Generování dokumentace k existujícímu kódu pomocí LLM

Amirkhan Abayev

18. května 2025

Obsah

1 Úvod do tématu

Dokumentace softwaru je základní součástí udržitelného vývoje. Umožňuje popsat chování aplikace přirozeným jazykem bez nutnosti čtení zdrojového kódu. To je výhodné jak pro stakeholdery – nadřazené, product owners či testery, kteří nemusí rozumět implementaci, tak pro vývojáře, kteří se po čase vracejí k „legacy“ modulům. Kvalitní dokumentace tedy snižuje technický dluh a zrychluje onboarding nových členů týmu.

Dokumentace existuje na různých úrovních:

- Architektonická a designová (diagramy, komunikační toky),
- Modulová a souborová (README, API reference),
- Funkční (popisy tříd a metod).

Právě funkční popisy zajišťují, že každá funkce, metoda nebo třída má jasně definované vstupy, výstupy i možné chyby. V Pythonu se běžně používají docstringy – řetězcové literály umístěné těsně pod hlavičkou objektu, které interpreter ukládá do atributu `__doc__`. Docstringy jsou vnitřním zdrojem pravdy:

- Interaktivně je lze vypsat příkazem `help(my_function)`.
- IDE a nástroje pro generování dokumentace (Sphinx, pdoc) je automaticky extrahují.
- Podporují jednotný workflow – změna logiky vyžaduje úpravu jediného řetězce, nikoli externí dokumentace.

```
def parse_boolean_state(text: str) -> bool:
    """
    Convert textual flag to boolean.

    Args:
        text: Input value such as "yes", "no", "1", "0".

    Returns:
        True  - if text represents an affirmative value;
        False - otherwise.
```

```
    Raises:
        ValueError: If the string cannot be interpreted.
    """
    ...
```

Docstring může být stručný:

```
class Timer:
    """Context-manager that measures elapsed time."""
```

nebo víceúrovňový podle stylu projektu (Google style [1], reST (PEP 287) [2], NumPy [3]).

S nástupem velkých jazykových modelů (LLM) se objevila možnost automatizovat psaní docstringů. Místo ručního vyplňování se specialist stává „editorem“ a „validátorem“ výstupu modelů. Přesto LLM trpí zásadními nevýhodami:

- Nedostatečná přesnost: modely mohou nepřesně popsat logiku kódu.
- Halucinace: vytvářejí věrohodně znějící, avšak falešné informace (např. neexistující parametry).

Cílem této práce je experimentálně ověřit, jak efektivně dokážou LLM generovat docstringy pro existující Python kód, jaké chyby se při tom vyskytují a zda kvalita automaticky generovaných popisů dosahuje úrovně středně pokročilého vývojáře.

2 Výzkumná rešerše

V posledních dvou letech vzniklo několik významných prací zabývajících se automatizovanou generací dokumentace kódu pomocí LLM:

- **Poudel et al.** představili systém *DocuMint*, který ukazuje, že i malé modely (2–8 mld. parametrů) po doladění na rozsáhlém datasetu 100 000 párů (funkce–docstring) dosahují až o 22,5 % lepší stručnosti a o 12,7 % vyšší přesnosti oproti stavu bez doladění; přitom Llama3 8B bez dalšího tréninku stále vede v metrice CodeBLEU [4]. Tato práce zároveň demonstruje využití LoRA pro zefektivnění doladění na menších GPU farmách, což otevírá cestu pro nasazení v menších týmech či firmách.

- **Dvivedi et al.** provedli komparativní analýzu pěti významných LLM (GPT-3.5, GPT-4, Bard, Llama 2, StarChat) na úlohách generování dokumentace na úrovních inline, funkce a soubor; hodnotili přesnost, úplnost i relevanci, stejně jako čas generování [5]. Zavedení objektivního checklistového hodnocení ukázalo, že uzavřené modely (GPT-4, Bard) obecně převažují nad open-source konkurencí, přičemž file-level dokumentace dosahuje významně horších výsledků než funkční či inline dokumentace. Autoři navrhují kombinaci více-krokových pipeline nebo víceagentní architektury pro rozsáhlejší generaci API-dokumentace.
- **Chen et al.** zkoumali retrieval-augmented generation (RAG) v kontextu málo frekventovaných či „zapomenutých“ knihoven, kde LLM obvykle postrádá znalosti obsažené v tréninkových datech. Kombinací vyhledávání relevantních částí oficiální dokumentace a následné generace docstringů dosáhli zvýšení přesnosti volání API až o 83–220 % [6]. Zjistili, že největší přínos přináší vložení reálných ukázek kódu, zatímco samotné popisy parametrů mají spíše doplňkový efekt.

Tyto studie společně ukazují klíčové trendy:

- potřebu doladění modelů na specializovaných datech,
- výhody více-krokových či víceagentních architektur,
- zásadní roli externího retrievalu pro pokrytí méně obvyklých API.

Nicméně stále chybí detailní experimenty zaměřené na objektivnou kvantifikaci chyb typu „vynechaný parametr“ či „nesprávný návratový typ“ a statistické porovnání současných modelů OpenAI. V této práci na ně navážu přesným měřením Coverage, Hallucination i BERTScore mezi GPT-4o, o4-mini-high a o3.

3 Výzkumné otázky

Formuluji tři klíčové výzkumné otázky, na jejichž základě budu hodnotit výkon a spolehlivost jednotlivých variant LLM při generování docstringů:

1. **Dokáže LLM generovat docstringy srovnatelné s původními?** Touto otázkou zjišťuji, nakolik se automaticky vytvořené popisy blíží lidsky psaným referenčním docstringům. Porovnám je pomocí metriky

BERTScore F1, která měří semantickou blízkost mezi gold docstringem a výstupem modelu. Vyšší průměrné skóre bude indikovat, že model zachytil význam a detaily původního popisu.

2. **Jaká je míra chyb u každého modelu?** Zde kvantifikuji konkrétní druhy chyb, kterých se modely dopouštějí:
   - *Vynechaný parametr* – počet parametrů ze signatury, které nejsou v docstringu zmíněny (Coverage %).
   - *Halucinace* – počet zmínek o parametrech nebo návratových typech, které ve skutečnosti signatura neobsahuje (Hallucination %).
3. **Existuje statisticky významný rozdíl mezi GPT-4o, o3 a o4--mini-high?** Cílem je ověřit, zda zjištěné rozdíly ve výkonu jsou statisticky podložené, nebo by mohly být dílem náhody. Pro každou z hlavních metrik (BERTScore, Coverage, Hallucination) použiji párový t--test ($\alpha = 0{,}05$) mezi dvojicemi modelů: GPT-4o vs o4-mini-high, GPT--4o vs o3 a o4-mini-high vs o3. Výsledek $p < 0{,}05$ mi umožní tvrdit, že jeden model generuje konzistentně lepší nebo horší docstringy než druhý.

4 Návrh experimentu

4.1 Výběr dat

Do experimentu jsem zařadil 15 funkcí a metod vybraných z deseti různých repozitářů [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]. Kritéria výběru byla dvojí:

- Novinky z populárních Python knihoven (verze vydané v letech 2024–2025, např. pydantic 2.11.0 [22]).
- Funkce ze starších či méně známých balíčků publikovaných před rokem 2020, tedy před nástupem ChatGPT a obdobných LLM „boomu“.

Tím minimalizuji riziko, že by modely již měly tyto ukázky ve svých tréninkových datech, a zajišťuji, že vygenerované docstringy budou skutečně originální. Původní docstringy slouží jako gold standard a jejich odstraněním (code_no_doc) vytvářím čistý vstup pro modely.

4.2 Modely a prostředí

Generování probíhalo v uživatelském rozhraní ChatGPT Plus s následujícími modely:

- GPT-4o,
- o4-mini-high,
- o3.

Pro každou funkci jsem otevřel samostatný chat, aby se zabránilo ovlivnění odpovědí kontextem předchozích promptů. Tím je zajištěna izolace každého volání a jednotné podmínky pro všechny ukázky.

4.3 Promptování

Jako vstupní prompt jsem použil minimalistickou anglickou instrukci:

```
```
<code_no_doc>
```
```

Add a docstring.
```

Tento stručný formulář ponechal modelům volnost ve volbě stylu, délky i detailu popisu a umožnil otestovat jejich schopnost spontánně zpracovat kód bez dalšího navádění.

## 4.4 Logování výsledků

Všechny odpovědi modelů jsem ručně zkopíroval do Google Sheets. Data jsem poté převedl do "tidy" formátu CSV pro efektivní skriptové vyhodnocení. Tento postup nevyžaduje přímý přístup k API a je zcela reprodukovatelný.

## 4.5 Metody vyhodnocení

Hlavní metrikou pro kvantitativní porovnání generovaných docstringů s referenčními byla BERTScore F1 (model `roberta-large`), která měří semantickou blízkost mezi texty [6]. Doplňuji ji metrikami:
```

- *Coverage %* – podíl parametrů a návratových typů signatury uvedených v docstringu,
- *Hallucination %* – počet fiktivních parametrů či návratových typů, které v kódu neexistují.

Pro ilustraci načtení a výpočtu BERTScore ve skriptu Python:

```
import evaluate

bertscore = evaluate.load("bertscore")
result = bertscore.compute(
    predictions=[extract_docstring(llm) for llm in llm_responses],
    references=[extract_docstring(gold) for gold in gold_docs]
)
# result["f1"] obsahuje skóre pro každou dvojici
```

4.6 Statistická analýza

Pro ověření, zda rozdíly mezi modely nejsou dílem náhody, jsem pro každou metriku provedl párový t-test ($\alpha = 0{,}05$) na datech z 15 funkcí. Kromě p-hodnot jsem spočítal i Cohenovo d pro odhad velikosti efektu.

4.7 Omezení

Experiment probíhal manuálně v uživatelském rozhraní ChatGPT Plus. Omezil jsem se na automatické metriky bez lidských recenzí a zaměřil jsem se pouze na Python. Tento návrh experimentu poskytuje transparentní a reprodukovatelnou cestu k hodnocení schopností moderních LLM generovat kvalitní dokumentaci kódu.

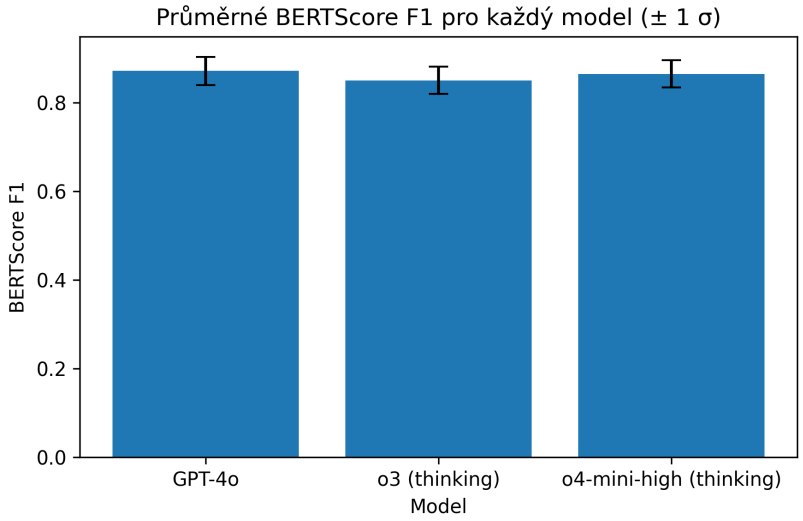
5 Výsledky a diskuze

V této části shrnuji hlavní kvantitativní i kvalitativní závěry experimentu a diskutují jejich dopad na praxi automatizované generace dokumentace kódu.

5.1 Režim „thinking“

Modely `o3` a `o4-mini-high` byly v ChatGPT Plus použity ve verzi označené jako *thinking*, která může prodloužit dobu generování a mírně změnit styl výstupu.

5.2 BERTScore F1



Obrázek 1: Průměrné BERTScore F1 s intervalem $\pm$ 1 α pro porovnané modely.

- GPT-4o: 0,871 $\pm$ 0,032
- o4-mini-high: 0,865 $\pm$ 0,031
- o3: 0,850 $\pm$ 0,031

Párové t-testy ukázaly:

- GPT-4o vs o3: $t = 4{,}27$, $p = 0{,}0008$, $d = 1{,}10$ (velký efekt, signifikantní),
- o4-mini-high vs o3: $t = 3{,}64$, $p = 0{,}0027$, $d = 0{,}94$ (velký efekt, signifikantní),
- GPT-4o vs o4-mini-high: $t = 1{,}69$, $p = 0{,}1135$, $d = 0{,}44$ (střední efekt, nesignifikantní).

5.3 Coverage a Hallucination

Manuální kontrola ukázala, že všechny modely dosáhly

- *Coverage Args/Returns*: 100 %,
- *Hallucination Args/Returns*: 0 %.

To znamená, že žádný parametr či návratový typ nebyl přeskakován ani vymyšlen.

Všechny původní (gold) i generované docstringy jsou k dispozici v repozitáři [23].

5.4 Shrnutí odpovědí na výzkumné otázky

1. **Srovnatelnost docstringů:** GPT-4o a o4-mini-high dosahují průměrného $BERTScore \approx 0,87$, tedy vysoké semantické podobnosti s referencí.
2. **Míra chyb:** Coverage 100 % a Hallucination 0 % potvrzují bezchybnost všech tří modelů v popisu parametrů a návratových typů.
3. **Statistická významnost:** Mezi o3 a oběma ostatními modely existuje signifikantní rozdíl (velké efekty); mezi GPT-4o a o4-mini-high není při $\alpha = 0{,}05$ signifikantní, ačkoliv d = 0,44 naznačuje střední efekt.

6 Závěr

V této práci jsem experimentálně ověřil schopnost moderních LLM generovat docstringy pro existující Python kód. Hlavní poznatky jsou:

- Modely GPT-4o a o4-mini-high dosahují vysoké semantické podobnosti s lidsky psanými docstringy (průměrné $BERTScore \approx 0,87$), zatímco model o3 zaostává ($\approx 0,85$).
- Všechny tři modely vykazují 100 % Coverage a 0 % Hallucination, tedy nevynechávají žádné parametry ani nepřidávají falešné informace.
- Statistická analýza potvrdila signifikantní rozdíl mezi o3 a GPT-4o, o4-mini-high (velké efekty, $p < 0{,}01$), zatímco rozdíl mezi GPT-4o a o4-mini-high nebyl při $\alpha = 0{,}05$ průkazný.

Praktické implikace:

- Pro většinu úloh je vhodné použít GPT-4o, který nabízí nejlepší kvalitu.
- Přestože modely generují kompletní a bezchybné docstringy, doporučuji rychlou kontrolu a odstranění případných importů či úryvků kódu mimo samotný docstring.

Budoucí práce by měla zahrnout:

- Rozšíření experimentu o další programovací jazyky a automatizované sběry dat přes API.
- Začlenění kvality generovaných popisů hodnocené experty.
- Vyzkoušení pokročilých promptovacích strategií a multiagentních přístupů pro zvýšení variability a přesnosti.

Tato studie dokládá praktickou využitelnost LLM k rychlé a spolehlivé dokumentaci kódu, což může významně zrychlit práci vývojářů a snížit technický dluh v reálných projektech.

Odkazy

1. GOOGLE. *Python Style Guide: Comments and Docstrings*. n.d. Dostupné také z: `https://google.github.io/styleguide/pyguide.html#38-comments-and-docstrings`.
2. GOODGER, David. *PEP 287 – reStructuredText Docstring Format*. Python Software Foundation, 2002. Dostupné také z: `https://peps.python.org/pep-0287`.
3. NUMPY COMMUNITY. *Numpydoc Style Guide*. n.d. Dostupné také z: `https://numpydoc.readthedocs.io/en/latest/format.html`.
4. POUDEL, Bibek; COOK, Adam; TRAORE, Sekou; AMELI, Shelah. DocuMint: Docstring Generation for Python using Small Language Models. *arXiv preprint arXiv:2405.10243*. 2024.
5. DVIVEDI, Shubhang Shekhar; VIJAY, Vyshnav; PUJARI, Sai Leela Rahul; LODH, Shoumik; KUMAR, Dhruv. A Comparative Analysis of Large Language Models for Code Documentation Generation. *arXiv preprint arXiv:2312.10349*. 2023.

6. CHEN, Jingyi; CHEN, Songqiang; CAO, Jialun; SHEN, Jiasi; CHEUNG, Shing-Chi. When LLMs Meet API Documentation: Can Retrieval Augmentation Aid Code Generation Just as It Helps Developers? *arXiv preprint arXiv:2503.15231*. 2025.
7. PYDANTIC. *people.py*. n.d. Dostupné také z: `https://github.com/pydantic/pydantic/blob/main/.github/actions/people/people.py#L358`.
8. PYDANTIC. *arguments_schema.py (commit f2e7385)*. n.d. Dostupné také z: `https://github.com/pydantic/pydantic/blob/f2e7385766926dfb14056b8da7fe0b88b2ce9f19/pydantic/experimental/arguments_schema.py#L14`.
9. TEXTUALIZE. *test_traceback.py (commit 17baaed)*. n.d. Dostupné také z: `https://github.com/Textualize/rich/blob/17baaed1c0525f23dca438b8192af79cb52b792e/tests/test_traceback.py#L363`.
10. FRETBOARDFREAK. *template.py*. n.d. Dostupné také z: `https://github.com/fretboardfreak/netify/blob/master/src/netify/template.py#L112`.
11. OLLIONORG. *main.py*. n.d. Dostupné také z: `https://github.com/ollionorg/risc-python/blob/master/risc/main.py#L393`.
12. MYL7. *utils.py*. n.d. Dostupné také z: `https://github.com/myl7/ssrcli/blob/master/ssrcli/utils.py#L11`.
13. BEDNARIK, Radek. *api.py*. n.d. Dostupné také z: `https://github.com/radekBednarik/att/blob/master/apitalker/api.py#L99`.
14. PYBPC. *argparse.py, line 9*. n.d. Dostupné také z: `https://github.com/pybpc/bpc-utils/blob/main/bpc_utils/argparse.py#L9`.
15. PYBPC. *argparse.py, line 50*. n.d. Dostupné také z: `https://github.com/pybpc/bpc-utils/blob/main/bpc_utils/argparse.py#L50`.
16. PYBPC. *argparse.py, line 120*. n.d. Dostupné také z: `https://github.com/pybpc/bpc-utils/blob/main/bpc_utils/argparse.py#L120`.
17. ALMAZKUN. *npss.py, line 6*. n.d. Dostupné také z: `https://github.com/almazkun/npss/blob/main/npss/npss.py#L6`.
18. ALMAZKUN. *npss.py, line 21*. n.d. Dostupné také z: `https://github.com/almazkun/npss/blob/main/npss/npss.py#L21`.
19. GAYA, Nick. *render.py*. n.d. Dostupné také z: `https://github.com/nickgaya/bravado-types/blob/master/bravado_types/render.py#L9`.

20. FINITE-LOOP. *txtutils.py*. n.d. Dostupné také z: https://gitlab.com/finite-loop/flutils/-/blame/master/flutils/txtutils.py?ref_type=heads#L25.

21. FINITE-LOOP. *validators.py*. n.d. Dostupné také z: https://gitlab.com/finite-loop/flutils/-/blame/master/flutils/validators.py?ref_type=heads#L15.

22. PYDANTIC. *Release v2.11.0*. 2025. Dostupné také z: https://github.com/pydantic/pydantic/releases/tag/v2.11.0.

23. ABAYEV, Amirkhan. *llm-docstrings-research*. 2025. Dostupné také z: https://github.com/rr3333mmAA/llm-docstrings-research.