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 Přesnost detekce chyb

Srovnání efektivity LLM a lidského code review představuje jedno z klíčových témat aktuálního výzkumu. Kang et al. [**?**] zjistili, že LLM modely dokáží v některých případech identifikovat stejné nebo dokonce více problémů než lidští recenzenti, především v oblasti konzistence kódu a základních syntaktických chyb. Lidští recenzenti však stále excelují v identifikaci logických chyb a problémů vyžadujících hluboké porozumění doméně.

Taecharungroj et al. [**?**] poukazují na několik klíčových limitací LLM při code review, jako je 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.

2.2 Výzvy tradičního code review a potenciál AI

Tradiční procesy revize kódu, ačkoliv jsou zásadní pro udržení kvality softwaru a sdílení znalostí v týmu [3], čelí několika významným výzvám. Mezi ně 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. V reakci na tyto limity se do popředí dostává potenciál umělé inteligence. AI nástroje, zejména velké jazykové modely, slibují automatizaci určitých aspektů revize kódu. Mohou 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 [2].

2.3 Vliv jazykových faktorů

Přestože současný výzkum přímo neadresuje rozdíly v kvalitě code review mezi různými přirozenými jazyky, Fan et al. [**?**] zdůrazňují flexibilitu LLM v interpretaci kódu a jejich schopnost poskytovat vysvětlení v přirozeném jazyce. Většina modelů byla primárně trénována na anglicky psaných materiálech, což může potenciálně vést k rozdílné kvalitě zpětné vazby v různých jazycích. Co se týče programovacích jazyků, Li et al. [**?**] v rámci projektu AlphaCode demonstrovali schopnost modelů řešit úlohy v různých programovacích jazycích, nicméně efektivita modelů pro méně rozšířené jazyky jako Haskell nebyla dosud dostatečně prozkoumána. Lze předpokládat, že modely budou vykazovat nižší výkon v jazycích, které jsou méně zastoupeny v trénovacích datech.

2.4 Integrace do vývojového cyklu

Příklady z praxe ukazují, že integrace LLM do procesu code review vyžaduje pečlivé nastavení workflow a technické infrastruktury. Společnost Faire 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. Jak uvádí Bjerring [1], tato architektura umožňuje automatické spouštění review při splnění kritérií (např. jazyk kódu nebo obsah změn), přičemž LLM dokáží efektivně zpracovat generické aspekty revizí jako vynucování stylu kódu, kontrolu testovacího pokrytí nebo detekci zpětně nekompatibilních změn.

Klíčovým přínosem této integrace je snížení latence v procesu review. LLM dokáží rychle zpracovat rutinní úkoly, zatímco lidské recenzenty se mohou soustředit na komplexnější problémy vyžadující hlubší kontext. Faire navíc implementoval dvouúrovňový systém hodnocení kvality: kvantitativní (pomocí LLM evaluačních nástrojů) a kvalitativní (feedback od vývojářů). Tento hybridní přístup, jak poznamenává Bjerring [1], umožňuje iterativní vylepšování modelů: „While LLMs offer flexibility and speed, their outputs require iterative refinement of prompts to ensure consistency and correctness.“

Pro udržení konzistence v agilních vývojových cyklech Faire využívá tzv. *fixtures* – snapshoty výstupů funkcí pro pozdější re-use. Toto řešení adresuje problém transientní povahy pull requestů a ukazuje, že úspěšná integrace LLM do CI/CD vyžaduje nejen kvalitní modely, ale i robustní infrastrukturu pro správu kontextu. Zkušenosti Faire demonstrují, že i když LLM nenahradí lidské recenzenty v oblastech jako architektonické rozhodování, jejich role v automatizaci rutinních kontrol se stává nepostradatelnou.

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.

- **Má jazyk vliv na code review?**
  V rámci experimentu budu porovnávat výsledky code review prováděného v češtině a angličtině. Toto srovnání může být zajímavé vzhledem k tomu, že dat v českém jazyce je výrazně méně než v angličtině, což by mohlo ovlivnit kvalitu

zpětné vazby od LLM. Zároveň je možné, že na programovacím jazyce a jeho syntaxi nezáleží tolik jako na přirozeném jazyce, ve kterém je review prováděno.

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