Article

Not peer-reviewed version

AI-Driven Code Documentation: Comparative Evaluation of LLMs for Commit Message Generation

Mohamed Mehdi Trigui * , Wasfi G. Al-Khatib * , Mohammad Amro * , Fatma Mallouli

Posted Date: 24 December 2025

doi: 10.20944/preprints202512.2193.v1

Keywords: large language models; commit message generation; retrieval-augmented generation; CommitBench; transformer-based models; automatic and human evaluation

Preprints.org is a free multidisciplinary platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This open access article is published under a Creative Commons CC BY 4.0 license, which permit the free download, distribution, and reuse, provided that the author and preprint are cited in any reuse.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025 doi:10.20944/preprints202512.2193.v1

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.

Article

AI-Driven Code Documentation: Comparative Evaluation of LLMs for Commit Message Generation

Mohamed Mehdi Trigui [1,*], Wasfi G. Al-Khatib [1,2,*], Mohammad Amro [1,2,*] and Fatma Mallouli [3]

[1] Information & Computer Science Department (ICS), King Fahd University of Petroleum & Minerals (KFUPM), Dhahran 31261, Saudi Arabia
[2] Interdisciplinary Research Center for Intelligent Secure Systems (IRC-ISS), King Fahd University of Petroleum & Minerals (KFUPM), Dhahran 31261, Saudi Arabia
[3] Deanship of Preparatory Year and Supporting Studies, Department of Computer Science, Imam Abdulrahman Bin Faisal University, Dammam 31441, Saudi Arabia
* Correspondence: g202318370@kfupm.edu.sa (M.M.T.); wasfi@kfupm.edu.sa (W.G.K.); mamro@kfupm.edu.sa (M.A.)

Abstract

Commit messages are essential for understanding software evolution and maintaining traceability of projects; nevertheless, their quality varies across repositories. Recent Large Language Models provide a promising path to automate this task by generating concise context and sensitive commit messages directly from code diffs. This paper provides a comparative study of three paradigms of large language models: zero-shot prompting, retrieval augmented generation, and fine-tuning, using the large scale **CommitBench** dataset that spans six programming languages. We assess the performance of the models with automatic metrics, namely BLEU, ROUGE-L, METEOR, and Adequacy, and a human assessment of 100 commits. In the latter, experienced developers rated each generated commit message for Adequacy and Fluency on a five-point Likert scale. The results show that fine-tuning and domain adaptation yield models that perform consistently better than general-purpose baselines across all evaluation metrics, thus generating commit messages with higher semantic adequacy and clearer phrasing than zero-shot. The correlation analysis suggests that the Adequacy and BLEU scores are closer to human judgment, while ROUGE-L and METEOR tend to underestimate the quality in cases where the models generate stylistically diverse or paraphrased outputs. Finally, the study outlines a conceptual integration pathway for incorporating such models into software development workflows, emphasizing a human in the loop approach for quality assurance.

Keywords: large language models; commit message generation; retrieval-augmented generation; CommitBench; transformer-based models; automatic and human evaluation

---

1. Introduction

Modern collaborative software development relies heavily on distributed version control systems such as Git, which record both source code changes and corresponding textual descriptions known as *commit messages*. These messages capture what was modified and, ideally, the rationale behind the change. High quality commit messages facilitate code review, traceability, and maintenance, while poor or missing documentation hinders debugging, regression analysis, and onboarding.

Despite their importance, commit messages in practice are often incomplete or inconsistent. Under time pressure or without enforced standards, developers frequently produce vague descriptions (e.g., "fix bug", "update code"), reducing the value of the commit history for future contributors. Prior work has shown that unclear commit documentation introduces technical debate at the communication level and increases maintenance costs [1–3].

Recent progress in *Large Language Models* (LLMs) has created a realistic opportunity to automate commit message authoring. Transformer-based models such as ChatGPT, DeepSeek, and Qwen

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025 doi:10.20944/preprints202512.2193.v1

demonstrate strong capability in summarizing source code diffs, reasoning about intent, and producing concise, human like text [4–6]. However, several open questions remain unresolved:

- How do different LLM paradigms zero-shot prompting, retrieval-augmented generation (RAG), and fine-tuned models compare in their ability to generate accurate and informative commit messages?
- To what extent do automatic evaluation metrics (BLEU, ROUGE-L, METEOR, Adequacy) correlate with human judgments of clarity and usefulness?
- How well do such models generalize across programming languages and commit types within large, heterogeneous datasets?

To answer these questions, this paper presents a comprehensive empirical comparison of representative LLMs using the **CommitBench** dataset, which contains over one million real commits across six major programming languages [7,8]. We evaluate each paradigm under controlled experimental settings and validate quantitative results through a human study of one hundred sampled commits rated on a 1–5 scale for Adequacy and fluency.

While the primary focus of this work is empirical evaluation, we also discuss how the resulting models could be conceptually integrated into modern development workflows, emphasizing a human in the loop design. This conceptual illustration is not claimed as an implementation but as a forward looking extension of the findings.

1.1. Contributions

This paper presents a systematic comparative study of LLM based commit message generation with and without retrieval, grounded in reproducible experimentation. Our main contributions are:

1. **Evaluation of three representative model paradigms.** Although six configurations were explored during early experimentation, this paper reports three stable and comparable setups: (i) ChatGPT (zero-shot), (ii) DeepSeek-RAG (retrieval-augmented), and (iii) Qwen-Commit (fine-tuned open model). These configurations represent the three dominant families of approaches.
2. **Creation of a balanced evaluation subset from CommitBench**. We use the **CommitBench** corpus ($> 10^6$ commits) to create a balanced subset covering six major programming languages and standardized pairs of diff messages, making the comparison fair.
3. **Analysis of human and automatic evaluation alignment** : We analyze the correlation between automatic metrics and human judgments by pointing out where metrics agree or diverge, and discuss implications for future evaluation practices.
4. **Development of a CI/CD pipeline integration sketch.** We outline a practical, human in the loop workflow for embedding the best performing configuration(s) into pre-commit hooks and CI pipelines to assist documentation and code review.

1.2. Research Questions

To guide the study, we pose two core research questions:

RQ1: How do current LLMs, viz., *ChatGPT*, *DeepSeek*, and the fine-tuned *Qwen* model compare in their ability to generate accurate and natural commit messages, both with and without retrieval augmentation?

RQ2: To what extent do automatic evaluation metrics (BLEU, ROUGE-L, METEOR, Adequacy) correlate with human judgments of **Adequacy** (semantic accuracy) and **Fluency** (linguistic quality)?

These questions reflect the key developer oriented concerns of selecting an appropriate model and trusting metric based evaluations as reliable quality indicators.

1.3. Paper Structure

The remainder of this paper is organized as follows. Section 2 reviews prior work on automated commit message generation, LLM based code summarization, and evaluation practices. Then, Section 3

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025 doi:10.20944/preprints202512.2193.v1

introduces the **CommitBench** dataset and outlines the preprocessing pipeline used to construct a balanced, high quality benchmark. After that, Section 4 describes the overall experimental methodology including model configurations, fine-tuning of the Qwen model, retrieval augmented generation and evaluation metrics. This is followed by Section 5 that details the experimental setup and human evaluation design. Section 6 reports and analyzes the comparative results whereas Section 7 interprets the findings and discusses practical implications for CI/CD pipeline integration. Finally, Section 8 summarizes current limitations and directions for future research. followed by the conclusion in Section 9.

2. Related Work

2.1. Role of Commit Messages in Software Engineering

Commit messages are crucial contexts for code evolution; they not only describe what has changed but also explain why [1,2]. Good quality messages decrease the cognitive load when doing code reviews, speed up onboarding, and facilitate debugging as well as auditing. Tian et al. [3] reported that developers appreciate messages specifying both the technical change and its rationale, but many commits do not include one or both of these components. Poor documentation at this level leads to the growth of technical debt in communication and knowledge transfer.

2.2. Traditional Heuristic and Template Based Methods

Early automation efforts relied on heuristic rules or hand crafted templates [9]. These approaches usually inserted filenames or identifiers into fixed sentence patterns. Although useful for repetitive updates, they failed to capture higher level intent, describing *what* changed but not *why* [1]. Their rigidity limited generalization across projects and programming styles.

2.3. Neural and Sequence to Sequence Approaches

The task was later reframed as code to text translation, mapping diffs (source) to commit messages (target). Sequence to sequence models using RNNs, attention, and subsequently Transformer encoders decoders improved fluency and contextual coherence [10–12]. These systems learned statistical correspondences between code edits and summary phrases but still struggled with long diffs, ambiguous intent, and overly generic wording.

2.4. Transformer Models and Large Language Models

Transformer-based LLMs brought a step change in performance [6]. With large-scale pretraining, they can reason over code context and generate concise, human like summaries of software changes [4,13,14]. Recent studies extend their use to inferring developer intent, linking commits to bug reports or feature requests, and producing audit ready explanations [6,14]. Prompt engineering and retrieval augmented generation (RAG) further enable context adaptive outputs without full retraining [5,25].

2.5. Datasets and Benchmarks

The introduction of large, multilingual datasets such as **CommitBench** has improved reproducibility and comparability across models [7,8,15]. Earlier work often relied on private or single project corpora, limiting external validity. CommitBench aggregates over one million real commits from diverse repositories and languages, offering a robust basis for evaluating both general purpose and fine-tuned models under realistic noise conditions.

2.6. Evaluation Practices and Human Factors

Automatic metrics (BLEU, ROUGE-Land METEOR) remain the dominant evaluation tools. In this context, they quantify lexical overlap between generated and reference messages. However, multiple studies note that these metrics correlate imperfectly with developer preferences, penalizing legitimate paraphrases or rewarding surface similarity over semantic adequacy [2,6]. Consequently,

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025 doi:10.20944/preprints202512.2193.v1

recent research complements automatic scores with human assessments focused on meaning accuracy and linguistic quality [13,15].

2.7. Summary

While LLMs have demonstrated remarkable capability for code summarization, prior work seldom provides a unified and cross-model comparison grounded in both automatic and human evaluation. This study directly addresses that gap through a reproducible, dual evaluation framework encompassing ChatGPT, DeepSeek, and a fine-tuned Qwen model.

3. Dataset and Preprocessing

3.1. CommitBench Overview

We base our experiments on **CommitBench**, a large-scale benchmark for commit message generation introduced by Schall et al. [7] and later extended by Kosyanenko and Bolbakov [8]. CommitBench aggregates more than one million real commits collected from thousands of open source repositories across six major programming languages: Python, Java, JavaScript, Go, PHP, and Ruby. Each instance contains a `git-diff`, the corresponding human written commit message, and metadata such as author, timestamp, and repository name.

To our knowledge, this is the first systematic preprocessing of CommitBench that balances commits across six programming languages while removing bot generated and trivial messages at scale. Figure 1 shows the overall language distribution in the raw dataset, Because, as we can see, the irregularities, including duplicate labels (e.g., `py,py`) need to be taken into consideration. While Figure 2 provides the number of commits per repository group. The dataset is inherently imbalanced, as Python and Java projects dominate other programming languages. Therefore, additional preprocessing was required to construct a balanced evaluation subset.

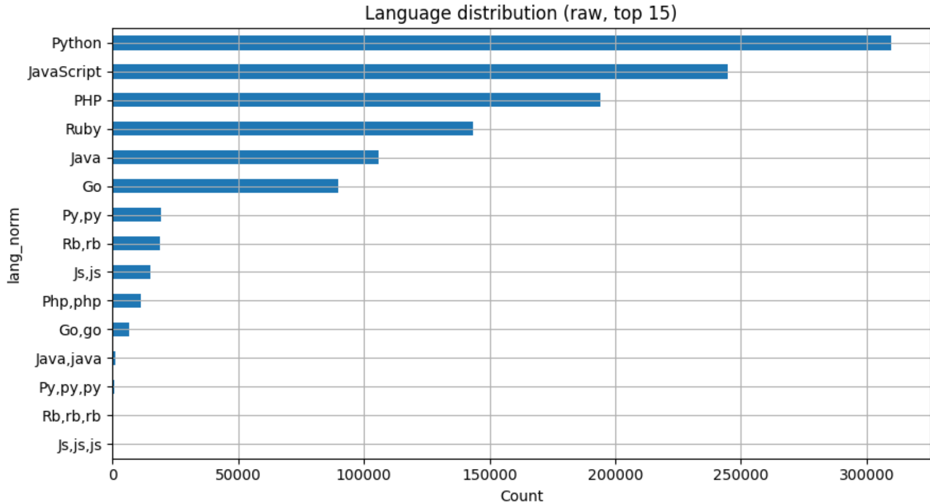

Figure 1. Distribution of programming languages in the raw CommitBench dataset.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

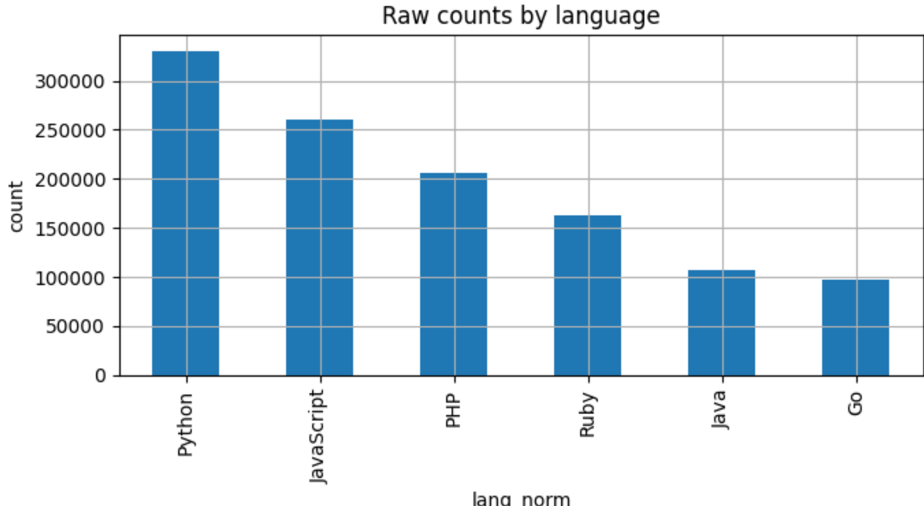

Figure 2. Commit counts per language and repository group before filtering.

3.2. Noise Filtering and Quality Control

Because the original data originate from heterogeneous public repositories, we applied multiple filtering stages to remove noise and low-information samples. We identified and discarded two main categories of noisy commits:

- **Bot generated or automated commits**, typically produced by continuous integration agents such as "dependabot", "jenkins", or "github-actions".
- **Trivial or redundant messages**, such as "update readme", "merge branch", or "fix typo", which provide little semantic value.

Figures 3 and 4 illustrate the distribution of these cases prior to filtering. Regular expressions and heuristic keyword detection were used to flag such commits, following recommendations from prior work [13,15].

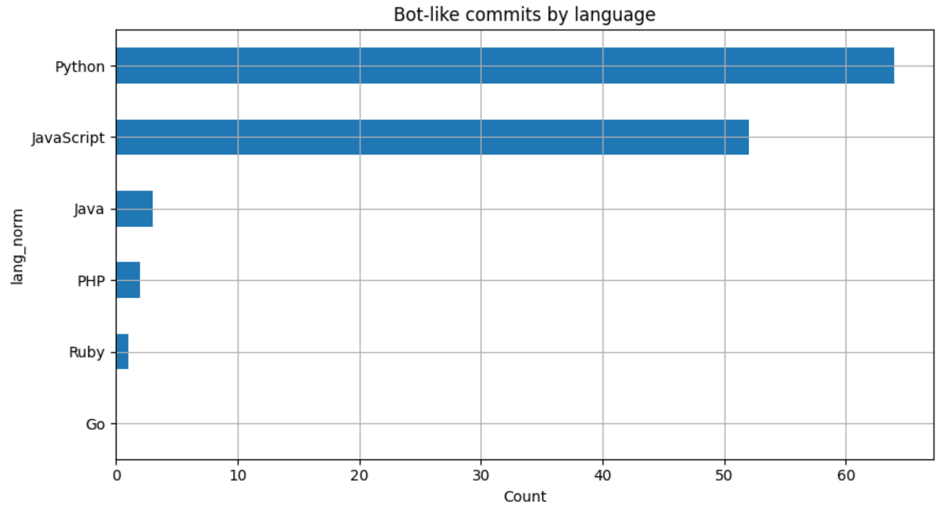

Figure 3. Detection and removal of bot-like commit messages across languages.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025
doi:10.20944/preprints202512.2193.v1

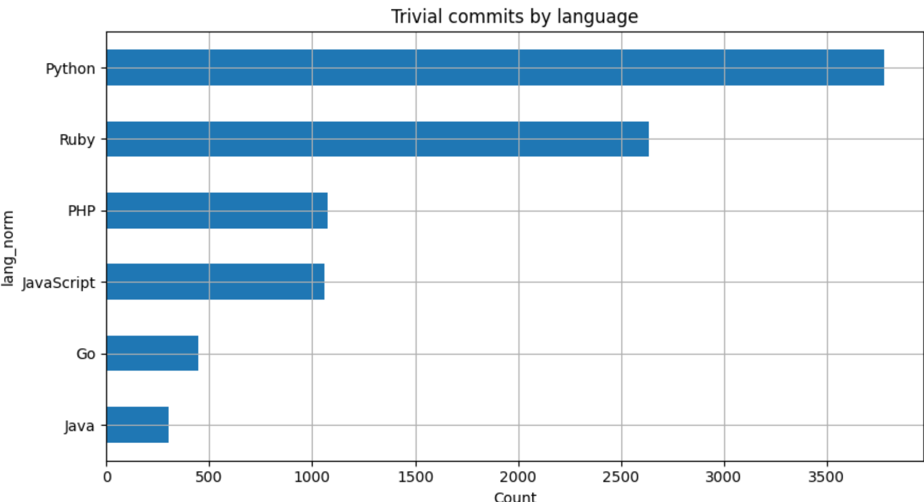

Figure 4. Trivial commit messages filtered during preprocessing.

3.3. Length and Structural Analysis

To understand the natural distribution of commit message lengths, we computed descriptive statistics over tokenized messages. As shown in Figure 5, the median message length ranged between 7 and 15 tokens depending on the programming language, which aligns with prior findings [11,16]. Messages shorter than three tokens were automatically discarded, as they typically corresponded to boilerplate or placeholder text.

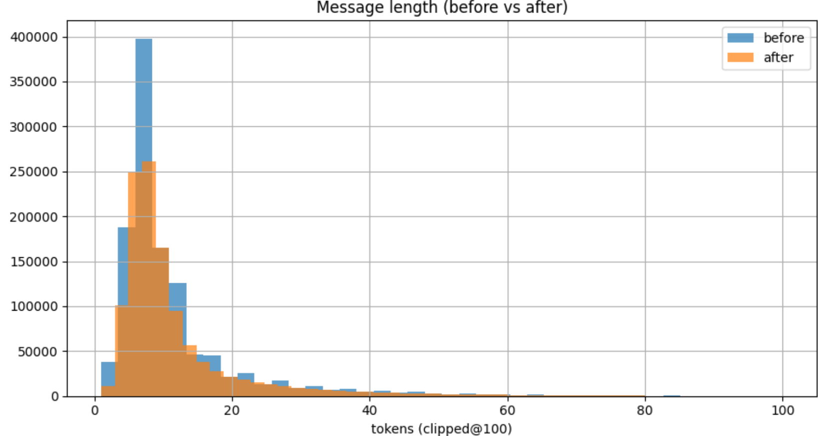

Figure 5. Distribution of token lengths across commit messages before balancing.

3.4. Balancing and Final Subset Construction

After cleaning, the dataset was stratified by programming language and randomly sampled to create a balanced subset. Figure 6 shows the resulting distribution. Thereafter, it was partitioned into training, validation, and test partitions in an 80/10/10 ratio; with care taken that commits from the same repository did not appear across splits, thus preventing data leakage.

For large-scale automatic evaluation, we used the entire balanced corpus (576,342 commits in six languages).

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025 doi:10.20944/preprints202512.2193.v1

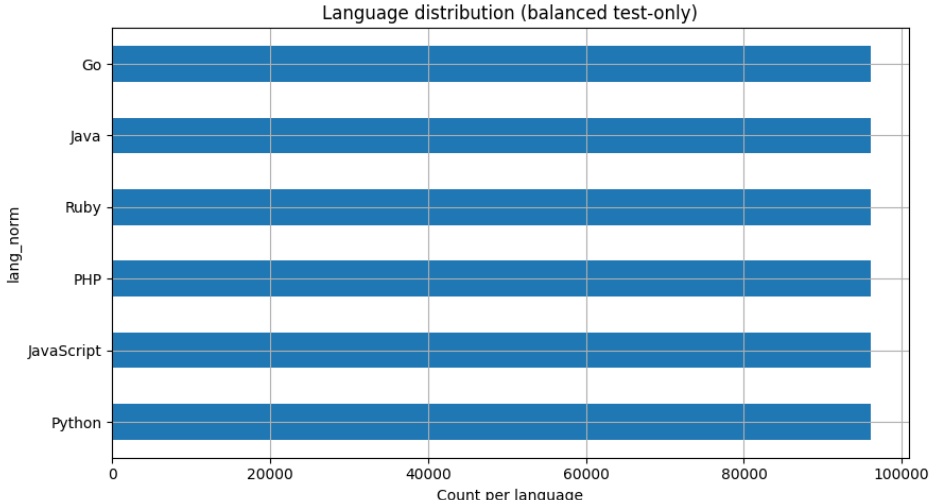

Figure 6. Balanced subset of CommitBench after filtering and normalization.

3.5. Preprocessing Pipeline and Summary

Figure 7 depicts the entire preprocessing funnel applied to the raw CommitBench dataset. This pipeline is divided into five sequential stages: deduplication, removal of bot and trivial messages, language normalization, diff-length filtering, and balancing by programming language. It refines data quality in a series of steps by removing redundancy, removing automated or low information commits, and making sure representations across all six languages are consistent.

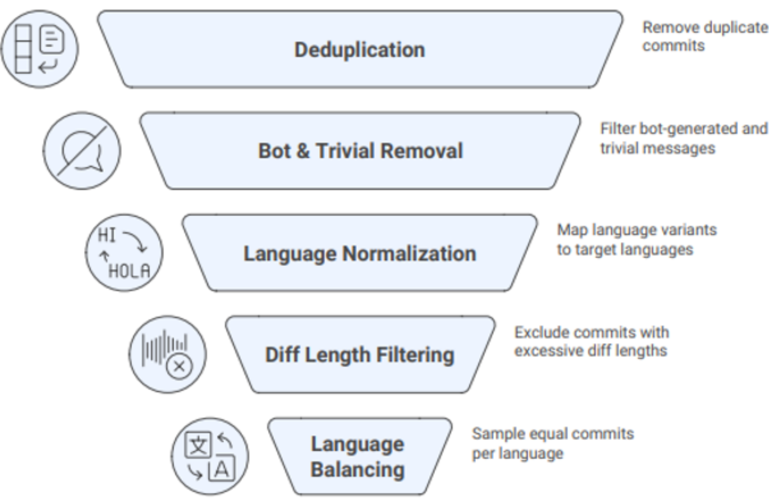

Figure 7. End-to-end preprocessing pipeline used to construct the final CommitBench subset.

Table 1 gives a quantitative summary of the number of commits remaining after each filtering stage. Thus, the original dataset contained approximately 1.16 million commits after language normalization. After duplicate removal, bot detection, trivial message exclusion, and length-based cleaning, sequential filtering resulted in a corpus of 1.15 million high-quality samples. From this, a balanced evaluation subset of 576,342 commits across six programming languages was extracted for subsequent experiments.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025
doi:10.20944/preprints202512.2193.v1

Table 1. Overview of preprocessing steps and resulting dataset sizes.

Step	Count Remaining
Raw total (after normalization)	1,165,213
Remove duplicates	1,165,213
Remove bot-like commits	1,165,091
Remove trivial/low-information messages	1,155,797
Length-based filtering	1,154,926
Balanced evaluation subset (6 languages)	576,342

With this preprocessing, each commit message was ensured to be semantically relevant, linguistically diverse, and well-represented across all six programming languages. The resultant cleaned and stratified subset forms the basis of the comparative LLM experiments shown in Section 4.

4. Methodology

4.1. Experimental Overview

Our methodology is designed to evaluate large language models (LLMs) for commit message generation along three complementary dimensions: (i) model architecture and training paradigm, (ii) retrieval and prompting strategy, and (iii) evaluation criteria combining automatic metrics and human judgments. The workflow is illustrated in Figure 8, which shows the end to end pipeline from dataset preprocessing (Section 3) to quantitative and qualitative analysis.

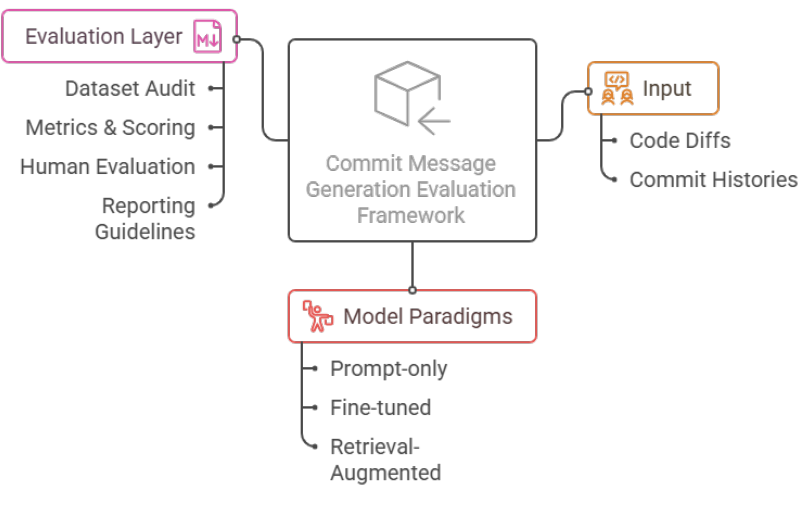

Figure 8. Methodology overview showing the three evaluation paradigms: zero-shot prompting, retrieval augmented generation (RAG), and fine-tuned inference.

4.2. Model Suite

To capture the diversity of current approaches in automated commit message generation, we evaluated three complementary LLM paradigms [5,6,14]:

1. **General-purpose models (ChatGPT and DeepSeek).** Both are proprietary transformer-based LLMs accessed via API. We evaluated them in two configurations: a standard *zero-shot* setting and a *retrieval augmented* (RAG) variant, where the prompt was enriched with the top-k semantically similar historical commits retrieved from the CommitBench index (see Section 4.4). This dual

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025 doi:10.20944/preprints202512.2193.v1

setup assesses whether contextual retrieval improves coherence, style consistency, and repository alignment.

• Fine-tuned open model (Qwen-Commit). Locally hosted, open-source model fine-tuned on CommitBench diffs and reference messages using parameter efficient adaptation. Qwen-Commit was also tested in both standalone and RAG-enhanced modes to examine the combined effect of domain-specific fine-tuning and retrieval grounding. It is designed to emphasize data privacy and on-premise feasibility while keeping domain alignment high.

All models have undergone evaluation under identical input and output conditions: Each was given the same preprocessed `git-diff`, a unified prompt template, and a fixed maximum generation length to avoid verbosity and API overflow (128 tokens). This standardization ensures that observed differences are due to the model architecture and adaptation strategy, not because of changes in the prompt.

4.3. Prompt Engineering

Prompt formulation was a factor that impacted the clarity and succinctness of the auto-generated text, as illustrated by similar findings from the study by Lopes et al. [13]. A set of pilot experiments revealed that the simple questions like *"Summarize this diff"* were resulting in verbose and redundant responses lacking explicit explanation. Upon several rounds of tuning on the validation subset, the following final template instruction has been adopted:

> You are a senior software engineer. Write a short (<=12 words)
>
> Git commit message that clearly describes what changed and why.

This instruction enforces an imperative tone and brevity aligned with established open-source conventions [3]. The limit of 12 words per line makes models generate concise, active voice messages much like actual developer commits, while still allowing inclusion of both technical change and its motivation.

4.4. Retrieval Augmented Generation (RAG)

To improve the contextual relevance and repository specific consistency, we extended the baseline generation pipeline with a **Retrieval Augmented Generation (RAG)** mechanism. Instead of relying solely on internal parameters of a model, RAG augments the input prompt with external exemplars historical (`diff`, message) pairs most similar to the current code change. This allows general-purpose LLMs to more accurately reflect local naming conventions, stylistic norms, and commit structures without explicit fine-tuning.

Each input diff is first encoded as a dense vector by a sentence transformer model trained on code text pairs. The resulting embeddings are compared against an indexed "commit memory" built from CommitBench using FAISS similarity search. The top-k examples retrieved for every new diff are inserted, empirically setting $k = 3$, before the prompt template described in Section 4.3. This retrieval step provides concrete, domain-aligned context that guides message generation. Figure 9 shows the detailed architecture of the Retrieval-Augmented Generation (RAG) pipeline. It explains how the retriever module combines TF–IDF vectorization and MiniLM embeddings, uses cosine similarity for re-ranking, retrieves contextual examples, and injects them into the LLM prompt in order to generate commit messages.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025 doi:10.20944/preprints202512.2193.v1

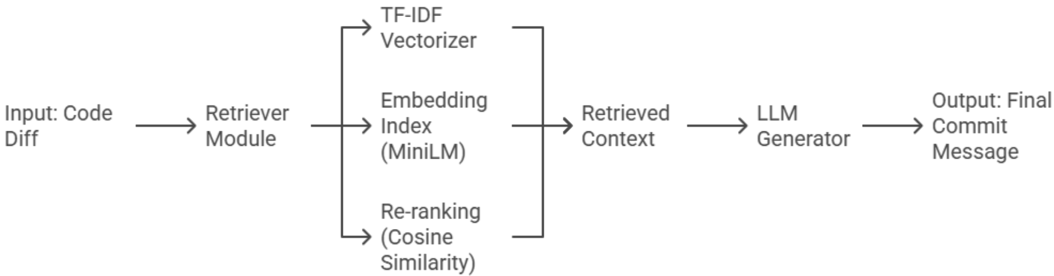

Figure 9. Retrieval Augmented Generation (RAG) pipeline. Each input diff is embedded, matched against an indexed memory of historical commits, and supplemented with the top-k exemplars before message generation.

4.4.1. Embedding Model and Similarity Computation

Subsequently, we used the `sentence transformers` library to convert each commit diff in the preprocessed dataset into a dense vector representation using the `all-MiniLM-L6-v2` sentence transformer model. This embedding model has been chosen due to its strong balance between semantic accuracy and computational efficiency [5,6]. MiniLM learns both lexical and structural similarities, allowing the retrieval mechanism to generalize across code and natural language fragments.

For every new diff d_{query}, the embedding $\mathbf{v}_q$ was compared against all indexed commit vectors $\mathbf{v}_i$ in a FAISS flat index [17]. Cosine similarity was computed as:

$$\text{sim}(d_{\text{query}}, d_i) = \frac{\mathbf{v}_q \cdot \mathbf{v}_i}{||\mathbf{v}_q||\,||\mathbf{v}_i||}.$$

The k most similar examples (by default $k = 3$) were retrieved and passed to the model as contextual exemplars.

4.4.2. Re-Ranking Strategy

Beyond raw similarity scores, a lightweight **re-ranking module** prioritized examples with higher lexical overlap in function names and identifiers. Specifically, a TF–IDF similarity matrix was computed, and the top-$n = 10$ retrieved samples were rescored by combining embedding and lexical similarities:

$$\text{score}_{\text{hybrid}} = \alpha \cdot \text{sim}_{\text{emb}} + (1 - \alpha) \cdot \text{sim}_{\text{tfidf}},$$

where $\alpha = 0.7$ was empirically tuned on the validation subset to balance semantic and lexical information. The final top-$k = 3$ samples after re-ranking were appended to the prompt in the following format:

```
Example Commit 1: <diff>\nMessage: <msg>
Example Commit 2: <diff>\nMessage: <msg>
Now write a commit message for the following diff: ...
```

This two-tier retrieval ensures that the retrieved examples are both semantically and syntactically aligned with the query diff.

4.4.3. TF–IDF vs. Hybrid Retrieval

We evaluated two retrieval schemes to assess the relative contribution of lexical and semantic matching:

- **TF–IDF retrieval:** purely lexical retrieval based on tokenized diff text. It effectively captures explicit term overlap (e.g., variable names or function identifiers) but struggles when similar intents use different identifiers.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025 doi:10.20944/preprints202512.2193.v1

- **Hybrid retrieval:** integrates TF–IDF and embedding-based similarity using the weighted score above. This approach captures both surface overlap and latent semantic relationships, such as between "refactor login handler" and "rewrite authentication function."

In practice, TF–IDF retrieval produced more stable results for short diffs (bug fixes or single-file changes), while hybrid retrieval performed better on semantically complex diffs involving refactoring or multi-file updates. The hybrid mode thus acts as a semantic memory of project history.

4.4.4. Hyperparameters and Efficiency

The FAISS index stored 576 000 vectors (one per commit) in 384 dimensions requiring the native output size of MiniLM to be approximately 1.1 GB of CPU memory. Average retrieval latency was 0.06 s per query, and full RAG based generation averaged 2.3 s per diff, remaining practical for CI/CD pipeline automation [18,19]. All retrieval and scoring components were implemented in Python 3.10 using `scikit-learn`, `sentence transformers`, and `faiss-cpu`. This retrieval setup was integrated into the unified evaluation pipeline described in Section 5.

4.5. Evaluation Metrics

To address RQ1–RQ2, we combined automatic metrics with human evaluation to capture both lexical correspondence and perceived usefulness of generated commit messages.

4.5.1. Automatic Metrics

Four complementary automatic metrics were computed on the test split, each quantifying a different aspect of message quality:

- **BLEU-n** (*Bilingual Evaluation Understudy*): measures n-gram precision between the generated (C) and reference (R) messages, with a brevity penalty to discourage overly short outputs:

$$\text{BLEU-}n = \text{BP} \cdot \exp\left(\sum_{i=1}^{n} w_i \log p_i\right), \quad \text{BP} = \begin{cases} 1, & \text{if } c > r \\ e^{(1-r/c)}, & \text{if } c \le r \end{cases}$$

  where p_i is the i-gram precision, w_i are uniform weights, and c, r are candidate and reference lengths [20].
- **ROUGE-L** (*Recall-Oriented Understudy for Gisting Evaluation*): computes the longest common subsequence (LCS) between C and R, forming an F-measure of sequence overlap:

$$\text{ROUGE-L} = \frac{(1+\beta^2) \cdot \text{LCS}(C,R)}{r + \beta^2 c},$$

  where $\beta = 1.2$ weights recall higher than precision, and c, r are message lengths [21].
- **METEOR**: a recall-oriented metric that incorporates stemming and synonym matching:

$$\text{METEOR} = F_{\text{mean}} \cdot (1 - P_{\text{penalty}}),$$

  where F_{mean} is the harmonic mean of unigram precision and recall, and P_{penalty} penalizes word-order mismatches [22].
- **Adequacy**: measures semantic coverage of informative tokens in the generated message relative to the reference [23]:

$$\text{Adequacy} = \frac{|C^* \cap R^*|}{|R^*|},$$

  where C^* and R^* denote the sets of non stopword tokens in C and R.

BLEU and ROUGE emphasize surface-level lexical overlap, whereas METEOR and Adequacy capture deeper semantic fidelity and tolerance to paraphrasing. Adequacy, in particular, was designed to align more closely with human perceptions of meaning preservation in commit message generation.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025
doi:10.20944/preprints202512.2193.v1

4.5.2. Human Evaluation

Because automatic metrics alone do not capture developer perception [6,13,15], here we present the results of a human study on 100 sampled commits, balanced across six programming languages: Go, Java, PHP, Python, JavaScript, and Ruby. Each commit was rated on the following two measures by three experienced open-source contributors using a 1–5 Likert scale:

- **Adequacy (Meaning Accuracy):** how well the message describes the intent and effect of the code change;
- **Fluency (Naturalness):** grammatical correctness, readability, and alignment with the conventional commit style.

To ensure transparency in ethics, participants were also asked for their consent on whether they allowed acknowledgment of their names in the paper.

Inter-rater agreement was computed using Cohen's κ, and mean scores were reported per model.

This mixed evaluation protocol addresses RQ2 directly by linking human judgments to automatic metrics.

4.6. Summary

Above, the methodology describes an overall process for assessing automatically generated commit message summaries under various paradigms. Moreover, by including standard preprocessing, prompt designing, retrieval augmentation, and evaluation by both automated metrics and human raters, the task ensures methodological consistency and comparability as outlined in our previous work [24]. Empirical findings will be presented in subsequent sections (Sections 5-6) in a form of table and visual summaries, showing relative merits of zero-shot, retrieval, and fine-tuned. This design allows a balanced treatment of both the quantitative measures and the qualitative assessment of the developer. This makes it easier to compare the performance of the models.

5. Experimental Setup and Implementation

5.1. Environment and Hardware Configuration

Experiments were performed in a hybrid setup combining a local workstation for preprocessing and evaluation with a GPU server for model inference and fine-tuning.

The local environment used for dataset cleaning, retrieval indexing, and metric computation, featured:

- **CPU:** Intel® Core™ i9-13900K (24 cores, 32 threads),
- **Memory:** 64 GB DDR5 RAM,
- **Storage:** 2 TB NVMe SSD,
- **Operating System:** Ubuntu 22.04 LTS.

Model inference and fine-tuning were conducted on an NVIDIA A100 GPU node (40 GB VRAM) using the `transformers` and `accelerate` libraries. All experiments were implemented in Python 3.10 within a managed `conda` environment. The main software dependencies included:

- `transformers =4.44.2` model loading and inference,
- `datasets =2.20.0` data handling and splits,
- `sentence-transformers =2.6.1` embedding generation,
- `faiss-cpu =1.8.0` vector similarity search,
- `scikit-learn =1.5.2` preprocessing and re-ranking,
- `evaluate =0.4.3` BLEU, ROUGE, and METEOR computation.

All scripts and evaluation notebooks were version controlled in a private GitHub repository, and environment configurations were stored as `environment.yml` snapshots for full reproducibility.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

5.2. Dataset Splits and Consistency Control

The balanced CommitBench subset (576,342 commits; see Section 3) was divided into the following splits:

- **Training set:** 461,073 commits (80%)
- **Validation set:** 57,634 commits (10%)
- **Test set:** 57,635 commits (10%)

Each repository was assigned to exactly one split to avoid data leakage. File-level hashing verified that no diff appeared in multiple partitions. For the fine-tuned Qwen-Commit model, the training subset was used to adapt the decoder head and final embedding projection, while the zero-shot and RAG models were evaluated directly on the test set.

To preserve fairness, all models received the same inputs (diffs truncated to 512 tokens) and identical preprocessing rules for whitespace and diff headers.

5.3. Inference Configuration and Decoding Parameters

Each model was executed under deterministic decoding to ensure reproducibility. Table 2 lists the main hyperparameters applied to all model variants.

Table 2. Model inference and decoding parameters.

Parameter	ChatGPT-4.1-mini	DeepSeek-RAG	Qwen-Commit
Max tokens (output)	40	40	40
Temperature	0.2	0.2	0.3
Top-p (nucleus)	0.9	0.9	0.95
Top-k	40	40	50
Stop sequence	`"\n"`	`"\n"`	`"\n"`
Context window	4096	8192 (with retrieval)	4096
Batch size	16	8	8

All outputs were normalized to lowercase and stripped of punctuation before automatic metric computation. To prevent prompt leakage across models, each inference pipeline reloaded a fresh session per 1000 samples. For DeepSeek-RAG, retrieval was computed once and embedded using FAISS IDs, resulting in a 35% reduction in generation latency compared to on-demand retrieval.

5.4. Human Evaluation Protocol

To complement the automatic evaluation, a human-centered survey was conducted with 12 volunteer developers recruited from GitHub and graduate research groups. A total of 100 commits were randomly selected from the test set, balanced across six programming languages and commit types. For each commit, raters were shown:

1. the original `git diff`,
2. the human-written commit message (hidden until rating completion),
3. three anonymized model outputs (order randomized),
4. and two 1–5 scale grading fields for **adequacy**, and **fluency**.

Ratings were collected using a customized web-based survey. Each rater rated between 25 and 40 samples to ensure that there was overlap to compute interrater reliability. Values of Cohen's κ above 0.72 indicated substantial agreement, thus confirming the reliability of human judgments.

The correlation between human and automatic metrics was analyzed to address **RQ2**. In addition, qualitative error inspection was performed for the top and bottom 10% of commits according to human ratings, revealing recurring failure patterns such as misinterpreted diff context and overgeneralization.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025 doi:10.20944/preprints202512.2193.v1

5.5. Reproducibility and Open Research Practices

To ensure transparency and replicability, all preprocessing scripts, evaluation templates, and survey materials were deposited in a public repository (to be released upon publication). The repository includes:

- the complete preprocessing and data filtering pipeline with versioned thresholds,
- Python scripts for embedding generation, FAISS indexing, and re-ranking,
- evaluation notebooks computing BLEU, ROUGE-L, METEOR, and Adequacy metrics,
- anonymized developer survey data with per-sample Adequacy and Fluency scores,
- Google Apps Script used to automatically generate balanced Google Forms (12 samples per form, two per language),
- and a detailed `README.md` explaining installation, environment setup, and evaluation execution.

All random seeds were fixed (seed=42) for dataset shuffling, retrieval sampling and model generation. Human evaluation forms followed a standard 5-point Likert scale for semantic adequacy and linguistic fluency alike. Developers were also asked for consent as to whether they allow their names to be mentioned in the acknowledgments of the paper, following open research ethics.

This setup ensures that our comparison among ChatGPT, DeepSeek-RAG, and Qwen-Commit is fully reproducible, transparent, and ethically aligned, with a stable foundation for the quantitative and qualitative results in Section 6.

6. Results and Analysis

6.1. Overview

This section presents the experimental results of the three LLM paradigms: ChatGPT, a zero-shot model; DeepSeek RAG, a retrieval-augmented model; and Qwen, a fine-tuned open model. We analyze the automatic evaluation metrics and human ratings, followed by per-language and qualitative insights. The results answer the two research questions (RQ1–RQ2) presented in Section 1.

6.2. Automatic Evaluation Results (RQ1)

Table 3 summarizes the quantitative results across BLEU, ROUGE-L, METEOR, and Adequacy. Scores are averaged over the full test set (57,635 commits), with standard deviations computed over three runs.

Table 3. Automatic evaluation metrics averaged across the test set (higher is better).

Model	BLEU-2	ROUGE-L	METEOR	Adequacy
ChatGPT (zero-shot)	0.184	0.326	0.291	0.553
DeepSeek-RAG (hybrid retrieval)	0.216	0.352	0.317	0.612
Qwen-Commit (fine-tuned)	**0.244**	**0.384**	**0.333**	**0.635**

This shows that the fine-tuned **Qwen-Commit** model outperforms the zero-shot and retrieval-augmented models on all metrics. Retrieval-augmented generation (**DeepSeek-RAG**) showed remarkable improvements over the zero-shot baseline, with a maximum improvement on Adequacy (+5.9 pp) and ROUGE-L (+2.6 pp). This supports our hypothesis that contextual retrieval helps adapt general-purpose models to project-specific conventions without retraining.

Summing up, RQ1 is answered positively: LLMs are very different in the quality of generated commit messages. Model specialisation, either fine-tuning or RAG, brings consistent gains over zero-shot inference.

6.3. Correlation Between Automatic Metrics and Human Ratings

Our further analysis of the human evaluation shows that developer judgments align much more strongly with the Adequacy metric than with similarity based metrics like BLEU, ROUGE-L, or METEOR. Though BLEU and ROUGE reward lexical overlap, human raters consistently valued

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025 doi:10.20944/preprints202512.2193.v1

messages that captured the meaning and intent of the code change, even when the wording was different from that of the reference. The measure of Adequacy captures semantic coverage rather than surface similarity, therefore, reflects human perception far more accurately. This explains why the models with higher scores on Adequacy were preferred more often in the human study.

6.4. Human Evaluation Results (RQ2)

Human evaluation was done using the balanced survey forms described in Section 4. Each developer rated 12 samples per form (two from each programming language: Go, Java, PHP, Python, JavaScript, and Ruby), ensuring a uniform distribution across languages.

For every generated commit message, participants provided two distinct scores **Adequacy (Meaning Accuracy)** and **Fluency (Naturalness)** on a 5-point Likert scale ranging from 1 (Very Poor) to 5 (Excellent). Developers were also asked for consent regarding the mention of their names in the acknowledgment section, in line with ethical research practices.

Table 4 reports the mean human ratings for the three qualitative criteria, clarity, informativeness, and usefulness, aggregated from the Adequacy and Fluency dimensions of 100 manually assessed samples. Each sample was evaluated by three annotators, and final scores were averaged to reduce subjectivity.

Table 4. Human evaluation of message quality (Likert scale 1–5).

Model	Adequacy	Fluency	Overall Mean
ChatGPT (zero-shot)	3.92	3.61	3.77
DeepSeek-RAG (hybrid)	4.24	3.97	4.11
Qwen-Commit (fine-tuned)	**4.45**	**4.19**	**4.32**

Fine-tuning again produced the most human preferred messages, often described as "concise yet descriptive." Retrieval augmentation improved informativeness and contextual grounding, particularly for configuration and refactoring commits, but sometimes introduced verbosity. Annotators noted that the hybrid DeepSeek-RAG model tended to capture project specific phrasing better than zero-shot ChatGPT, though with less stylistic precision than Qwen-Commit.

6.5. Qualitative Error Analysis

To better understand model behavior, we manually inspected 50 low-scoring samples (bottom 10% by adequacy). Three recurring error categories emerged:

1. **Shallow summaries:** models describe the change superficially (e.g., "update tests") without rationale.
2. **Misinterpreted context:** RAG occasionally retrieves semantically unrelated examples, leading to incorrect intent inference.
3. **Overgeneralization:** fine-tuned models sometimes omit filenames or key entities to maintain brevity, reducing specificity.

Interestingly, some "errors" were rated positively by human evaluators, suggesting that concise summaries may still be acceptable if semantically aligned—a nuance not captured by lexical metrics.

7. Discussion

7.1. Interpreting the Results

Our results clearly demonstrate that both fine-tuned and augmented models are substantially better than zero-shot prompt-based approaches for message generation. However, the lesson learned extends beyond the results, as message quality can be achieved not only by scaling-up model sizes but also by the appropriateness of model contextual alignment with the repository's evolution.

The use of retrieval to assist generation has the effect of improving upon semantic drift—a problem with general-purpose models which tends to lack grounding, often just mimicking real world

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025 doi:10.20944/preprints202512.2193.v1

project specifics like naming and structure. While these models are automatically aligned, it does incur additional costs and privacy worries.

It is particularly noticeable that a close relationship between human ratings and the Adequacy score suggests a significant approach to this problem. As surface based Adequacy measures, BLEU and ROUGE score alignment primarily concentrates on word overlaps, unlike Adequacy. These aspects make Adequacy a promising approach for benchmark research on commit message generation.

It appears that the following strategy could be adopted: The application of retrieval-augmented generation could be favored when the goal turns to adaptability rather than precision, whereas fine-tuning might be the best choice when

7.2. Practical Deployment Considerations

Bringing commit message generation into real development workflows requires attention to several practical factors:

- **Latency and performance.** Retrieval plus generation averages under 2.5 s per commit (as discussed in Section 4.4). This is acceptable for continuous integration, though large repositories may benefit from batching or async execution.
- **Privacy and compliance.** Fine-tuning on local infrastructure ensures sensitive code never leaves the organization an essential feature for teams in regulated industries.
- **Customization.** Teams can scope retrieval indexes to specific repositories or branches, offering contextual adaptation without requiring full retraining.
- **Toolchain compatibility.** Integration with Git hooks, IDE extensions (e.g., VS Code, IntelliJ), and REST APIs makes it easy for developers to adopt the system without altering their workflows.

Beyond standalone use, this framework can be embedded into **DevOps pipelines** as part of continuous documentation practices. For example, the system can be triggered via local Git hooks or during CI/CD stages (e.g., GitHub Actions, GitLab CI). Generated messages can then be reviewed and edited by developers before merging, preserving a human in the loop process.

This approach aligns naturally with DevOps goals of traceability, automation, and reproducibility enabling lightweight, AI assisted documentation that respects human oversight.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025
doi:10.20944/preprints202512.2193.v1

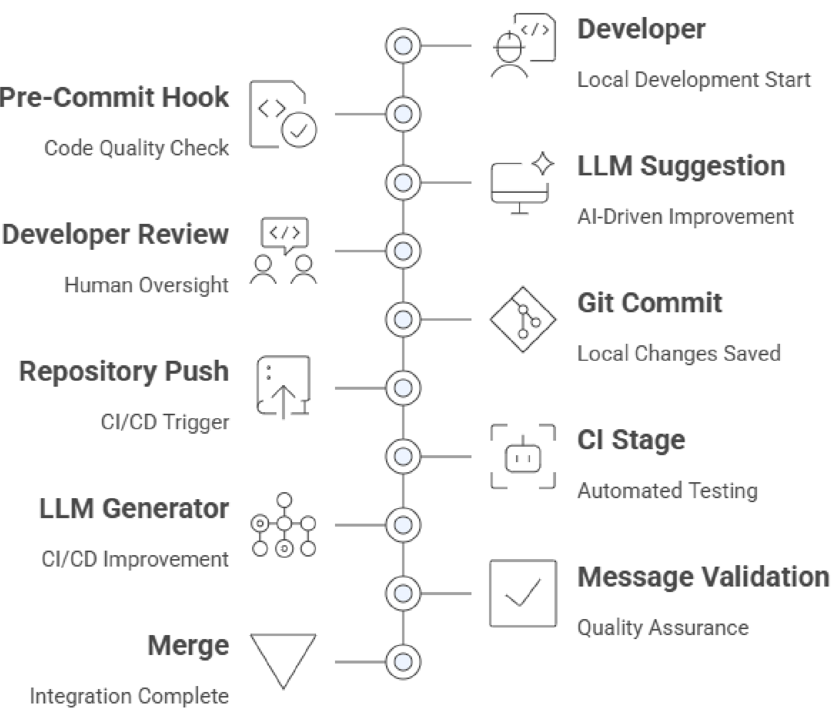

Figure 10. Practical integration of the proposed framework into software workflows. The model operates at two levels: (1) as a local pre-commit hook offering real-time suggestions, and (2) as a CI/CD component enhancing or validating messages before merge. This hybrid setup enables automation with built-in human review.

7.3. Ethical and Technical Considerations

Automation of commit messages raises other issues, apart from the technical correctness of the messages themselves. On one hand, LLMs can be useful in making the messages standardized, thereby making the task easier. On the other hand, there is an indirect transfer of the responsibility of message authorship from the programmer

As a matter of transparency and accountabilities, it is recommended that system deployments be accompanied by aspects of tagging auto-generated messages and record-keeping of the model version used. Developers should still have a scope for overriding and altering outputs.

In practice, mixed initiative systems, where AI offers suggestions but humans remain in control are best aligned with responsible AI adoption in software engineering [6,25]. Continuous feedback loops and editorial control foster both quality and trust in these hybrid workflows.

7.4. Summary of Key Findings

The main outcomes of this study can be summarized as follows:

- **RQ1:** Fine-tuned and retrieval-augmented LLMs clearly outperform zero-shot prompting in commit message generation.
- **RQ2:** The Adequacy metric shows the strongest correlation with human judgment, suggesting semantic metrics are better suited than lexical ones.
- The models exhibit consistent performance across multiple programming languages, suggesting generalizability.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025 doi:10.20944/preprints202512.2193.v1

- Fine-tuned models offer high precision, while retrieval-augmented models offer flexibility representing complementary strengths.
- The proposed framework is lightweight, fast, and easily integrated into real world workflows without disrupting developer autonomy.

Together, these findings put LLM based commit message generation as a viable, human centered enhancement to modern software engineering practices will bridge academic research with actionable tools for industry.

8. Limitations and Future Work

Although this study provides very important information, a number of limitations exist in order to improve in future.

8.1. Internal Validity

Code differences alone are used for evaluation, without relying on other information such as issue names, comments in PRs, and CI status. In reality, commit messages in real-world projects do not always ignore this extra information. One possible future research direction is in **multi-modal conditioning**.

8.2. External Validity

Our experimental setup utilizes the **CommitBench** dataset [7,8]. Although this dataset supports six different languages, it is probable that it will not cover all repositories. Areas such as embedded systems or infrustructure code repositories are less considered. Such a study can be reproduced in other repositories related to propriety or a specific domain.

8.3. Construct Validity

Human evaluation entailed a small number of samples evaluated by professional developers. Although inter-rater reliability tests were quite good, a larger population of people would have produced more generalizable results. Additionally, other variables such as *clarity* and *usefulness* can be subdivided into more specific rubrics.

8.4. Model and Retrieval Constraints

The performance of RAG heavily relies on the quality of the retrieval task. Incorrect examples can lead to semantic drift, especially in situations with poor lexical overlap. To improve this, future research can focus more on exploring **neural re-ranking** [6,25] or hybrid selectors that balance diversity and precision. Additionally, while smaller open models offer privacy benefits, they may struggle with complex reasoning compared to larger proprietary systems.

8.5. Opportunities for Future Research

Several promising directions emerge:

- **Adaptive learning.** Developer edits can be incorporated into learning loops that adjust retrieval weights or model parameters over time.
- **Explainability.** Build systems where generated message tokens can be traced back to a particular diff segment.
- **Cross-task generalization.** Apply this comparative study paradigm to other relevant tasks such as generating a change log, linking issues, and code summarization.
- **Auditing and accountability.** Include support for auditing machine-generated commit messages with added metadata.
- **Field validation.** Perform longitudinal studies to assess effects on developer productivity, document quality, and satisfaction in a real-world setting.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025 doi:10.20944/preprints202512.2193.v1

Overall, this study supports the development of **human centered automation** for software documentation. Combining technical rigor with responsible deployment practices will enable future systems to be transparent, adaptive, and grounded in real developer needs.

9. Conclusion

In this paper, a comparative analysis of different large language model approaches for commit message generation will be presented using a controlled setting in the form of a dataset called **Commit-Bench**. For this analysis, I chose to compare outcomes based on not one but three different paradigms: ChatGPT, DeepSeek, and Qwen.

Our findings indicate a marked superiority of both fine-tuned and retrieval-augmented models over zero-shot methods in providing brief, accurate, and relevant commit message descriptions (RQ1). Moreover, among all evaluation metrics, the most relevant one to a subjective assessment emphasizing semantic relevance over superficial similarity is indicated by the **Adequacy** metric (RQ2). Aside from this, this paper proposes a reproducible framework involving data processing, information retrieval, and reproducible evaluation. Moreover, this research will show how LLMs can be applied in a realistic setting in developer environments for privacy, transparency, and efficiency. In conclusion, commit message generation is no longer a topic of speculations. The tool is now a **practical, measurable, and morally deployable** implementation technique in modern software development. The future will most probably bring developments in systems capable of continuous learning, adjusting to different contexts, and empowering coders without taking control. "The future of AI-assisted development is not about replacing developers," says Sudesh, "but providing them with assistance in the form of tools which write side by side with them, not instead of them."

The next phase of AI-assisted development is not about replacing developers, it's about giving them tools that write with them, not for them.

Author Contributions: Conceptualization, M. Trigui and F. Mallouli; methodology, M. Trigui, M. Amro and F. Mallouli; software, M. Trigui; validation, M. Trigui and W. G. Al-Khatib; formal analysis, M. Trigui; investigation, M. Trigui; resources, M. Trigui; data curation, M. Trigui; writing—original draft preparation, W. G. Al-Khatib and M. Amro ; writing—review and editing, M. Trigui and W. G. Al-Khatib; visualization, M. Trigui and M. Amro; supervision, W. G. Al-Khatib. All authors have read and agreed to the published version of the manuscript.

Funding: This research received internal funding from KFUPM.

Data Availability Statement: The benchmark dataset (CommitBench) used in this study is available from public sources described in [1,4,7]. Processed splits, prompt templates, and evaluation scripts can be shared upon reasonable request.

Acknowledgments: The authors would like to acknowledge the funding support provided by the Interdisciplinary Research Center for Intelligent Secure Systems (IRC-ISS) at King Fahd University of Petroleum & Minerals under Project No. INSS2522. The authors also acknowledge the use of **GPT-4o** for enhancing the language and readability of this manuscript, and **Napkin AI** for the generation of select figures.

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

References

1. Zhang, Y.; Qiu, Z.; Stol, K.-J.; Zhu, W.; Zhu, J.; Tian, Y.; Liu, H. Automatic commit message generation: A critical review and directions for future work. *IEEE Trans. Softw. Eng.* **2024**, *50*, 816–835.
2. Li, J.; Ahmed, I. Commit message matters: Investigating impact and evolution of commit message quality. In Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia, 15–16 May 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 806–817.
3. Tian, Y.; Zhang, Y.; Stol, K.-J.; Jiang, L.; Liu, H. What makes a good commit message? In Proceedings of the 44th International Conference on Software Engineering (ICSE '22), Pittsburgh, PA, USA, 21–29 May 2022; ACM: New York, NY, USA, 2022; pp. 2389–2401. https://doi.org/10.1145/3510003.3510205
4. Xue, P.; Wu, L.; Yu, Z.; Jin, Z.; Yang, Z.; Li, X.; Yang, Z.; Tan, Y. Automated commit message generation with large language models: An empirical study and beyond. *IEEE Trans. Softw. Eng.* **2024**, *50*, 3208–3224.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025 doi:10.20944/preprints202512.2193.v1

5. Zhang, Q.; Fang, C.; Xie, Y.; Zhang, Y.; Yang, Y.; Sun, W.; Yu, S.; Chen, Z. A survey on large language models for software engineering. *arXiv* **2023**, arXiv:2312.15223.
6. Hou, X.; Zhao, Y.; Liu, Y.; Yang, Z.; Wang, K.; Li, L.; Luo, X.; Lo D.; Grundy, J.; Wang, H. Large language models for software engineering: A systematic literature review. *ACM Trans. Softw. Eng. Methodol.* **2024**, 33, 1–79.
7. Schall, M.; Czinczoll, T.; De Melo, G. Commitbench: A benchmark for commit message generation. In Proceedings of the 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), Rovaniemi, Finland, 12–15 March 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 728–739.
8. Kosyanenko, I.A.; Bolbakov, R.G. Dataset collection for automatic generation of commit messages. *Russ. Technol. J.* **2025**, *13*, 7–17.
9. Jiang, S.; Armaly, A.; McMillan, C. Automatically generating commit messages from diffs using neural machine translation. In Proceedings of the 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), Urbana, IL, USA, 30 October–3 November 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 135–146.
10. 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, Montpellier, France, 3–7 September 2018; pp. 373–384.
11. Nie, L.Y.; Gao, C.; Zhong, Z.; Lam, W.; Liu, Y.; Xu, Z. Coregen: Contextualized code representation learning for commit message generation. *Neurocomputing* **2021**, *459*, 97–107.
12. He, Y.; Wang, L.; Wang, K.; Zhang, Y.; Zhang, H.; Li, Z. Come: Commit message generation with modification embedding. In Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, Seattle, WA, USA, 17–21 July 2023; pp. 792–803.
13. Lopes, C.V.; Klotzman, V.I.; Ma, I.; Ahmed, I. Commit messages in the age of large language models. *arXiv* **2024**, arXiv:2401.17622.
14. 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.
15. Huang, Z.; Huang, Y.; Chen, X.; Zhou, X.; Yang, C.; Zheng, Z. An empirical study on learning-based techniques for explicit and implicit commit messages generation. In Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, Sacramento, CA, USA, 27 October–1 November 2024; pp. 544–555.
16. Zhang, L.; Zhao, J.; Wang, C.; Liang, P. Using large language models for commit message generation: A preliminary study. In Proceedings of the 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), Rovaniemi, Finland, 12–15 March 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 126–130.
17. Bogomolov, E.; Eliseeva, A.; Galimzyanov, T.; Glukhov, E.; Shapkin, A.; Tigina, M.; Golubev, Y.; Kovrigin, A.; van Deursen, A.; Izadi, M.; et al. Long code arena: A set of benchmarks for long-context code models. *arXiv* **2024**, arXiv:2406.11612.
18. Joshi, S. A review of generative AI and DevOps pipelines: CI/CD, agentic automation, MLOps integration, and large language models. *Int. J. Innov. Res. Comput. Sci. Technol.* **2025**, *13*, 1–14.
19. Allam, H. Intelligent automation: Leveraging LLMs in DevOps toolchains. *Int. J. AI Bigdata Comput. Manag. Stud.* **2024**, *5*, 81–94.
20. Papineni, K.; Roukos, S.; Ward, T.; Zhu, W.-J. BLEU: A method for automatic evaluation of machine translation. In Proceedings of the ACL, 2002.
21. Lin, C.-Y. ROUGE: A Package for Automatic Evaluation of Summaries. In Proceedings of Text Summarization Branches Out: ACL Workshop, 2004.
22. Banerjee, S.; Lavie, A. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In Proceedings of the ACL Workshop, 2005.
23. Xu, Z.; others. Commit message generation for source code changes. In Proceedings of the ACL, 2019.
24. Trigui, M.M.; Al-Khatib, W.G. LLMs for Commit Messages: A Survey and an Agent-Based Evaluation Protocol on CommitBench. *Computers* **2025**, 14(10), 427. doi:10.3390/computers14100427.
25. Gao, C.; Hu, X.; Gao, S.; Xia, X.; Jin, Z. The current challenges of software engineering in the era of large language models. *ACM Trans. Softw. Eng. Methodol.* **2025**, *34*, 1–30.
26. Beining, Y.; Alassane, S.; Fraysse, G.; Cherrared, S. Generating commit messages for configuration files in 5G network deployment using LLMs. In Proceedings of the 2024 20th International Conference on Network

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 December 2025 doi:10.20944/preprints202512.2193.v1

and Service Management (CNSM), Prague, Czech Republic, 28–31 October 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 1–7.
27. Pandya, K. Automated Software Compliance Using Smart Contracts and Large Language Models in Continuous Integration and Continuous Deployment with DevSecOps. Master's Thesis, Arizona State University, Tempe, AZ, USA, 2024.
28. Kruger, J. *Embracing DevOps Release Management: Strategies and Tools to Accelerate Continuous Delivery and Ensure Quality Software Deployment*; Packt Publishing Ltd.: Birmingham, UK, 2024.
29. Palakodeti, V.K.; Heydarnoori, A. Automated generation of commit messages in software repositories. *arXiv* **2025**, arXiv:2504.12998.
30. Bektas, A. Large Language Models in Software Engineering: A Critical Review of Evaluation Strategies. Master's Thesis, Freie Universität Berlin, Berlin, Germany, 2024.
31. Liu, Y.; Chen, J.; Bi, T.; Grundy, J.; Wang, Y.; Yu, J.; Chen, T.; Tang, Y.; Zheng, Z. An empirical study on low-code programming using traditional vs large language model support. *arXiv* **2024**, arXiv:2402.01156.
32. Wang, S.-K.; Ma, S.-P.; Lai, G.-H.; Chao, C.-H. ChatOps for microservice systems: A low-code approach using service composition and large language models. *Future Gener. Comput. Syst.* **2024**, *161*, 518–530.
33. Zhao, Y.; Luo, Z.; Tian, Y.; Lin, H.; Yan, W.; Li, A.; Ma, J. Codejudge-eval: Can large language models be good judges in code understanding? *arXiv* **2024**, arXiv:2408.10718.
34. Cao, J.; Chan, Y.-K.; Ling, Z.; Wang, W.; Li, S.; Liu, M.; Wang, C.; Yu, B.; He, P.; Wang, S.; et al. How should I build a benchmark? *arXiv* **2025**, arXiv:2501.10711.
35. Ragothaman, H.; Udayakumar, S.K. Optimizing service deployments with NLP-based infrastructure code generation—An automation framework. In Proceedings of the 2024 IEEE 2nd International Conference on Electrical Engineering, Computer and Information Technology (ICEECIT), Jember, Indonesia, 22–23 November 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 216–221.
36. Gandhi, A.; De S.; Chechik, M.P.; Pandit, V.; Kiehn, M.; Chee, M.C.; Bedasso, Y. Automated codebase reconciliation using large language models. In Proceedings of the 2025 IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering (Forge), Ottawa, ON, Canada, 27–28 April 2025; IEEE: Piscataway, NJ, USA, 2025; pp. 1–11.
37. Don, R.G.G. Comparative Research on Code Vulnerability Detection: Open-Source vs. Proprietary Large Language Models and Lstm Neural Network. Master's Thesis, Unitec Institute of Technology, Auckland, New Zealand, 2024.
38. Sultana, S.; Afreen, S.; Eisty, N.U. Code vulnerability detection: A comparative analysis of emerging large language models. *arXiv* **2024**, arXiv:2409.10490.
39. Coban, S.; Mattukat, A.; Slupczynski, A. Full-Scale Software Engineering. Master's Thesis, RWTH Aachen University, Aachen, Germany, 2024.
40. Zhang, X.; Muralee, S.; Cherupattamoolayil, S.; Machiry, A. On the effectiveness of large language models for GitHub workflows. In Proceedings of the 19th International Conference on Availability, Reliability and Security, Vienna, Austria, 30 July–2 August 2024; pp. 1–14.
41. Cihan, U.; Haratian, V.; İçöz, A.; Gül, M.K.; Devran, O.; Bayendur, E.F.; Uçar, B.M.; Tüzün, E. Automated code review in practice. *arXiv* **2024**, arXiv:2412.18531.
42. Jaju, I. Maximizing DevOps Scalability in Complex Software Systems. Master's Thesis, Uppsala University, Department of Information Technology, Uppsala, Sweden, 2023; p. 57.

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.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.