Code review pomocí velkého jazykového modelu

Valdemar Pospíšil

Květen 2025

Abstrakt

Tato práce se zabývá využitím velkých jazykových modelů (LLM) při procesu kontroly zdrojového kódu, tzv. *code review*, ve vývoji softwaru. Cílem je prozkoumat možnosti, přínosy a limity těchto modelů v reálném vývojářském workflow a navrhnout experimenty, které ověří jejich efektivitu ve srovnání s lidskými recenzenty.

1 Úvod do tématu

V současném softwarovém vývoji představuje *code review* nedílnou součást procesu zajišťující kvalitu zdrojového kódu. Slouží nejen k odhalování chyb, ale i k předávání znalostí mezi členy týmu, udržování konzistentního stylu kódu a zvyšování celkové udržovatelnosti systému. Tato praxe je klíčová zejména ve větších týmech a projektech s dlouhodobým vývojem. V posledních letech se do vývojářského procesu stále více zapojují nástroje založené na umělé inteligenci. Jedním z nejvýraznějších pokroků v této oblasti jsou tzv. *velké jazykové modely* (LLM – Large Language Models), jako je ChatGPT, Claude, Gemini nebo GitHub Copilot. Tyto modely dokáží porozumět strukturovanému i nestrukturovanému textu a generovat smysluplné odpovědi, komentáře nebo návrhy na základě vstupních dat. Otázkou tedy je, do jaké míry lze tyto nástroje využít pro automatizaci nebo podporu code review. Může LLM odhalit stejné chyby jako zkušený programátor? Je jeho návrh na refaktoring použitelný v reálném prostředí? A především – může takový model plnohodnotně doplnit, nebo dokonce nahradit lidského recenzenta? Tato seminární práce si klade za cíl popsat současný stav výzkumu v této oblasti, formulovat výzkumné otázky a navrhnout experiment, který pomůže zodpovědět, jak efektivní je využití LLM při provádění code review.

2 State-of-the-art

V současné době dochází k výraznému průniku nástrojů umělé inteligence do procesu vývoje softwaru, včetně code review. Tato sekce představuje stručný přehled aktuálního stavu výzkumu s důrazem na oblasti relevantní pro naše výzkumné otázky.

2.1 Výhody a nevýhody lidského code review

Tradiční procesy revize kódu, ačkoliv jsou zásadní pro udržení kvality softwaru a sdílení znalostí v týmu [5], čelí několika významným výzvám. Mezi hlavní nevýhody lidského code review patří především časová náročnost, možnost lidské chyby a nekonzistence hodnocení, která může pramenit z odlišných zkušeností a standardů jednotlivých revidentů. Jak uvádí Falcon [2], tyto nedostatky mohou vést ke zpoždění v rámci agilních vývojových cyklů, jako je kontinuální integrace a nasazování (CI/CD), a k nedostatečnému odhalení specifických technických problémů, pokud revidující postrádá hlubší znalost konkrétní technologie.

Na druhou stranu, lidské code review přináší nezpochybnitelné výhody. Foster [3] zdůrazňuje, že lidští recenzenti excelují v porozumění širšímu kontextu aplikace, dokáží identifikovat subtilní architektonické problémy a zohledňovat specifické požadavky projektu či organizace. Navíc tento proces slouží jako prostředek ke sdílení znalostí a mentoringu v týmu, což je aspekt, který automatizované nástroje nemohou plně nahradit.

2.2 Výhody a nevýhody LLM code review

V reakci na limity lidského code review se do popředí dostává potenciál umělé inteligence. AI nástroje, zejména velké jazykové modely, slibují automatizaci určitých aspektů revize kódu. Dle Falcona [2] mohou LLM provádět statickou analýzu kódu k identifikaci běžných syntaktických chyb, stylistických prohřešků, potenciálních bezpečnostních zranitelností či použití zastaralých částí kódu. Dále mohou navrhovat vylepšení směřující k lepší čitelnosti, efektivitě a udržovatelnosti kódu v souladu s osvědčenými programátorskými postupy. AI je také schopna detekovat anomálie a odchylky od zavedených týmových konvencí a v neposlední řadě může usnadnit práci lidským revidentům tím, že provede prvotní kontrolu a upozorní na klíčové oblasti vyžadující podrobnější lidské posouzení.

Navzdory těmto výhodám, Foster [3] identifikuje několik klíčových limitací LLM při code review. Mezi tyto nevýhody patří omezené chápání kontextu celé aplikace, problém s halucinacemi (generování přesvědčivě znějících, ale fakticky nesprávných informací) a zejména tzv. "nekritická pasivnost", kdy modely nejsou schopny rozpoznat subtilní designové problémy. Dalším významným omezením je absence porozumění specifickým potřebám projektu a organizačním standardům, které nejsou explicitně vyjádřeny v kódu samotném.

2.3 Praktické implementace v reálném prostředí

Příklad praktického nasazení LLM pro code review popisuje Bjerring [1] na implementaci ve společnosti Faire. Ta vyvinula orchestrátorovou službu *Fairey*, která propojuje GitHub webhooky s OpenAI Assistants API a využívá techniku RAG (Retrieval Augmented Generation) pro získání kontextu specifického pro daný pull request. Tato

architektura umožňuje automatické spouštění review při splnění kritérií jako je jazyk kódu nebo obsah změn.

Klíčovým přínosem této integrace je snížení latence v procesu review. LLM dokáží rychle zpracovat rutinní úkoly, zatímco lidští recenzenti se mohou soustředit na komplexnější problémy vyžadující hlubší kontext. Zkušenosti Faire demonstrují, že i když LLM nenahradí lidské recenzenty v oblastech jako architektonické rozhodování, jejich role v automatizaci rutinních kontrol se stává významným doplňkem vývojového procesu.

2.4 Nástroje a technologie pro automatizované code review

V současné době existuje několik způsobů, jak využít LLM pro code review v různých fázích vývojového procesu. Foster [3] popisuje jednoduchý, ale účinný přístup pro ad-hoc code review: k URL adrese pull requestu na GitHubu stačí přidat příponu `.diff`, zkopírovat výsledný diff soubor a vložit ho do libovolného chatovacího LLM jako ChatGPT, Claude nebo Gemini. Tento přístup je limitován kontextovým oknem daného modelu, ale poskytuje rychlou zpětnou vazbu bez nutnosti specializovaných nástrojů.

Pro systematičtější integraci do vývojového procesu Falcon [2] představuje řešení založené na kombinaci git hooks a Code Llama modelu běžícího v Docker kontejneru. Jeho implementace spočívá ve vytvoření pre-commit hooku, který automaticky spouští code review pro všechny modifikované Python soubory před dokončením commitu. Tento přístup nabízí několik výhod:

- Okamžitá zpětná vazba ještě před odesláním kódu do repozitáře
- Konzistentní kontrola kódu pro každou změnu
- Automatizovaná dokumentace doporučení v markdown formátu
- Možnost lokálního běhu bez závislosti na externích službách

Vedle těchto přístupů existují i integrovaná řešení jako GitHub Copilot [4], který poskytuje code review přímo v prostředí GitHub pull requestů, nebo samostatné nástroje jako Code Rabbit, které se automaticky aktivují při vytvoření pull requestu. Tyto nástroje často využívají pokročilé techniky jako je RAG (Retrieval Augmented Generation) pro lepší porozumění kontextu kódu a poskytují strukturovanější a relevantnější zpětnou vazbu než obecné chatovací modely.

3 Výzkumné otázky

V rámci této práce se zaměřím na následující výzkumné otázky:

- **Jak přesná je detekce chyb (bugů, antipatternů) LLM ve srovnání s lidským code reviewerem?**
  Tato otázka je zásadní pro pochopení skutečné efektivity LLM v kontextu code review. Zaměřuje se na schopnost modelů identifikovat různé typy problémů v kódu - od syntaktických chyb přes sémantické problémy až po narušení designových vzorů a architektonické nedostatky. Současný výzkum naznačuje, že LLM mohou být efektivní při identifikaci formálních chyb, ale jejich schopnost odhalit subtilnější problémy vyžadující kontextuální porozumění může být omezená.

Experiment bude zahrnovat kvantitativní srovnání počtu a typů nalezených problémů mezi LLM a lidskými recenzenty.

- **Má nekritická pasivnost vliv na kvalitu code review?**
  Nekritická pasivnost představuje tendenci LLM vyhýbat se kritickým hodnocením a přílišné důvěře v předložený kód. Tato otázka zkoumá, do jaké míry tento fenomén ovlivňuje kvalitu a užitečnost automatizovaného code review ve srovnání s lidskými recenzenty.

- **Jak dobře si LLM poradí s review kódu v méně běžném jazyce jako je Haskell?**
  Zaměřím se výhradně na Haskell, jelikož jde o méně používaný funkcionální programovací jazyk s odlišným paradigmatem než běžnější imperativní jazyky. Tato volba je zajímavá především proto, že na internetu existuje znatelně méně zdrojových kódů v Haskellu oproti jazykům jako Python, Java nebo JavaScript. To může potenciálně znamenat, že LLM měly během svého trénování k dispozici méně příkladů a best practices specifických pro Haskell, což by mohlo vést k méně kvalitním výsledkům code review pro tento jazyk.

4 Návrh experimentu

- příprava prostředí(code base, code na přidání)
- jak jsem použil nástroje
- jak budu hodnotit výstupy od LLM

5 Výsledky a diskuze

- výsledky review
- opovězení na otázky
- jaké jsou dopady na týmovou práci

6 Závěr

- shrnutí zjištění
- moje omezení (no money na chat premium, a nedělám v týmu se seniorem který by mi dal dobrý cr a tak)

Reference

[1] Luke Bjerring. Automated code reviews with llms. *The Craft by Faire*, 2024. URL `https://craft.faire.com/automated-code-reviews-with-llms/`.

[2] Falcon. How to get automatic code review using llm before committing, 2024. URL https://dev.to/docker/how-to-get-automatic-code-review-using-llm-before-committing-3nkj. Příspěvek na blogu dev.to.

[3] Greg Foster. Ai won't replace human code review. *Graphite Blog*, 2023. URL https://graphite.dev/blog/ai-wont-replace-human-code-review.

[4] GitHub. Github copilot: Your ai pair programmer, 2023. URL https://github.com/features/copilot.

[5] Jiří Knesl. Jak na code review. *Zdrojak*, 2022. URL https://zdrojak.cz/clanky/jak-na-code-review/.