Box 3).

Box 3. Example input diff excerpt and the corresponding generated commit message.

Example. Input diff and generated commit message

Input (Target Diff, excerpt)

diff --git a/.../SendMessage.java b/.../SendMessage.java

- if (type.equals("Play")) { ... }
- if (type == SessionMessageType.PLAY) { ... }
- if (type.equals("Stop")) { ... }
- if (type == SessionMessageType.STOP) { ... }

Output (Generated Commit Message)

Replace string message types with SessionMessageType enum

4. Experiments

4.1. Experimental Setup

We follow the retrieve-then-generate pipeline defined in Section 3. The input is the target commit's code change (diff), which is used as a query to perform BM25-based sparse retrieval over a reference database. The retrieval index is constructed at the commit level, and only diffs are used as indexing text. Commit messages are not used during retrieval and are used only in the example-context construction stage. The retrieval stage and the returned top-ranked similar case are kept identical across all conditions to prevent retrieval quality or retrieval target differences from confounding the comparisons.

Experiments are conducted on the commit dataset constructed in Section 3.2. We split the full dataset into an evaluation query set (Eval) and a retrieval reference database (Retrieval DB). To mitigate potential language imbalance and language-driven bias in evaluation, we construct Eval using stratified random sampling by programming language. Specifically, we first assign each commit to a primary **programming language** using repository metadata and/or file extensions observed in diff headers. We then uniformly sample 2000 commits from each of the four most frequent programming languages in our corpus (Python, Java, C++, and JavaScript), yielding 8000 evaluation samples in total. Each evaluation sample includes a diff as input and the corresponding ground-truth commit message as the reference.

To characterize dataset diversity, we report descriptive statistics of change scope and message properties, including diff length proxies (added/deleted lines), number of modified files, and reference message length. These statistics clarify the range of change sizes covered in evaluation and help interpret how representation effects may vary with change scope.

The Retrieval DB consists of commits disjoint from the evaluation set, and we explicitly exclude self-matches so that the evaluation commit itself is not returned by retrieval. The Retrieval DB is built from the remaining filtered commits after removing Eval; we do not enforce a balanced language distribution in the Retrieval DB, since it is used only as the candidate pool for retrieval. Importantly, the retrieval stage and the returned top-ranked similar case are kept identical across all experimental conditions, ensuring that comparisons are not confounded by retrieval variability.

Evaluation is performed on 8000 samples (Eval), where each sample includes a diff as input and the corresponding ground-truth commit message as the reference. The Retrieval DB consists of commits disjoint from the evaluation set, and we explicitly exclude self-matches so that the evaluation commit itself is not returned by retrieval.

The controlled experimental variables are whether example context is provided (K) and how the example context is constructed. The K = 0 condition corresponds to a zero-shot generation setting with no example context. The K = 1 condition provides a minimal retrieval-augmented setting, in which only the single top-1 similar case returned by BM25 is included as an example. Under K = 1, we consider three context representations: Case A (Diff-only) provides only the retrieved diff text; Case B (Message-only) provides only the retrieved commit message; and Case C (Diff + Message pair) provides both the retrieved diff and its message. Because all three cases use the same retrieved top-1 example, differences across conditions arise solely from the type of information exposed in the prompt.

For generation, we use a single open-source LLM and report all results measured with Qwen-7B. The instruction text, prompt skeleton, output-format constraints, decoding configuration, and output-cleaning rules are identical across all conditions. Decoding is performed deterministically (without sampling) for reproducibility, and outputs are constrained to a single-line commit message. The input length is limited to 2048 tokens; when the input exceeds this budget, we truncate from the tail of the diff so that headers

and early change lines are preferentially preserved. If the generated output contains line breaks, only the first line is retained. If the output includes labels (e.g., "Commit message:") or surrounding quotation marks, we remove them and normalize whitespace. These post-processing steps are not intended to advantage any specific condition, but rather to enforce the common artifact definition of a "single-line commit message" across all settings.

We use Qwen-7B to enable reproducible, controlled experimentation under a fixed inference budget with an openly available model. Our goal is not to claim model-universal effects, but to isolate representation-driven differences under identical decoding and prompt structure; different LLM families may exhibit different sensitivities to the same representation change.

Hardware and operating system. Experiments were conducted on a workstation equipped with an NVIDIA GeForce RTX 4090 GPU (NVIDIA, Santa Clara, CA, USA), running Ubuntu 24.04.3 LTS (Canonical, London, UK).

Software. All experiments were implemented in Python (v3.10.18) using PyTorch (v2.7.1+cu126) and Hugging Face Transformers (v4.54.1). BM25 retrieval was implemented with rank-bm25 (v0.2.2). Data processing used NumPy (v2.2.6) and pandas (v2.3.3).

Table 1 summarizes the fixed settings used across retrieval, input processing, generation, and post-processing to ensure fair comparisons among experimental conditions. All conditions employ the same BM25-based sparse retrieval, and both the indexing text and the query are restricted to the code change (diff). Commit messages are not used during retrieval and are utilized only when transforming the retrieved case into prompt context.

Table 1. Fixed settings used across all experimental conditions for fair comparison (retrieval, input processing, generation, and post-processing).

Category	Parameter or Item	Value or Description
Retrieval BM25	Library Implementation	rank_bm25 (OkapiBM25), k1 = 1.5, b = 0.75
Retrieval BM25	Tokenizer and Preprocessing	Whitespace tokenization with lowercasing
Retrieval BM25	Indexed Text for BM25	Diff text only message used only for prompt context construction
Retrieval BM25	Query Scope	Input diff code changes in the query commit
Input Processing	Context Window Limit	2048 tokens total (prompt + target diff + optional retrieved context)
Input Processing	Truncation Strategy	Right truncation preserving the header and start of the diff
Generation	Decoding Strategy	Greedy decoding temperature 0.0
Generation	Max New Tokens	60 Qwen 7B
Generation	Batch Size	1 (single-sample inference for reproducibility)

In input processing, the total input length including the prompt and the target diff is capped at 2048 tokens. When this limit is exceeded, truncation is applied from the tail of the diff so that file headers and early change regions are preferentially preserved. For generation, we use the Qwen-7B model with deterministic decoding without sampling (greedy decoding, temperature 0.0). The output is limited to a maximum of 60 tokens, and the batch size is fixed to 1.

In post-processing, if the generated output contains line breaks, only the first line is retained. Unintended formatting artifacts, such as message labels or surrounding quotation marks, are removed. These settings are not intended to advantage any particular condition; rather, they serve as control variables to maintain a consistent input–output definition and a shared generation environment across all experimental conditions.

4.2. Dataset and Evaluation Metrics

Dataset composition for evaluation. Because commit corpora are often dominated by a small number of languages, evaluating on a naively sampled subset can over-represent high-volume languages and under-represent others, making aggregate results difficult to interpret. To reduce this risk, we evaluate on a language-balanced query set (Eval) created via stratified random sampling. We select the four most frequent languages in our corpus (Python, Java, C++, and JavaScript) and sample 2000 commits per language (total 8000). This design controls for language imbalance when comparing experimental conditions, so that observed metric shifts are less likely to be artifacts of a skewed language distribution. All reported results are computed on this fixed Eval set, and the same Eval instances are used across all conditions. This study focuses on comparing how generation outcomes change as a function of whether context is provided (K) and how the context is represented (Cases A/B/C). Accordingly, we do not treat absolute metric values as definitive judgments of commit message quality. Instead, we interpret results primarily through the direction and magnitude of score changes induced by controlled condition shifts.

We evaluate generated messages using three automatic metrics: BLEU-4 [12], ROUGE-L [13], and METEOR [14]. All three quantify the degree of string-level alignment between a generated message and its reference message. In general, higher scores indicate that the generated output is more closely aligned with the reference in terms of word choice and surface structure, whereas lower scores indicate greater divergence at the expression level.

- BLEU-4 [12]: BLEU computes a score based on how many word n-grams overlap between the generated message and the reference, where an n-gram is a contiguous sequence of words. BLEU-4 considers overlaps up to 4-g. We report BLEU-4 on a 0–100 scale, where higher values indicate that the generated message contains more word sequences identical to those in the reference. An increase in BLEU-4 suggests that the output is moving toward the reference's phrasing patterns, while a decrease suggests a shift toward different word combinations. Because commit messages are extremely short, BLEU-4 can fluctuate substantially even with a single-word difference. Therefore, we use BLEU-4 primarily as a signal of whether a condition change nudges outputs toward the reference's surface phrasing.
- ROUGE-L [13]: ROUGE-L is computed based on the length of the longest common subsequence (LCS) between the generated message and the reference while preserving token order. We report ROUGE-L on a 0–100 scale, where higher values indicate that a longer ordered sequence of words is shared with the reference. An increase in ROUGE-L implies that the output's word ordering and sentence frame more closely match the reference, whereas a decrease implies weaker sequential alignment, even if some words overlap. Given the short length of commit messages, changes in word order can strongly affect ROUGE-L; thus, we use it as a signal of whether condition changes pull the output's structural pattern toward the reference.
- METEOR [14]: METEOR also measures word-level alignment between the generated message and the reference, but it extends beyond exact token matches by partially accounting for morphological variants and heuristic matching rules. We report METEOR on a 0–100 scale, where higher values indicate stronger lexical alignment with the reference, including cases where matched words differ slightly in form. An increase in METEOR suggests closer word choice alignment or more successful matches under variant forms, whereas a decrease suggests divergence in lexical choice and reduced matching on project-specific terms or abbreviations. In the commit message domain, stylistic factors such as prefix forms, short idiomatic phrases, and abbreviation usage can influence METEOR; therefore, we interpret METEOR primarily through relative changes across conditions rather than absolute values.

- Semantic similarity (embedding-based): To complement lexical overlap metrics, we additionally compute an embedding-based semantic similarity between the generated message and the reference. This measure is less sensitive to surface-form paraphrasing and provides a coarse check on semantic adequacy when lexical metrics change due to stylistic alignment.

Because these metrics are sensitive to different aspects of surface similarity, their response directions may not be consistent under the same condition change. Therefore, rather than concluding quality improvements from a rise in a single metric, we summarize results by examining the patterns of how BLEU-4, ROUGE-L, and METEOR move in direction and magnitude as we vary context inclusion and context representation. All metrics in this paper are reported on a 0–100 scale. Accordingly, all paired deltas (K = 1 − K = 0) reported in the same 0–100 unit (i.e., percentage-point differences).

4.3. Research Questions

In a retrieve-then-generate setting where the top-1 similar case obtained via BM25-based sparse retrieval is injected as prompt example context, this study investigates how automatic evaluation metrics vary as a function of whether an example is provided (K = 0 vs. K = 1) and how the example context is represented (Diff-only, Message-only, Diff + Message pair). We formulate the following research questions:

RQ1. *Between K = 0 (no example context) and K = 1 (one top-1 similar example), what are the direction and magnitude of changes in BLEU-4, ROUGE-L, and METEOR?*

RQ2. *Under the K = 1 condition, when the example context is represented as Diff-only, Message-only, or Diff + Message pair, how do the magnitudes of change in BLEU-4, ROUGE-L, and METEOR differ across context types?*

RQ3. *Do the same condition changes yield consistent responses across the three metrics, or do the metrics exhibit different directions and sensitivities depending on what aspect of surface similarity they capture?*

4.4. Experimental Results

This section compares changes in automatic evaluation metrics as a function of context inclusion (K = 0 vs. K = 1) and context representation (Cases A/B/C) on 8000 evaluation samples (Eval). For all K = 1 settings, we use the same top-1 similar case returned by BM25-based retrieval and vary only how that case is exposed in the prompt. Case A provides only the retrieved code-change text (Diff-only), Case B provides only the retrieved commit message (Message-only), and Case C provides both the retrieved diff and message (Diff + Message pair).

Table 2 reports the automatic evaluation results (BLEU-4, ROUGE-L, and METEOR) measured with Qwen-7B across K and context types. Because K = 0 corresponds to a zero-shot setting in which the entire example-context block is removed, the prompt becomes identical across nominal context types. We therefore report a single shared K = 0 baseline and use it to quantify changes under K = 1.

Table 2. Automatic evaluation results on 8000 evaluation samples using Qwen-7B.

K	Case A			Case B			Case C		
	BLEU 4	**ROUGE L**	**METEOR**	**BLEU 4**	**ROUGE L**	**METEOR**	**BLEU 4**	**ROUGE L**	**METEOR**
0	1.14	7.48	4.91	1.14	7.48	4.91	1.14	7.47	4.88
1	2.49	6.59	4.90	2.81	7.03	5.18	2.79	6.97	5.05

Under K = 1, however, metric responses diverge by context representation even with the same one-shot example (Table 2). BLEU-4 increases relative to K = 0 for all three cases, reaching 2.49 for Diff-only, 2.81 for Message-only, and 2.79 for Diff + Message pair. This suggests that providing an example context tends to shift outputs toward the reference's n-gram patterns. In contrast, ROUGE-L decreases for all representations compared to K = 0, yielding 6.59 (Diff-only), 7.03 (Message-only), and 6.97 (Diff + Message pair). METEOR shows minimal change for Diff-only (4.90), but increases for Message-only (5.18) and Diff + Message pair (5.05). Notably, within the K = 1 comparisons, Message-only maintains relatively higher ROUGE-L and METEOR than the other representations, indicating that presenting example context in message form can yield more favorable alignment under certain metrics.

Table 2 also highlights that metric response directions are not necessarily consistent. BLEU-4 is driven by n-gram overlap and thus tends to increase when outputs move toward specific phrasing patterns induced by context. ROUGE-L is sensitive to ordered subsequence similarity, and for short texts such as commit messages, even small word-order or framing changes can substantially affect the score. METEOR partially accounts for morphological variants, but matching can remain limited when project-specific terms or abbreviations dominate. Consequently, the automatic scores reported here should not be interpreted as direct proxies for real-world usefulness; rather, they serve as relative indicators of how outputs become more or less aligned with the reference messages' surface forms and structures under controlled changes in context inclusion and representation.

4.5. Effect Size and Consistency Across Samples (RQ3)

Table 2 reports aggregate averages, but averages can hide whether metric shifts are widespread or driven by a small subset of commits. To assess sample-level consistency, we compute paired deltas for each evaluation instance under the same representation (K = 1 minus K = 0): ΔBLEU-4, ΔROUGE-L, and ΔMETEOR. Table 3 summarizes the resulting delta distributions using the median, mean, and interquartile range (IQR), together with the proportion of samples with positive deltas.

Table 3. Paired delta statistics (K = 1 − K = 0), reported in the same 0–100 scale as the main metrics (percentage-point differences).

Exp	Scope	Median	Metric	*n*	Mean	IQR	Pos (%)
A	ALL	0.00	ΔBLEU-4	8000	0.0020	0.0049	38.0
A	ALL	0.00	ΔROUGE-L	8000	0.0022	0.0034	26.0
A	ALL	0.00	ΔMETEOR	8000	−0.0013	0.0017	30.3
B	ALL	0.00	ΔBLEU-4	8000	0.0043	0.0124	45.8
B	ALL	0.00	ΔROUGE-L	8000	0.0060	0.0157	31.2
B	ALL	0.00	ΔMETEOR	8000	0.0029	0.0184	36.8
C	ALL	0.00	ΔBLEU-4	8000	0.0039	0.0064	36.4
C	ALL	0.00	ΔROUGE-L	8000	0.0048	0.0000	23.6
C	ALL	0.00	ΔMETEOR	8000	0.0019	0.0012	27.4

Across all representations, the median deltas are 0.0000, indicating that improvements are not uniform at the instance level. Instead, Table 3 shows small distributional shifts with substantial heterogeneity: in Exp A (Diff-only), mean deltas are slightly positive for BLEU-4 and ROUGE-L (0.0020 and 0.0022) while METEOR is slightly negative on average (−0.0013), and positive-delta rates range from 26.0% to 38.0. In Exp B (Message-only), the mean deltas are larger (BLEU-4: 0.0043; ROUGE-L: 0.0060; METEOR: 0.0029) with higher positive-delta rates (31.2–45.8%), suggesting a stronger tendency toward surface-form alignment with the reference. In Exp C (Diff + Message), ROUGE-L exhibits IQR = 0.0000, implying a point

mass at zero for many instances even though the mean remains positive, which further highlights that metric responses are concentrated near zero and highly heterogeneous across samples.

Table 3 shows that the one-shot effect (K = 1 − K = 0) is not a broad shift but a sparse, representation-dependent perturbation. The key signal is the joint pattern (median = 0, small IQR, Pos% < 50%) across all metrics and all representations: the median of 0.00 means the "typical" instance experiences no measurable change, and the sub-50% Pos rates (A: 26.0–38.0; B: 31.2–45.8; C: 23.6–36.4) imply that even where the mean is positive, improvement is not majority behavior but driven by a minority of instances with non-zero gains. Representation differences are also explicit: Message-only (Exp B) has the largest means and the largest IQRs for all metrics (ΔBLEU-4 mean 0.0043, IQR 0.0124; ΔROUGE-L mean 0.0060, IQR 0.0157; ΔMETEOR mean 0.0029, IQR 0.0184), indicating that message-form context more frequently and more strongly reshapes surface phrasing. In contrast, Diff + Message (Exp C) yields ROUGE-L IQR = 0.0000 with Pos = 23.6%, which implies a heavy point mass at exactly zero (ordered subsequence similarity unchanged for most commits) and a small tail responsible for the positive mean; thus, "more information" does not translate into broadly stronger alignment under ROUGE-L. Finally, Diff-only (Exp A) is the only setting with a negative mean ΔMETEOR (−0.0013), suggesting that when code-centric context changes outputs, it tends to alter lexical choices in ways that reduce METEOR-style matches. Overall, Table 3 supports the interpretation that K = 1 primarily affects a subset of commits and that the type of contextual signal determines whether the changes manifest as broader lexical reshaping (Exp B) or near-zero movement with occasional outliers (Exp C), rather than delivering consistent per-instance gains.

4.6. Qualitative Examples and Recurring Error Patterns

To explain why the same condition change can increase one metric while decreasing another, we inspected representative outputs and observed three recurring patterns. First, brevity shifts occur when K = 1 encourages shorter, conventional commit phrasing, which can increase precision-oriented overlap (e.g., BLEU-4) while reducing sequence-level similarity (ROUGE-L) when the reference uses a different framing. Second, stylistic imitation appears when Message-only context induces common verb choices and conventional phrases (e.g., "Fix", "Update", "Refactor"), improving lexical alignment in some cases without necessarily improving intent capture. Third, structural noise includes occasional formatting artifacts or truncated fragments, which can disproportionately harm ROUGE-L despite partial token overlap. These cases support interpreting metric movements primarily as surface-form alignment effects rather than unambiguous semantic improvements. As shown in Table 4, BLEU-4 can increase while ROUGE-L and/or METEOR decreases under the same context change, even when all values are compared in the same 0–100 unit. This indicates that gains in precision-oriented overlap do not necessarily translate into higher sequence-level similarity.

Table 4 provides concentrated instance-level evidence that the one-shot setting (K = 1) induces a systematic pattern of metric disagreement, not isolated anomalies. Across the 11 examples shown (Exp A: 4, Exp B: 4, Exp C: 3), ΔROUGE-L is negative in all 11/11 cases and ΔMETEOR is also negative in all 11/11 cases, whereas ΔBLEU-4 is positive in 10/11 cases (and exactly 0 in the remaining case). This asymmetry indicates that, in these representative samples, K = 1 primarily increases local n-gram precision overlap with the reference (BLEU-4) while consistently reducing sequence-level alignment (ROUGE-L) and token-level match robustness under METEOR's matching heuristics. The magnitudes reinforce that this is not a trivial effect: ΔBLEU-4 ranges from near-zero gains (e.g., +0.0024, +0.0084) to large jumps (+0.3948, +0.4506), while ΔROUGE-L spans from small decreases

(e.g., −0.0238) to substantial drops (−0.2727). ΔMETEOR likewise exhibits both mild and severe declines (e.g., −0.0610 up to approximately −0.4167/−0.4135), showing that BLEU-4 improvements frequently co-occur with non-negligible losses in other overlap views.

Table 4. Qualitative examples illustrating metric disagreement across BLEU-4, ROUGE-L, and METEOR (Δ = K = 1 − K = 0, percentage-point differences on the 0–100 scale).

Exp	Sha (Short)	Gold (Reference)	K = 0 Output	K = 1 Output	ΔBLEU-4	ΔROUGE-L	ΔMETEOR
A	c64446d51b	test: xfail Driver/ sanitize_address_use _odr_indicator...	Implement ODR indicator for address sanitization...	Add REQUIRES comments for rdar72514092 and asan...	+0.3948	−0.1111	−0.0610
A	a81254dfff	fix port sort	Update _addSubItem to include item.port...	Update _addSubItem to include item.port...	+0.0024	−0.0952	−0.4167
A	54ffbbe7db	remove asserts from compute_float_64()	Update numberparsing.h: remove asserts for power...	Update numberparsing.h: simplify assert checks...	+0.0000	−0.2000	−0.3001
A	99442b6e04	Remember last selected media type for downloads	Update DownloadDialog: use prefs for default media...	Update DownloadDialog to use preferences for default...	+0.0084	−0.1135	−0.2803
B	e5c0560e46	compat: update forward compatibility horizon...	Update forward compatibility horizon to 22 December 2020	Update forward compatibility horizon to 22 December 2020	+0.4506	−0.0451	−0.0703
B	7d4486ff28	[compiler] Use Handles for source positions in concurrent...	Update BytecodeGraphBuilder/ heap-refs to use Handles...	Update BytecodeGraphBuilder and HeapRefs to use Handles...	+0.0163	−0.1850	−0.2989
B	7a60d48a99	Add test case for ShardingStrategyFactory()	Add ShardingStrategyFactory test cases for standard...	Add MasterSlaveData-SourceRouter to ShardingStrategy...	+0.0383	−0.0238	−0.4135
B	d6076096c7	Update pom.xml	Update hibernate-types artifact name to hibernate-...	Update hibernate-libraries artifact name and add...	+0.0844	−0.1600	−0.1754
C	e5c0560e46	compat: update forward compatibility horizon...	Update forward compatibility horizon to 22 December 2020	Update forward compatibility horizon to 22 December 2020	+0.4506	−0.0451	−0.0703
C	7a60d48a99	Add test case for ShardingStrategyFactory()	Add ShardingStrategyFactory test cases for standard...	Add MasterSlaveData-SourceRouter to ShardingStrategy...	+0.0401	−0.0238	−0.4135
C	62f32eb715	Use relative location for invokes for runtime compilation...	Update NonSnippetLowerings.java: use LocationIden...	Update NonSnippetLowerings to handle FloatConvert...	+0.0040	−0.2727	−0.1364

The paired outputs further suggest that the disagreement is driven by recurring edit mechanisms that alter surface form in ways that different metrics weight differently. First, format/normalization substitutions can raise BLEU-4 without preserving ordered overlap: for sha e5c0560e46 (Exp B/C), the K = 1 output rewrites a natural-language date into an ISO-like form ("22 December 2020" → "22 December 2020"), increasing short n-gram matches around the date content while disrupting the reference's phrasing structure, yielding negative ΔROUGE-L and ΔMETEOR. Second, framing and word-order reshaping (e.g., changing the head verb phrase or relocating modifiers) is common; because commit messages are short, small reorderings sharply affect the longest common subsequence, producing ROUGE-L drops even when some content words remain shared. Third, Table 4 includes clear cases of entity/target drift under K = 1: for sha 7a60d48a99 (Exp B/C), the K = 1 output shifts focus toward a different component ("MasterSlaveDataSourceRouter") than the gold's primary target ("ShardingStrategyFactory"), yielding a small positive ΔBLEU-4 but large negative ΔMETEOR and ΔROUGE-L, consistent with a plausible-but-mismatched specificity error. Overall, these examples support interpreting the K = 1 effect primarily as a representation-driven shift in surface alignment behavior (local lexical

convergence plus reframing), rather than a monotonic improvement in change-intent adequacy, and they concretely explain why BLEU-4 gains can coincide with systematic ROUGE-L/METEOR losses in the minimal-context regime.

5. Discussion

5.1. Experimental Results Analysis

This section quantitatively analyzes how context type and minimal context injection (K = 1) affect generation outcomes, based on automatic evaluation results measured on 8000 evaluation samples. Our analysis emphasizes not the absolute metric values, but the direction and magnitude of changes from K = 0 to K = 1 for each context type, as well as the structural inconsistency in how different metrics respond.

Under the no-context condition (K = 0), the three context types form an essentially identical baseline. BLEU-4 is identical at 1.14 across all types, ROUGE-L is 7.48, 7.48, and 7.47 (within 0.01), and METEOR is similarly close at 4.91, 4.91, and 4.88. This confirms that, with the prompt structure, instructions, and target-diff handling held constant, removing context prevents the nominal distinction among context types from producing metric differences. Consequently, the K = 0 setting provides a clean baseline for interpreting changes observed under K = 1 as effects attributable to context injection and context representation.

When switching to the minimal-context condition (K = 1), the magnitude of change and the direction of metric responses become clearly differentiated by context representation. For the Diff-only setting, BLEU-4 increases from 1.14 to 2.49, an absolute gain of +1.35, while ROUGE-L decreases from 7.48 to 6.59 (−0.89). METEOR changes negligibly, from 4.91 to 4.90, effectively remaining flat with a slight decrease. In other words, Diff-only yields a substantial increase in n-gram alignment as reflected by BLEU-4, but a concurrent decrease in sequential/structural alignment as captured by ROUGE-L. This suggests that injecting code-centric context can strengthen token-level overlap signals without necessarily making the overall message structure closer to the reference.

For Message-only, both BLEU-4 and METEOR increase, whereas ROUGE-L decreases relative to K = 0 (from 7.48 to 7.03). Importantly, within the K = 1 setting, ROUGE-L remains higher than Diff-only (7.03 vs. 6.59). METEOR rises from 4.91 to 5.18, a +0.27 increase, which is the largest among the three context representations. This pattern indicates that presenting message text as context may improve lexical-level alignment and matching behavior (including variant forms) more effectively than providing diff text alone.

For the Diff + Message pair setting, BLEU-4 increases from 1.14 to 2.79 (+1.65), ROUGE-L decreases from 7.47 to 6.97 (−0.50), and METEOR increases from 4.88 to 5.05 (+0.17). Although this representation provides the most information, it does not achieve consistent dominance across metrics. In particular, while BLEU-4 exhibits a large gain comparable to Message-only, METEOR shows a more limited increase than Message-only.

These comparisons show that minimal context injection (K = 1) cannot be summarized simply as "improving performance." Even with a single example, metric responses diverge by context type. BLEU-4 increases substantially for all three representations, whereas ROUGE-L and METEOR display mixed behavior, including decreases, stability, and increases depending on what signal is provided. This implies that the metrics are sensitive to different facets of the generated outputs and respond differently to changes in contextual channels.

Given that commit messages are short and highly variable in surface form, the pronounced METEOR gain under Message-only is consistent with the interpretation that message examples directly provide stylistic and lexical cues, thereby strengthening surface alignment signals to which automatic metrics are sensitive. In contrast, under Diff-only, despite providing code-centric cues that could plausibly support change understanding,

neither ROUGE-L nor METEOR shows a consistent increase. This suggests that even if code-centric context contributes to semantic adequacy, its effects may not be reliably captured by single-reference string-similarity metrics.

The results on 8000 samples provide clear evidence that, even under the same minimal context injection regime, the response structure of automatic metrics depends on how retrieved context is represented. Under a BLEU-4-centric view, Diff + Message pair and Message-only show comparably large gains, but when lexical-focused metrics such as METEOR are considered, Message-only exhibits the most salient improvement. Therefore, these findings support interpreting retrieval-augmented prompting effects not through a single metric or a single "improvement" direction, but through the interaction between context type and metric sensitivity.

The findings reported in this study are tightly coupled to the chosen experimental design and evaluation methodology, and this dependence introduces structural threats that can lead to misinterpretation if not made explicit. Accordingly, this section does not merely list limitations; instead, it identifies concrete risk points where the current results could be misunderstood or overgeneralized.

First, the use of BM25-based sparse retrieval fundamentally constrains the interpretation scope of our results. BM25 operates primarily on surface token overlap between a query and documents, making cues such as file paths, identifiers, and string literals in diffs decisive factors for retrieval ranking. As a result, changes with high lexical overlap are more likely to yield appropriate top-ranked examples, whereas changes that are semantically similar but share few surface tokens may produce markedly less relevant retrieved cases. In such situations, observed differences across context types may reflect not a genuine "context construction effect," but rather an accidental prompt response following retrieval failure. In other words, the context-type patterns observed in this study may hold mainly when BM25 successfully retrieves a suitable similar case, and they may fail to reproduce under degraded retrieval quality.

Second, automatic evaluation metrics represent the most direct yet most substantial threat to validity in this work. BLEU, ROUGE, and METEOR are all computed from string-level alignment between a generated message and a reference message. For commit messages, which are extremely short and exhibit high surface-form variability, score fluctuations are often dominated by single-word choices or prefix-form differences. Consequently, the increases observed in this study, particularly the METEOR gain under the Message-only setting and the relative ROUGE-L advantage (smaller decline and higher values within $K = 1$), may reflect stronger stylistic alignment rather than improved reflection of change intent. If such stylistic alignment is mistaken for content-level improvement, the results may be overinterpreted in a direction unrelated to actual change understanding. Therefore, our metric outcomes should be interpreted not as direct evidence of "quality improvement," but as indicators of shifts in surface alignment patterns under controlled condition changes.

Third, comparisons across context types are conducted with a single reference message per commit, which can introduce systematic bias. Although multiple phrasings may be valid summaries of the same change, automatic metrics evaluate proximity to a single reference. This makes conditions that encourage expression patterns similar to the reference structurally advantaged, while outputs that accurately capture the change intent but differ in wording may be penalized. In particular, providing only the message text as context directly exposes stylistic and lexical patterns that are likely to overlap with the reference, potentially creating an inherent advantage under string-similarity metrics. As a result, some score differences across context representations may be artifacts of the evaluation setting rather than genuine differences in generation quality.

Fourth, regarding external validity, our results are strongly tied to a specific dataset and model. The experiments are conducted on a multi-language evaluation set (Python, Java, C++, and JavaScript) derived from MCMD [16] and evaluated with a single model (Qwen-7B). The same context-type response patterns may not hold under different datasets, language distributions, or commit cultures. The same context-type response patterns may not appear under different programming languages, different commit cultures, or projects with different scales and characteristics. Moreover, LLMs differ in prompt sensitivity and in-context learning behavior, so it is not justified to assume that the observed patterns will generalize across models. Accordingly, the reported findings should be interpreted as empirical observations under a specific configuration, not as universal properties of LLM-based commit message generation.

Finally, this study restricts analysis to K = 0 (no example) and K = 1 (a single example). As a result, potential effects that emerge as the number of examples increases, such as information overload, inter-example interference, or context-type-specific saturation patterns, are outside the scope of our analysis. It is plausible that different context representations exhibit different response curves as K grows, but our results provide no guarantees for such expanded settings. Therefore, the observed context-type differences should be understood as phenomena specific to a minimal-context regime, and any generalization to multi-example prompting should be treated with strict caution.

5.2. Comparison with Prior Work

Prior retrieval-augmented and prompt-based commit message generation studies typically evaluate the impact of adding examples or increasing the number of examples. Our findings complement this line of work by holding the retrieved top-1 case and the decoding configuration constant and isolating only how that same case is represented in the prompt. Under this control, we observe that representation alone can induce different metric response patterns, which refines how "example usefulness" should be interpreted when comparing prompting strategies.

Compared with prior retrieval-augmented CMG studies such as RACE, which primarily evaluate the benefit of adding retrieved exemplars to improve generation quality, our results refine the interpretation of 'example usefulness' by showing that the *same* retrieved top-1 case can lead to different metric behaviors depending solely on how it is represented (diff vs. message vs. diff + message) [7]. Moreover, recent LLM-based CMG work emphasizes prompt settings, demonstration design, or multi-step reasoning to boost performance; in contrast, we control these factors and isolate representation as an independent variable under identical decoding and post-processing [2,3]. Finally, retrieval-augmented frameworks such as REACT focus on strengthening retrieval and augmentation pipelines, whereas our findings highlight that even after retrieval is fixed, representation choice remains a non-trivial design decision that induces metric trade-offs and affects evaluation interpretation [8].

6. Related Research

Commit message generation aims to summarize the intent of a code change, represented as a diff, into a concise one-line natural-language description. This problem has been studied extensively in code summarization and software engineering automation. Early studies primarily treated diffs as sequential inputs and applied sequence-to-sequence models, or provided both pre-change and post-change code fragments to better capture change context [18,19]. While these approaches focus on learning a mapping between code changes and messages, prior work has also noted limitations in generalization due to the

heterogeneous structure of diffs (e.g., file paths, identifiers, added/deleted lines) and their large length variance [20].

With the rise in large-scale pretraining, general-purpose language models capable of processing both code and natural language have emerged, and an increasing number of efforts have explored applying such models to commit message generation [21,22]. More recently, practical use has expanded toward prompt-based generation without fine-tuning by leveraging the in-context learning capabilities of LLMs [3,5]. In this line of work, researchers often emphasize careful input formatting, explicit output constraints, and the inclusion of shown examples to improve output stability, rather than making changes to model architectures. However, because commit messages admit diverse valid phrasings for the same change and because stylistic conventions vary across projects, it remains difficult to characterize usefulness solely through score comparisons across models; this concern has been repeatedly raised in prior studies [20].

Against this background, retrieval-based methods have been adopted as a means of providing auxiliary context for commit message generation [7,20]. Retrieval-Augmented Generation (RAG) was proposed to complement parameter-only generation by retrieving external documents or examples and incorporating them into model inputs [6]. In commit message generation, the retrieval corpus is often not a generic document set but a collection of historical commit cases [7,20]. The purpose of retrieval in this setting is to examine how presenting similar historical cases alongside the target change affects generation behavior. More broadly, in code-related generation tasks, there is a growing trend toward providing similar code snippets, similar descriptions, or similar change histories as contextual evidence to improve output specificity.

In commit message generation, retrieved context can differ substantially depending on what information is provided. Some work emphasizes providing similar diffs to strengthen change context, whereas other studies focus on providing similar commit messages to induce writing style and phrasing patterns [2,7]. Nevertheless, much of the existing literature has centered on whether examples are provided at all, or how performance changes as the number of examples increases. Systematic comparisons of how the same retrieved result produces different outcomes depending on how it is represented in the prompt remain relatively limited [7,20].

From the retrieval perspective, both sparse and dense paradigms have been explored. Dense retrieval offers the advantage of capturing semantic similarity via embeddings, but retrieval outcomes can be sensitive to the choice of embedding model, index construction, and hyperparameter settings, and computational costs are often higher [23]. In contrast, sparse retrieval methods such as BM25 are simple, reproducible, and frequently reported to be strong baselines for code- and commit-related text where file paths and identifiers provide salient lexical cues [9,10]. When the objective is to compare prompt design factors or example presentation strategies, reducing complexity in the retrieval stage is often an analytically appropriate choice for controlled comparison.

On the evaluation side, commit message generation studies have widely used string-based similarity metrics such as BLEU, ROUGE, and METEOR. These metrics compute n-gram overlap or subsequence-level similarity between generated outputs and reference messages, enabling efficient large-scale comparisons. However, because commit messages are short, compact, and flexible in expression, surface overlap with a single reference may not adequately reflect quality. As messages become shorter, a single word can dominate score differences, and stylistic elements such as prefixes and conventional phrases can strongly influence metric outcomes. Consequently, automatic metrics are useful for relative comparisons and trend analysis, but they are constrained as proxies for developer preference or real-world utility.

Retrieval-augmented example provision has become an increasingly common design element in commit message generation, yet prior work has more often emphasized example inclusion or example count, rather than how retrieved cases are represented in prompts. This study compares three representations while keeping both the retrieval pipeline and the prompt skeleton fixed: presenting the retrieved case as code changes only, as message text only, or as both code changes and message. We further contrast a no-example condition with a minimal-context condition that includes only a single top-1 example, thereby removing confounding factors that arise when the number of examples increases. This perspective complements prior analyses by focusing on how fine-grained design choices in retrieval-augmented in-context prompting translate into differences in generation behavior and metric response patterns.

Retrieval-augmented example provision has become an increasingly common design element in commit message generation. Prior work has largely emphasized whether examples are included and how performance changes as the number of examples increases, while comparatively less attention has been paid to how the same retrieved case is represented in the prompt (e.g., diff-form vs. message-form) and how such representation choices interact with automatic metrics. This motivates our controlled analysis: we hold the retrieval pipeline and the retrieved top-1 case constant and vary only the context representation injected into an identical prompt skeleton. In the next sections, we describe the experimental design and report results under a minimal-context setting (K = 0 vs. K = 1) to isolate representation-driven changes in measured alignment behavior.

7. Conclusions

This study quantitatively investigated how context inclusion and context representation affect automatic commit message generation with large language models when historical commit cases retrieved via similarity-based search are injected into the prompt. Rather than focusing on the mere presence of retrieval-augmented examples, we designed our experiments to isolate how exposing the same retrieved result in different information forms is reflected in changes in automatic evaluation metrics.

Using Qwen-7B and 8000 evaluation samples, we compared a no-context baseline (K = 0) against a minimal-context setting that injects a single top-1 similar case (K = 1). Under K = 1, we further compared three context representations: Diff-only, Message-only, and Diff + Message pair. Across all conditions, we held the prompt skeleton, decoding configuration, output constraints, and post-processing rules constant to ensure that observed differences can be attributed to context construction rather than incidental variation in generation settings.

The results show that under K = 0, BLEU-4, ROUGE-L, and METEOR form an almost identical baseline across the three nominal context types, confirming that when the example-context block is removed, the context-type distinction itself does not induce measurable differences. In contrast, under K = 1, BLEU-4 increases relative to the zero-shot baseline for all context types, while ROUGE-L decreases across all types. These findings indicate that even minimal context injection can shift string-based alignment metrics. However, METEOR responds differently depending on context representation: it increases under Message-only, decreases slightly under Diff-only, and exhibits only limited change under Diff + Message pair. Comparing the context types within K = 1, BLEU-4 is higher for Message-only and Diff + Message pair than for Diff-only; ROUGE-L decreases overall under K = 1 but remains relatively higher for Message-only than for the other representations; and METEOR shows the largest increase under Message-only.

These results suggest that it is difficult to generalize that providing "more" contextual information consistently improves all automatic metrics. Instead, the type of signal

included in the prompt context can induce structurally different metric response patterns, even under the same minimal one-shot setting. Our findings empirically show that in retrieval-augmented in-context prompting, not only the presence of context but also its representation can differentially shape surface alignment behavior captured by common metrics. Notably, even with a single example, the nature of the contextual information appears to be a key design factor distinguishing output alignment characteristics and metric sensitivity. This reinforces the broader implication that prompt design at inference time, even without fine-tuning or architectural modification, can measurably influence generation behavior.

At the same time, the metric changes observed in this study should not be directly interpreted as improvements in semantic adequacy or in how well the generated message reflects the true change intent. Commit messages are short and exhibit substantial variation in surface form, and automatic metrics may respond strongly to stylistic alignment and lexical overlap. Accordingly, our conclusions should be understood in terms of differences in alignment patterns captured by automatic metrics, and the practical usefulness of each context representation for actual change understanding requires additional validation.

Future work can strengthen these conclusions by incorporating complementary evaluations that more directly assess content fidelity, such as auxiliary measures of diff–message semantic consistency or human judgments. It is also necessary to test how context-type effects evolve as the number of examples increases, and whether similar response patterns hold across different LLMs with varying in-context learning behavior. Such follow-up studies would help delineate the applicability boundaries of the context-representation-centered observations reported in this work.

Author Contributions: Conceptualization, D.A. and G.Y.; Methodology, D.A.; Software, D.A.; Validation, D.A. and G.Y.; Formal analysis, D.A.; Investigation, D.A.; Resources, G.Y.; Data curation, D.A.; Writing—original draft preparation, D.A.; Writing—review and editing, D.A. and G.Y.; Visualization, D.A.; Supervision, G.Y.; Project administration, G.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Tao, W.; Wang, Y.; Shi, E.; Du, L.; Han, S.; Zhang, H.; Zhang, D.; Zhang, W. On the Evaluation of Commit Message Generation Models: An Experimental Study. In Proceedings of the 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), Luxembourg, 27 September–1 October 2021; pp. 126–136. [CrossRef]
2. Li, J.; Faragó, D.; Petrov, C.; Ahmed, I. Only Diff Is Not Enough: Generating Commit Messages Leveraging Reasoning and Action of Large Language Model. *Proc. ACM Softw. Eng.* **2024**, *1*, 745–766. [CrossRef]
3. Wu, Y.; Wang, Y.; Li, Y.; Tao, W.; Yu, S.; Yang, H.; Jiang, W.; Li, J. An Empirical Study on Commit Message Generation using LLMs via In-Context Learning. In Proceedings of the 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), Ottawa, ON, Canada, 26 April–6 May 2025. [CrossRef]
4. Jiang, S.; Armaly, A.; McMillan, C. Automatically Generating Commit Messages from Diffs Using Neural Machine Translation. *arXiv* **2017**, arXiv:1708.09492. Available online: https://arxiv.org/abs/1708.09492 (accessed on 20 January 2026). [CrossRef]
5. Brown, T.B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models are Few-Shot Learners. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Online, 6–12 December 2020.
6. Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.-t.; Rocktäschel, T.; et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Online, 6–12 December 2020.

7. Shi, L.; Liang, Y.; Jiang, H.; Yu, L. RACE: Retrieval-Augmented Commit Message Generation. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP), Abu Dhabi, United Arab Emirates, 7–11 December 2022.
8. Zhang, L.; Zhang, H.; Wang, C.; Liang, P. RAG-Enhanced Commit Message Generation. *arXiv* **2024**, arXiv:2406.05514. [CrossRef]
9. Robertson, S.; Zaragoza, H. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* **2009**, *3*, 333–389. [CrossRef]
10. Robertson, S.E.; Walker, S.; Jones, S.; Hancock-Beaulieu, M.; Gatford, M. Okapi at TREC-3. In Proceedings of the 3rd Text REtrieval Conference(TREC-3), Gaithersburg, MD, USA, 2–4 November 1994.
11. Bai, J.; Bai, S.; Chu, Y.; Cui, Z.; Dang, K.; Deng, X.; Fan, Y.; Ge, W.; Han, Y.; Huang, F.; et al. Qwen Technical Report. *arXiv* **2023**, arXiv:2309.16609. Available online: https://arxiv.org/abs/2309.16609 (accessed on 20 January 2026). [CrossRef]
12. Papineni, K.; Roukos, S.; Ward, T.; Zhu, W.-J. BLEU: A Method for Automatic Evaluation of Machine Translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, PA, USA, 7–12 July 2002; pp. 311–318.
13. Lin, C.-Y. ROUGE: A Package for Automatic Evaluation of Summaries. In Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004), Barcelona, Spain, 25–26 July 2004; pp. 74–81.
14. Banerjee, S.; Lavie, A. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization, Ann Arbor, MI, USA, 29 June 2005; pp. 65–72.
15. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention Is All You Need. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Long Beach, CA, USA, 4–9 December 2017.
16. Wei, T. MCMD | Multi-Programming-Language Commit Message Dataset. *Zenodo*, Dataset, Ver. v2, Apr. 2021. Available online: https://zenodo.org/records/5025758 (accessed on 20 January 2026).
17. An, D. Appl_CMG_Dataset_8000, GitHub Repository. 2025. Available online: https://github.com/adk0715/dataset8000 (accessed on 20 December 2025).
18. Liu, Z.; Xia, X.; Hassan, A.E.; Lo, D.; Xing, Z.; Wang, X. Neural-machine-translation-based commit message generation: How far are we? In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE), Montpellier, France, 3–7 September 2018; pp. 373–384. [CrossRef]
19. Liu, S.; Gao, C.; Chen, S.; Nie, L.; Liu, Y. ATOM: Commit message generation based on abstract syntax tree and hybrid ranking. *IEEE Trans. Softw. Eng.* **2020**, *48*, 1800–1817. [CrossRef]
20. Li, J.; Ahmed, I. Revisiting Learning-based Commit Message Generation. In Proceedings of the 45th International Conference on Software Engineering (ICSE), Melbourne, Australia, 14–20 May 2023.
21. Jung, H. Commit Message Generation Using Pre-Trained Neural Machine Translation Model. In Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog) (Workshop at ACL-IJCNLP 2021), Online, 6 August 2021.
22. Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H.P.d.O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. Evaluating Large Language Models Trained on Code. *arXiv* **2021**, arXiv:2107.03374. [CrossRef]
23. Karpukhin, V.; Oguz, B.; Min, S.; Lewis, P.; Wu, L.; Edunov, S.; Chen, D.; Yih, W.-T. Dense Passage Retrieval for Open-Domain Question Answering. *arXiv* **2020**, arXiv:2004.04906. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.