CS846 Week 8 Summary Review: Paper 1

Brian Do

Paper: *Prompting in the wild: An empirical study of prompt evolution in software repositories.*

Problem Being Solved

As developers increasingly build applications that embed LLMs, “prompts” - natural-language instructions - have become a central software artifact. Yet, little is known about how prompts evolve in real-world software projects: how developers modify and refine them, and what patterns of “prompt engineering” emerge in practice. This paper addresses that gap by empirically analyzing prompt evolution directly within software repositories.

New Idea

The paper’s central contribution is an empirical, repository-scale study of how prompts evolve in open-source projects. The authors compile a dataset of 1,262 prompt changes across 243 GitHub repositories and analyze these edits through four research themes: (1) types and patterns of prompt changes, (2) co-occurrence of prompt and code changes across maintenance activities, (3) documentation and developer intent behind prompt edits, and (4) the impact of prompt changes on logical consistency and on LLM-generated outputs.

Methodologically, the paper treats prompts as first-class software artifacts, analogous to code or documentation, and applies software-evolution analysis to their lifecycle. The authors refine an existing taxonomy of prompt components, annotate edits as additions, modifications, or removals, and investigate where these changes occur in the software lifecycle. They also use an LLM (via chain-of-thought prompting) to detect potentially inconsistent edits, followed by manual validation - combining automated scale with human oversight to estimate real-world impact.

Key findings include: prompt edits are primarily additions or modifications rather than removals; most occur during feature development; feature development emphasizes new instructions, bug fixes focus on modifying existing behavior, and refactoring targets clarity improvements. Only 21.9% of prompt changes are documented in commit messages, and those that are tend to focus on task-specific improvements rather than general prompt-engineering concerns. Prompt edits frequently introduce logical inconsistencies, especially in instruction alignment and structure, and prompt modifications do not always achieve their intended effects on LLM outputs.

Positive Points

- The paper has a systematic and rigorous methodology. The authors carefully mine GitHub repositories to identify prompt-related files and track their evolution through version histories. Their pipeline feels well-designed, replicable, and offers a foundation for future studies on prompt management in real projects.
- The paper has novel empirical angle on LLM usage. The study investigates a genuinely new and timely topic - how prompts evolve in the wild - bridging empirical software engineering and prompt engineering.
- They created a valuable dataset foundation for future research. It represents one of the first artifact-level datasets of its kind, enabling reproducibility and future comparative analyses.

Negative Points

- The study infers developer intent mainly from commit messages and code artifacts, without direct qualitative input (interview or survey). The current analysis focuses on “what” changes occur, but less on “why,” limiting interpretability of prompt evolution as meaningful improvement.
- The dataset focuses on open-source Python projects and excludes short prompts (under 15 words). As the authors note, this may miss key categories of prompts (e.g., concise task cues, configuration prompts, or enterprise code), which limits the generalizability of findings.
- The analysis of whether prompt edits achieved intended changes relies on a small subset of data (7 projects with usable test cases out of 101 with documented changes). The results, while insightful, remain suggestive rather than conclusive about prompt predictability and behavioral consistency.

Future Work

A valuable extension would be a developer-centered study exploring why prompt edits are rarely documented and how documentation practices could be improved. Using the dataset from this paper, one could contact maintainers of LLM-integrated repositories and conduct a mixed-methods study combining surveys and semi-structured interviews. The goal would be to identify real-world barriers such as time constraints, lack of standards, or insufficient tooling. Based on those insights, a lightweight solution could be prototyped - for example, an IDE plugin or Git commit template that automatically detects prompt edits and prompts developers to record intent or expected behavior. Such tooling could make prompt maintenance more transparent and reliable.

Rating

I give this paper 4/5. It is an excellent foundational study, but it needs more human-centered depth to fully explain why prompt evolution happens.

Discussion Points

- Should prompts be version-controlled and reviewed like code? Should prompts have dedicated versioning conventions, regression testing frameworks, or linting tools to detect contradictions or drift?
- Can we measure "good" prompt evolution empirically? Should quality be judged by length, clarity, or effectiveness?
- How can we mitigate unpredictable effects of prompt changes? Is the unpredictability inherent or is it due to prompt construction choices?

Class Discussion Summary

Discussion Topic 1: Should prompts be version-controlled and reviewed like code?

1. Most agreed that prompts should be version-controlled to track changes and the rationale behind them, enabling systematic improvement and auditability.
2. However, for individual developers or small exploratory tasks, strict version control might be overly heavy or difficult to standardize.
3. One idea was to treat prompts like saved SQL queries - automatically versioned and linked to specific data or experiments - to enable quick replay and comparison.
4. LLM-assisted reviews could provide "prompt linting" by flagging ambiguity, risky phrasing, or missing context, supporting a lightweight review process similar to code review.

Discussion Topic 2: Can we measure good prompt evolution empirically?

1. Earlier models tended to respond better to longer, more detailed prompts, whereas newer models (e.g., GPT-5) often perform better with concise, focused ones.
2. Prompt quality should be evaluated relative to its intended purpose or task type, rather than a single universal metric.
3. Future research could work toward developing shared, standardized measures for prompt quality and evolution over time, enabling more rigorous comparisons across studies and models.