DOI: 10.5281/zenodo.21026831

A Local Human-in-the-Loop Assistant for AI-Assisted SVN Commit-Message Generation

Marco Righi

09 July 2026

Consiglio Nazionale
delle **Ricerche**

National Research Council of Italy

Corresponding author:
Marco Righi
marco.righi@cnr.it

Authors, ORCID iD, email, and institutional affiliation:
Marco Righi: https://orcid.org/0000-0002-1448-0960; marco.righi@cnr.it; National Research Council of Italy

Contents

Keywords

Subversion; SVN; commit-message generation; local Large Language Models; Ollama; human-in-the-loop software engineering; reproducible command-line tools

Abstract

This technical report presents `svn_ai_msg`, an open-source command-line assistant for generating commit-message candidates from Subversion working copies. The tool combines deterministic repository inspection, lightweight rule-based processing, and local Large Language Model inference through Ollama. The proposed workflow extracts information from `svn status` and selected diff content, generates multiple commit-message candidates, and preserves user control over the final commit operation. The implementation is designed for local execution in order to support source-code confidentiality, reproducibility, and integration with lightweight command-line environments. A preliminary functional evaluation on representative SVN changes indicates that the proposed workflow can generate useful commit-message candidates while maintaining a human-in-the-loop interaction model. The source code is publicly available and archived with a persistent DOI.

1 Introduction

1.1 Problem Definition

Version-control systems preserve the technical history of a software project, but the usefulness of this history depends strongly on the quality of the associated commit messages. In Subversion-based workflows, commit descriptions are typically written manually after inspecting the local working copy status and the corresponding textual diffs. This manual step is repetitive, context-dependent, and prone to under-specification: developers may write messages that are too generic, too long, inconsistent across commits, or insufficiently connected to the actual code changes. The problem addressed in this report is the semi-automatic generation of concise, informative, and technically grounded commit messages for SVN working copies. The generated messages should reflect the actual local changes, remain short enough to be used directly as commit summaries, and avoid introducing information that is not supported by the observed repository state. At the same time, the generation process should fit within lightweight command-line workflows and should not require the source code or diffs to be sent to external cloud services. This requirement is particularly relevant for legacy or institutional software projects where SVN is still used, where traceability and auditability are important, and where local execution is preferable for privacy, reproducibility, and operational control. The target problem is therefore not general-purpose code summarisation, but the narrower and operationally constrained task of producing reliable commit-message candidates from local SVN metadata and diffs.

1.2 Objectives and Scope

The objective of **svn_ai_msg** is to support developers in writing concise and informative commit messages for Subversion working copies. The tool is designed to extract a compact representation of the local repository state, including changed files and selected diff content, and to use this information to generate candidate commit messages through a locally served Large Language Model.

The main design goals are the following: to reduce the manual effort required to inspect changes and formulate a commit summary; to produce multiple candidate messages with different levels of detail; to keep the generation process reproducible and configurable; and to preserve source code confidentiality by relying on a local inference endpoint rather than an external cloud service. The tool also aims to remain compatible with lightweight command-line workflows, requiring only standard utilities such as Bash, Subversion, Python, and a local Ollama instance.

The scope of the software is intentionally narrow. It does not perform automatic code review, semantic verification of program correctness, or autonomous commits without user supervision. Instead, it acts as a commit-message assistant: it analyses local SVN metadata and diffs, generates candidate messages, applies conservative post-processing, and allows the user to select the final message. This restricted scope is intended to make the tool practical, auditable, and suitable for institutional or legacy software environments where SVN is still in use.

1.3 Main Contributions

The main contributions of this work are the following.

DOI: 10.5281/zenodo.21026831

- A local-first commit-message generation workflow for Subversion working copies, designed to avoid the transmission of source-code diffs to external cloud services.
- A hybrid generation strategy that combines deterministic repository inspection, rule-based heuristics, and local Large Language Model inference.
- A context-restricted prompting approach that attempts to reduce hallucinations by explicitly limiting the information that can be used during message generation.
- A multi-candidate generation workflow that produces several alternative commit-message formulations and preserves human supervision during the final commit process.
- A reproducible open-source implementation released under the MIT License and archived with a persistent DOI.

1.4 State of the Art

Automatic commit-message generation is an established topic in software engineering research. Early approaches formulated the problem as a sequence-to-sequence or neural machine translation task, where source code diffs are transformed into natural-language descriptions [**?**]. Subsequent work introduced richer representations of source-code changes, including structural and semantic information extracted from the modified code [**?**]. Pre-trained programming-language models were later applied to the same task, as in CommitBERT, which uses code modifications and commit messages to train models for automatic message generation across multiple programming languages [**?**].

A second line of research has focused on improving the contextual grounding of generated messages. Retrieval-augmented approaches such as RACE retrieve similar commits and use them as exemplars to guide message generation [**?**]. Other work has investigated the use of project history, shifting part of the problem from pure message generation to history-aware completion [**?**]. More recent approaches, such as COMET, represent code changes through delta-graph context representations in order to capture more information than a plain textual diff [**?**].

Large Language Models have further changed the landscape of commit message generation. Recent studies evaluate both closed and open LLMs for this task, showing that LLM-based approaches can generate useful commit descriptions but also require careful prompting, context selection, and evaluation beyond automatic metrics [**?**, **?**]. Privacy and deployment constraints have also motivated local and open-source solutions, as illustrated by OMEGA, which investigates whether smaller open-source models can compete with proprietary LLMs in commit-message generation [**?**].

In parallel with the research literature, several practical tools have appeared in developer workflows. Most of them target Git repositories and operate as command-line tools, hooks, IDE extensions, or automation actions. Examples include `aicommits`, `gptcommit`, and `OpenCommit`, which generate commit messages from Git changes using external or configurable AI backends [**?**,**?**,**?**]. Support for Subversion is less common but not absent: for example, AI Commits provides an IDE-based workflow and explicitly supports both Git and Subversion as version-control systems [**?**].

The contribution of **svn_ai_msg** is positioned in this narrower space. Unlike most AI-assisted commit-message tools, which are primarily designed around Git repositories and Git-specific workflows, **svn_ai_msg** operates directly on Subversion working copies, supports fully local execution through Ollama, and combines rule-based filtering with LLM-based candidate generation.

DOI: 10.5281/zenodo.21026831

Rather than providing a general IDE plugin or a Git-oriented assistant, it offers a lightweight Bash implementation focused on SVN working copies. It extracts local repository status and diff information, queries a locally served LLM through Ollama, and returns multiple candidate commit messages while preserving user control over the final commit [**?**]. This local-first design is relevant when source-code confidentiality, command-line integration, auditability, and legacy SVN workflows are operational requirements.

1.5 Existing Methods

Existing methods for improving commit-message quality can be grouped into four main categories. The first and most common approach is manual commit-message writing, where the developer inspects the modified files, reads the textual diff, and writes a concise summary of the change. This approach gives maximum control to the developer, but its quality depends on time, discipline, and consistency. In practice, commit messages may become too generic, incomplete, or poorly aligned with the actual code changes.

A second approach is based on message conventions and lightweight standards. Specifications such as Conventional Commits define a structured format for commit messages, typically including a type, an optional scope, and a short description of the change [**?**]. These conventions improve readability and can support downstream automation, such as changelog generation or semantic versioning. However, they do not solve the problem of deriving the message content from the actual code diff; they only constrain the format of the final message.

A third category consists of AI-assisted tools integrated into Git workflows. Several command-line tools and hooks generate commit messages from staged Git changes, including `aicommits`, `gptcommit`, and `OpenCommit` [**?**, **?**, **?**]. These tools demonstrate the practical value of using Large Language Models for commit-message generation, but they are primarily designed around Git concepts such as staged changes, Git hooks, and distributed repository workflows.

A fourth category includes IDE or graphical-client integrations. These tools generate commit messages inside a development environment and may support multiple version-control systems. For example, AI Commits provides an IDE-based workflow and supports both Git and Subversion [**?**]. Such integrations are convenient for users working inside supported IDEs, but they are less suitable for lightweight terminal-based workflows, scripting, or environments where the developer does not want the commit-message assistant to depend on a specific editor.

The method implemented by **svn_ai_msg** follows a narrower and more operationally constrained path. It keeps the interaction at the command-line level, targets SVN working copies directly, extracts local `svn status` and `svn diff` information, and queries a locally served LLM through Ollama [**?**]. The tool therefore combines elements of AI-assisted commit-message generation with a local-first, SVN-oriented, and human-in-the-loop workflow. This positioning distinguishes it from both generic commit-message conventions and Git-centred AI commit tools.

1.6 Limitations and Open Issues

Despite the availability of LLM-based commit-message generators, several practical limitations remain open. The first limitation concerns output quality. Producing a useful

DOI: 10.5281/zenodo.21026831

commit message is not equivalent to producing a fluent sentence: the message must be concise, technically grounded in the actual diff, and sufficiently specific to support later inspection of the project history. In practice, LLM-generated messages may become too generic, may overemphasise secondary changes, or may introduce concepts that are not explicitly supported by the observed repository state.

A second limitation concerns reproducibility. Commit-message generation depends on the selected model, prompt structure, sampling parameters, context length, and post-processing rules. Even when the same source-code changes are analysed, different models or parameter settings may produce different messages. For this reason, a practical tool should expose the main generation parameters and should keep the prompt and post-processing pipeline simple enough to be inspected and reproduced.

A third limitation is related to context selection. Source-code diffs can be long, noisy, or dominated by mechanical changes. Passing the entire diff to the model is often inefficient and may exceed the available context window, especially when using small or medium-size local models. Conversely, aggressive truncation may remove the most informative part of the change. This creates a trade-off between computational cost, available context, and message specificity.

A fourth limitation concerns local deployment. Running the generation pipeline on local hardware preserves source-code confidentiality, but it also introduces performance constraints. On mobile workstation-class hardware, such as a Linux Manjaro system equipped with an NVIDIA TU117GLM Quadro T2000 Mobile / Max-Q GPU, local inference is feasible but can be slow when the model, context size, or number of generated candidates increases. As a consequence, the tool must balance message quality against latency, memory usage, and operational responsiveness.

A final open issue is human oversight. Commit messages affect the long-term auditability of a software repository and should not be accepted blindly. The generation system should therefore be understood as an assistant rather than an autonomous committer. The user must remain in control of the final message, with the possibility of selecting, discarding, or manually editing the generated candidates. This human-in-the-loop requirement motivates the design of **svn_ai_msg**, which generates multiple candidate messages from local SVN metadata and diffs while preserving manual control over the final commit [**?**].

2 Proposed Solution

This section describes the proposed solution, its implementation assumptions, execution parameters, and expected outputs. The goal is to provide enough detail to allow an independent implementation of the same workflow or an equivalent reimplementation in another scripting language.

2.1 Method

The proposed method is implemented by **svn_ai_msg**, a command-line assistant for generating commit-message candidates from local Subversion working copies. The method combines deterministic repository inspection, lightweight rule-based processing, and local LLM-based natural-language generation.

The workflow starts from an SVN working copy containing uncommitted changes. The

DOI: 10.5281/zenodo.21026831

tool first validates the execution environment by checking the availability of the required command-line programs and by verifying that the selected directory is a valid SVN working copy. It then reads the repository state using `svn status`. Changed paths are grouped according to their SVN status, including added, modified, deleted, and replaced files. Unversioned paths are reported separately because they are not part of the commit until they are explicitly added to SVN.

For versioned changes, the method extracts bounded and compact representations of SVN diff information. Whole-commit candidates are based on `svn diff .`, whereas per-file candidates use `svn diff -- PATH`. The implementation does not read source files directly outside the SVN status and diff commands. This keeps the available input tied to the version-control state and avoids adding extra, uncontrolled context from the working tree.

The extracted SVN context is complemented by generic structural information derived from the observed status and diff output, including file extensions, changed paths, command-line options, function names, configuration variables, and other identifiers exposed by the diff. This processing is intentionally domain-agnostic: the script does not encode project-specific commit heuristics and does not rely on previous model conversation state.

The extracted context is converted into restrictive prompts for a local Large Language Model served through Ollama. The prompts require one-line English imperative messages, preserve exact filenames, and explicitly restrict the model to the SVN data supplied in the prompt. The generated text is then normalised, sanitised, checked for prompt leakage or weak generic output, and replaced by deterministic fallbacks when necessary.

Rather than attempting to predict a single optimal commit message, the proposed workflow is explicitly designed as a candidate-generation system. It produces three LLM-assisted candidates and three deterministic safe candidates. The user remains responsible for selecting, editing, or discarding the final message.

2.2 Implementation Details

The reference implementation is a Bash script designed for Unix-like systems. It requires Bash, Subversion, Python 3, and a local Ollama instance. The default model is `qwen2.5-coder:3b`, and the default Ollama endpoint is `http://localhost:11434/api/generate`. Both the model and the endpoint can be overridden through environment variables.

The main command-line parameters are:

- `-p PATH`: select the target SVN working copy;
- `-n LEVEL`: set the creativity level from 1 to 5;
- `-s` or `--suggestion`: provide optional user guidance for the generated commit message;
- `-d` or `--dry-run`: print the selected commit command without executing it;
- `-D` or `--debug`: print diagnostic information to standard error;
- `--model-lifecycle=stop|keep`: select whether the Ollama model should be explicitly stopped before and after each model call.

The generation process is also controlled by environment variables. The most relevant are `OLLAMA_MODEL`, `OLLAMA_URL`, `OLLAMA_NUM_CTX`, `SVN_AI_DEBUG`, `SVN_AI_MODEL_LIFECYCLE`,

DOI: 10.5281/zenodo.21026831

SVN_AI_DIFF_LINES, SVN_AI_MAX_BLOCK_CHARS, SVN_AI_MAX_FILES, SVN_AI_NUM_PREDICT, SVN_AI_MSG_MAX_
SVN_AI_PERFILE_MSG_MAX_CHARS, SVN_AI_FILE_DESC_MAX_CHARS, SVN_AI_TEMP_MIN, SVN_AI_TEMP_MAX,
SVN_AI_TOP_P_MIN, SVN_AI_TOP_P_MAX, SVN_AI_REPEAT_MIN, SVN_AI_REPEAT_MAX, and SVN_AI_SUGGESTION. These parameters control the model backend, context size, sampling behaviour, maximum diff context, candidate length, per-file description length, debug output, and model lifecycle policy.

A typical execution is:

Listing 1: Typical execution of svn_ai_msg.

```
./svn_ai_msg.sh -p /path/to/svn/working-copy -n 3
```

A dry-run execution with explicit user guidance is:

Listing 2: Dry-run execution with user guidance.

```
./svn_ai_msg.sh -p /path/to/svn/working-copy -n 3 \
--suggestion "Summarise the main bug fix" --dry-run
```

Diagnostic output can be enabled either with the command-line flag `-D` or through the environment variable `SVN_AI_DEBUG`:

Listing 3: Execution with debug output.

```
SVN_AI_DEBUG=1 ./svn_ai_msg.sh -p /path/to/svn/working-copy -n 3 --dry-run
```

The tool is intended for local execution. This design avoids sending source-code diffs to external cloud services and makes the workflow suitable for environments where code confidentiality, reproducibility, and operational control are relevant. Model calls are made with stateless local requests, using raw prompts and zero keep-alive time in the Ollama request configuration.

2.3 Expected Outputs

The expected output is a terminal report containing a file-change summary and six candidate commit messages. The file-change summary lists the detected SVN paths grouped by status, including added, modified, deleted, replaced, and unversioned paths when present. Unversioned paths are reported as not directly committable until they are explicitly added to SVN.

The current implementation generates six candidate messages: `per-file`, `general`, `technical`, `safe-per-file`, `safe-general`, and `safe-technical`. The first three candidates are LLM-assisted: `per-file` uses one model call per changed file, `general` generates a whole-commit summary, and `technical` emphasises implementation details when supported by the SVN data. The remaining three candidates are deterministic safe fallbacks derived from SVN status and diff information.

If the user selects one of the proposed messages, the tool can execute the corresponding `svn commit -m` command. In dry-run mode, the command is printed but not executed. Therefore, the expected artifacts are not additional files, but reproducible terminal outputs, selected commit messages, and, when not in dry-run mode, standard SVN commits recorded in the repository history.

DOI: 10.5281/zenodo.21026831

3 Experimentation

The experimentation reported in this section should be interpreted as a preliminary functional evaluation, not as a large-scale benchmark. The objective was to verify whether `svn_ai_msg` behaves coherently on representative SVN working-copy changes and whether the generated candidate messages are usable by a developer in an interactive command-line workflow.

The current version of the tool does not return a single deterministic commit message. Instead, it generates six alternative candidates: `per-file`, `general`, `technical`, `safe-per-file`, `safe-general`, and `safe-technical`. For this reason, the relevant experimental question is not whether the first generated message exactly matches a reference message. The relevant question is whether at least one of the generated candidates is concise, technically grounded in the observed SVN changes, and suitable for use as a commit message with little or no manual editing.

The evaluation was performed on a limited number of representative cases, using local execution only. The tested environment was a Linux workstation equipped with an NVIDIA TU117GLM Quadro T2000 Mobile / Max-Q GPU with 4 GB of video memory, NVIDIA driver 610.43.02, and CUDA UMD 13.3. This hardware configuration is suitable for lightweight local inference, but it imposes practical constraints on model size, context length, latency, and the number of generated candidates. The results should therefore be considered indicative of feasibility under resource-constrained local conditions, rather than conclusive evidence of general performance.

3.1 Experimental Setup

The experimental setup was based on a local SVN working copy containing controlled changes to shell scripts and related documentation. Each test started from a clean working copy. A specific modification was then introduced, and `svn_ai_msg` was executed before committing the change. The generated file-change summary and the six proposed commit-message candidates were inspected manually.

The tested cases included representative modifications such as adding a new script, modifying command-line help text, improving option handling, fixing shell-variable defaults under `set -u`, adding or modifying dry-run behaviour, adding user guidance for message generation, refactoring helper functions, and changing the logic used to score or filter candidate messages.

For each test, the following information should be recorded:

- the local SVN status before running the tool;
- the type of change introduced in the working copy;
- the model used by the local Ollama endpoint;
- the context length and generation parameters;
- the six generated candidate messages;
- the candidate selected by the user, if any;
- whether the selected candidate required manual editing.

This setup reflects the intended use of the tool. The user remains responsible for selecting, rejecting, or editing the final message. The system is therefore evaluated as an interactive assistant, not as an autonomous commit-message generator.

DOI: 10.5281/zenodo.21026831

3.2 Evaluation Criteria

The generated candidates were assessed qualitatively using four practical criteria.

- **Grounding**: the message should be supported by the observed `svn status` and `svn diff` information.
- **Specificity**: the message should identify the main technical change rather than using a generic formulation such as *Update files*.
- **Conciseness**: the message should be short enough to be used directly as an SVN commit summary.
- **Usefulness**: at least one of the six generated candidates should be acceptable with no editing or only minor editing.

A candidate was considered acceptable when it satisfied all four criteria. A test case was considered successful when at least one of the six generated alternatives was acceptable. This evaluation criterion is consistent with the design of the tool, because the script intentionally presents multiple candidate messages and delegates the final selection to the user.

3.3 Preliminary Results

The preliminary tests showed that the tool can generate useful commit-message candidates for small and medium-sized SVN changes. In particular, the most useful outputs were generally produced when the change affected a clearly identifiable feature, such as command-line help, dry-run execution, user-provided guidance, or fallback handling. In these cases, at least one of the six candidates was usually close to a usable commit message.

The experiments also highlighted some limitations. First, candidate quality depends on the amount and relevance of the diff context passed to the model. If the diff is too long, noisy, or dominated by secondary edits, the generated message may become too generic or may focus on a less important aspect of the change. Second, different candidate-generation modes may produce partially overlapping messages. This is not necessarily a failure, because the modes are intended to provide alternative levels of specificity, but it reduces diversity when the underlying change is small. Third, local execution on limited hardware constrains the practical model size and context length, which can affect both latency and message quality.

The adopted evaluation criterion differs from the exact-match metrics commonly used in commit-message-generation research. Since the system generates multiple alternatives, the practical success criterion is the availability of at least one candidate that is sufficiently grounded, concise, and useful for human selection.

The current evaluation should therefore be interpreted as a feasibility assessment. It indicates that the proposed local SVN workflow is operational and can produce useful candidate commit messages in representative cases. However, a broader evaluation would be required to quantify performance more rigorously. Such an evaluation should include a larger set of controlled SVN changes, repeated runs with fixed and variable generation parameters, timing measurements, and comparison against manually written reference commit messages.

The preliminary test cases can be reported as follows:

DOI: 10.5281/zenodo.21026831

- **T1 – Added script**: the tool should generate at least one candidate describing the purpose of the new script.
- **T2 – Help update**: the tool should generate at least one candidate referring to command-line help or option documentation.
- **T3 – Dry-run change**: the tool should generate at least one candidate identifying dry-run behaviour or commit preview.
- **T4 – User guidance**: the tool should generate at least one candidate identifying suggestion or guidance support.
- **T5 – Refactoring**: the tool should generate at least one candidate describing helper-function refactoring, restoration, or cleanup.

4 Code Availability

The source code of `svn_ai_msg` is publicly available and archived with a persistent software DOI [**?**]. The GitHub repository is used as the development location, while the Zenodo record provides the citable archived software release. The present technical report is associated with the DOI reported in the document metadata.

4.1 Repository and Release

The development repository is available at:

`https://github.com/marcorighi/svn_ai_msg`

The archived software release is available through Zenodo and should be cited as [**?**]. The archived release currently referenced in this report is version `2.0`, with DOI:

`https://doi.org/10.5281/zenodo.20849102`

The repository contains the main Bash implementation, `svn_ai_msg.sh`, together with documentation, citation metadata, and licensing information.

4.2 License and Dependencies

The software is released under the MIT License [**?**]. The implementation is designed for Unix-like systems and requires the following runtime dependencies:

- Bash version 4 or later;
- Subversion version 1.10 or later;
- Python 3;
- Ollama version 0.1.33 or later;
- a local Ollama-compatible language model, with `qwen2.5-coder:3b` used as the default model.

A GPU is not strictly required, but it can reduce inference time. The workflow is intended for local execution. This design avoids sending source-code diffs or repository metadata to external cloud services and is therefore suitable for use cases where code confidentiality, reproducibility, and operational control are relevant.

DOI: 10.5281/zenodo.21026831

The software can be obtained with:

Listing 4: Cloning the development repository.

```
git clone https://github.com/marcorighi/svn_ai_msg.git
cd svn_ai_msg
chmod +x svn_ai_msg.sh
```

The basic system dependencies can be installed according to the package manager of the target operating system. On Debian-like systems, a minimal installation is:

Listing 5: Installing basic system dependencies on Debian-like systems.

```
sudo apt install subversion python3 git
```

After installing Ollama, the default local model used by the script can be pulled with:

Listing 6: Pulling the default local model.

```
ollama pull qwen2.5-coder:3b
```

4.3 Reproducibility Commands

The following commands reproduce the basic execution workflow. First, clone the software repository and ensure that the script is executable:

Listing 7: Preparing the script.

```
git clone https://github.com/marcorighi/svn_ai_msg.git
cd svn_ai_msg
chmod +x svn_ai_msg.sh
```

Then run the tool against an SVN working copy containing local uncommitted changes:

Listing 8: Default execution on an SVN working copy.

```
./svn_ai_msg.sh -p /path/to/svn/working-copy
```

A more explicit run can be obtained by fixing the model, context length, and creativity level:

Listing 9: Execution with explicit model and context configuration.

```
OLLAMA_MODEL=qwen2.5-coder:3b \
OLLAMA_NUM_CTX=2048 \
./svn_ai_msg.sh -p /path/to/svn/working-copy -n 3
```

A dry-run execution can be used to inspect the generated commit command without modifying the repository history:

Listing 10: Dry-run execution.

```
./svn_ai_msg.sh -p /path/to/svn/working-copy -n 3 --dry-run
```

The optional user-guidance mode can be tested with:

Listing 11: Dry-run execution with user guidance.

```
./svn_ai_msg.sh -p /path/to/svn/working-copy -n 3 \
--suggestion "Summarise the main bug fix" \
--dry-run
```


DOI: 10.5281/zenodo.21026831

For diagnostic and reproducibility purposes, debug output can be enabled as follows:

Listing 12: Execution with debug output.

```
SVN_AI_DEBUG=1 \
OLLAMA_MODEL=qwen2.5-coder:3b \
OLLAMA_NUM_CTX=2048 \
./svn_ai_msg.sh -p /path/to/svn/working-copy -n 3 --dry-run
```

The expected output is a terminal report containing a file-change summary and six candidate commit messages. In dry-run mode, no commit is executed. In normal mode, the user can select one of the proposed candidates, after which the corresponding `svn commit -m` command is executed.

Declaration on the Use of Generative AI and AI-Assisted Tools

The original concept, objectives, functional requirements, and overall design of the software presented in this report were developed by the author. During software development, the author used ChatGPT, GitHub Copilot, DeepSeek, and Google Gemini as AI-assisted programming tools. These systems were used to obtain implementation suggestions, generate candidate code fragments, identify possible improvements, and support the revision of selected parts of the source code.

All AI-generated suggestions and code fragments were critically examined by the author before being incorporated into the software. The author performed the integration, debugging, testing, and validation of the resulting implementation and independently verified its behaviour and outputs. AI-generated material was not accepted or used without human review.

The author retains full responsibility for the software architecture, the final source code, the experimental assessment, the interpretation of the results, and the content of this technical report. The AI-assisted tools are not considered authors or contributors and bear no responsibility for the work.

Acknowledgements

The author acknowledges the developers and maintainers of Subversion, Ollama, and the Qwen model family, whose tools make it possible to implement and test local LLM-assisted development workflows. The author also acknowledges GitHub and Zenodo as dissemination and archival infrastructures used to make the software source code and the archived release available with a persistent identifier.

No specific external funding was used for the preliminary work reported in this technical report. The present technical report is associated with the DOI reported in the document metadata.

A Appendix: Reproducibility Checklist

This appendix summarizes the current reproducibility status of the work. The checklist is intended to make explicit which elements are already available and which elements require future extension.

DOI: 10.5281/zenodo.21026831

- **Source code archived with a persistent identifier: yes.** The software `svn_ai_msg` is archived on Zenodo with DOI `https://doi.org/10.5281/zenodo.20849102` and is also available through the public GitHub repository `https://github.com/marcorighi/svn_ai_msg`.
- **Input data or synthetic data generation scripts available: partially.** The tool does not require a fixed dataset. It operates on local SVN working copies containing uncommitted changes. The preliminary evaluation was based on controlled local test cases. A more systematic future evaluation should provide a dedicated synthetic SVN testbed or scripts that generate reproducible working-copy changes.
- **Software versions documented: partially.** The report documents the main software requirements: Bash, Subversion, Python 3, Ollama, and the default model `qwen2.5-coder:3b`. The tested hardware environment included an NVIDIA TU117GLM Quadro T2000 Mobile / Max-Q GPU, NVIDIA driver 610.43.02, and CUDA UMD 13.3. Future releases should also record the exact operating-system version, Subversion version, Python version, Ollama version, and model digest.
- **Random seeds documented: partially.** The script uses deterministic seed values derived from the selected creativity level. However, local LLM execution may still depend on model version, backend implementation, context size, and runtime configuration. Future experiments should report the exact seed, model digest, and generation parameters for each test case.
- **Commands for reproducing outputs documented: yes.** The report provides commands for cloning the repository, preparing the script, running default execution, running dry-run execution, setting the model and context length, and enabling debug output. These commands reproduce the expected terminal workflow, including the file-change summary and the six candidate commit messages.
- **Known limitations and failure cases reported: yes.** The report explicitly describes the preliminary nature of the evaluation, the use of a limited number of representative cases, the constraints imposed by local hardware, the dependence on context selection, and the possibility that generated candidates may be too generic, partially overlapping, or focused on secondary changes.

References

DOI: 10.5281/zenodo.21026831