Final Project

MSR 2024 Mining Challenge

Analysing developer-ChatGPT conversations

MSR Challenge

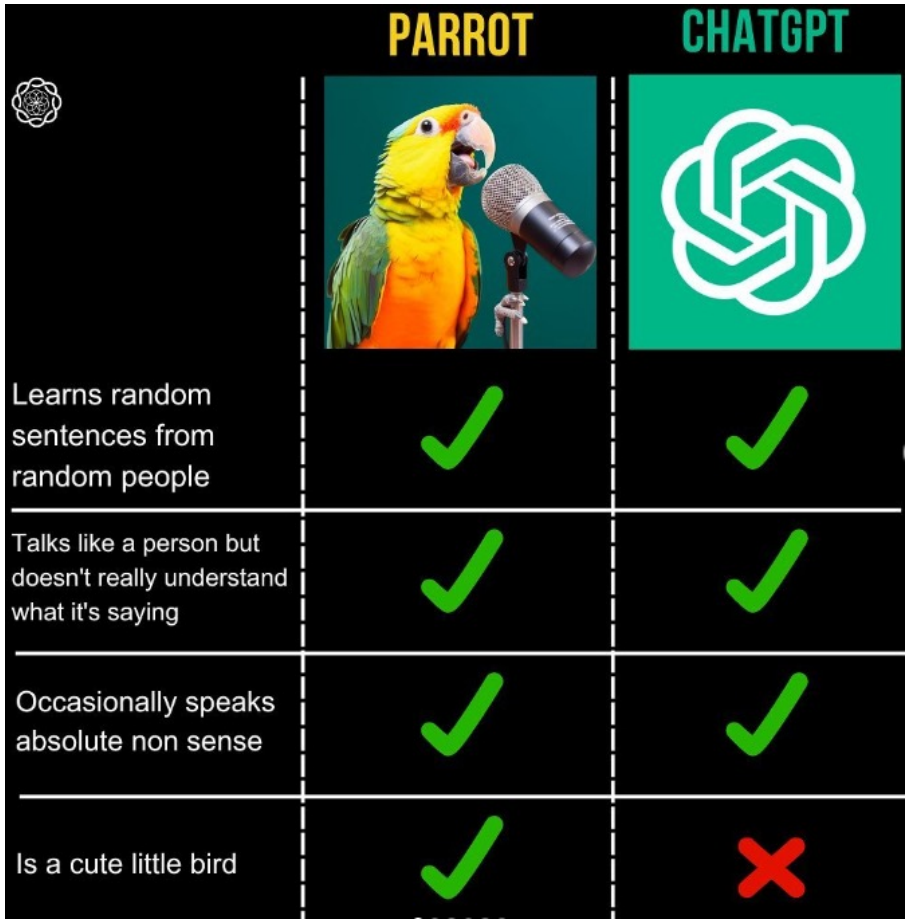
- MSR = International Conference on Mining Software Repositories
- 2024 is MSR 24th edition
- MSR mining challenge started in 2006 and continue annually
  - A real dataset proposed by a team of researchers/practitioners
  - The challenge is for researchers and practitioners **to bravely** use their mining tools and approaches on a dare.
  - https://2024.msrconf.org/track/msr-2024-mining-challenge#Call-for-Mining-Challenge-Proposals
  - Deadlines 7 Dec 2023, Abstract Deadline. 11 Dec 2023, Paper Deadline

LOG6307 Final Project

- You are NOT required to participate in the actual MSR challenge.
- You must submit your final project in Moodle before our internal deadline to get your grade for the project.
  - Proposed two research questions – November 8th
  - Final project presentation - November 28th
  - Final report submission – December 12th

MSR 2024 Mining challenge

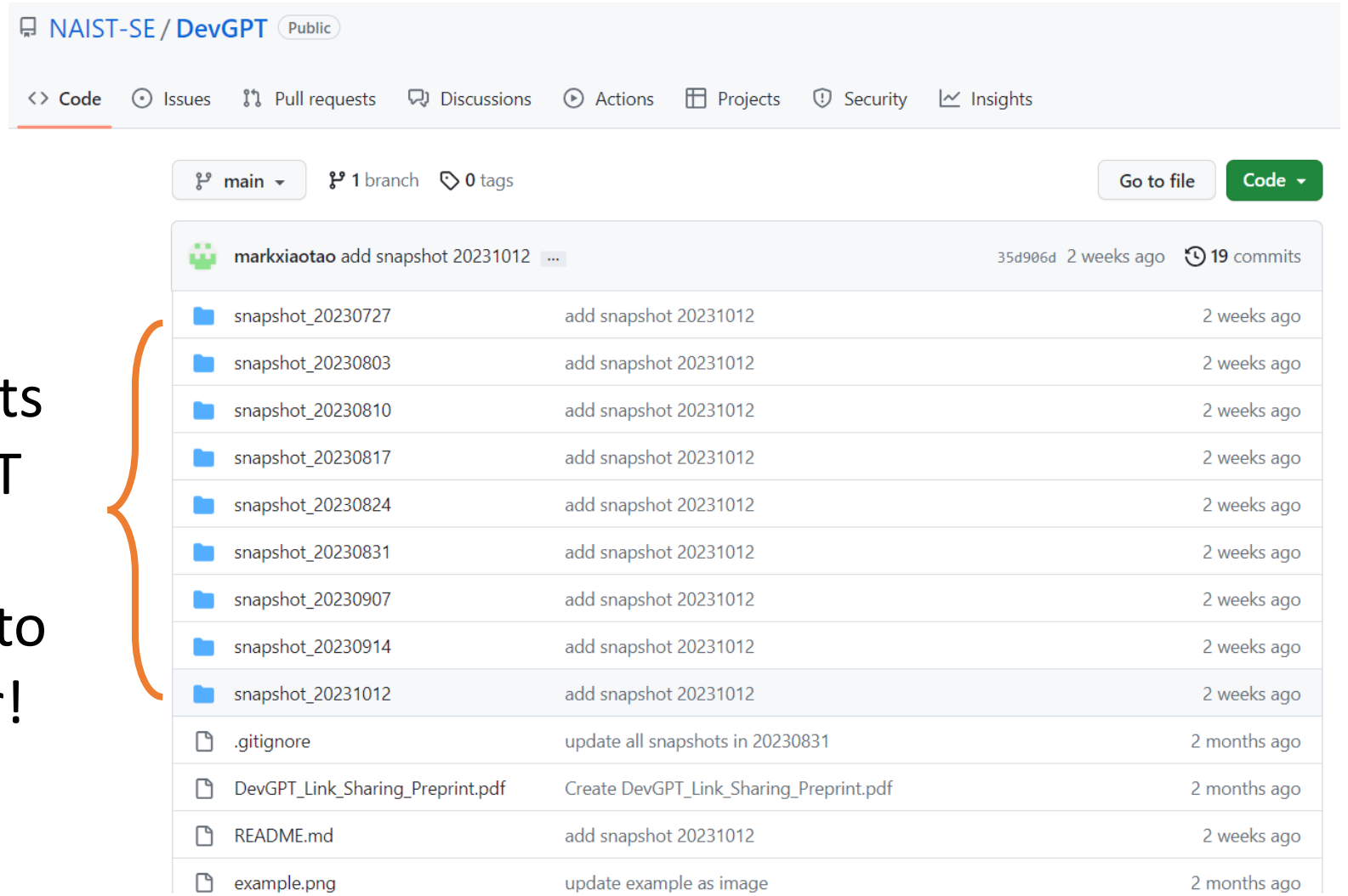
- Large language models (LLMs), e.g., Chat GPT
- Machine learning models, specifically natural language processing models
- Designed to understand and generate human-like text based on the patterns they have learned.
- Building complex models allow them to handle a wide variety of language tasks, from answering questions to generating creative content.



MSR 2024 Mining challenge

- Mine DevGPT database
- DevGPT is a curated dataset which encompasses:
  - 16,129 prompts and ChatGPT’s responses including 9,785 code snippets
  - Coupled with the corresponding software development artifacts, e.g., source code, commits, issues, pull requests, and Hacker News threads
- The goal is to have a thorough understanding of Chat GPT interactions with developers
- The challenge is open-ended

DevGPT data



9 Snapshots of ChatGPT data.
From July to November!

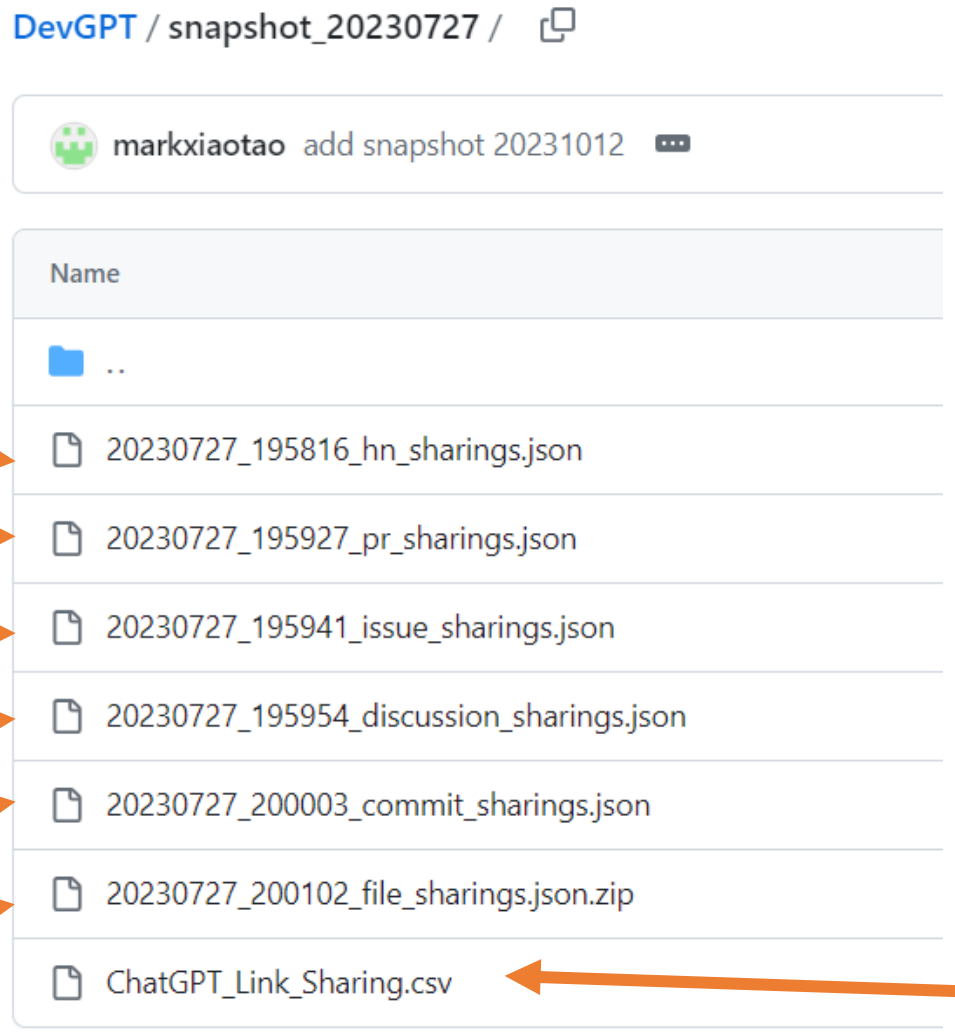
Links shared on GitHub and HackerNews

Top three programming languages:

1. Python (1,735)
2. JavaScript (1,530)
3. Bash (1,435)

Snapshot Data

- HackerNews
- Pull requets
- Issues
- Discussions
- Commits
- Code files



All combined

Attributes of each JSON file could be found here: Link

Snapshot Data

Merged **FlatAHB Dump: Don't use new GUID for otherwise unchanged lines** #158

Author merged 4 commits into `main` from `dont-use-new-guid` 3 weeks ago

Reviewer reviewed 3 weeks ago — View reviewed changes

`src/kohlrahbi/unfoldedahb/unfoldedahbtable.py` Outdated

```
299 +         if dump_equals_existing_file_except_for_guids:
300 +             ahb_line_schema = AhbLineSchema()
301 +             for line, existing_line in zip(flat_ahb.lines, existing_flat_ahb.lines):
302 +                 line_copy = ahb_line_schema.load(ahb_line_schema.dump(line))
```

Reviewer 3 weeks ago • edited — Collaborator

This is a C# pattern for deep copies isn't it?
I thought in Python there are methods to create deep copies.
asking chatGPT
ah yes there is a standard lib called `copy`
See https://chat.openai.com/share/4ad4c1ad-6f13-4a21-a0e3-aaf75e17c556

Reply...

ChatGPT shared conversation

Modified commit based on the review

besser — ✕ 23e4ecf

Deep Copy in Python.

July 24, 2023

You are a Python expert.
How can I create a deep copy of a variable?

Certainly! In Python, if you want to create a deep copy of a variable, especially for complex objects like lists or dictionaries that contain nested objects, you should use the `copy` module which provides the `deepcopy` function.

Here's how you can use it:

python — Copy code

```python
import copy

original_list = [[1, 2, 3], [4, 5, 6]]
deep_copied_list = copy.deepcopy(original_list)

# Now, if you modify the nested list inside the original list...
original_list[0][0] = 99

print(original_list)        # Outputs: [[99, 2, 3], [4, 5, 6]]
print(deep_copied_list)     # Outputs: [[1, 2, 3], [4, 5, 6]] (remains uncha
```

52 src/kohlrahbi/unfoldedahb/unfoldedahbtable.py

```
30 + def _lines_are_equal_when_ignoring_guid(line1: AhbLine, line2: AhbLine) -> bool:
31 +     """
32 +     returns true iff the line1 and line2 are equal except for their guid
33 +     """
34 +     line1_copy = copy.deepcopy(line1)
35 +     line2_copy = copy.deepcopy(line2)
36 +     line1_copy.guid = None
37 +     line2_copy.guid = None
38 +     return line1_copy == line2_copy
```

Suggestions by MSR

- What types of issues (bugs, feature requests, theoretical questions, etc.) do developers most commonly present to ChatGPT?
  - For example, using LDA (latent Dirichlet allocation). Similar paper for StackOverflow
  - Qualitative coding. Similar paper for Stack Overflow: C. Treude, O. Barzilay and M. -A. Storey, "How do programmers ask and answer questions on the web?: NIER track," 2011 33rd International Conference on Software Engineering (ICSE), Honolulu, HI, USA, 2011, pp. 804-807, doi: 10.1145/1985793.1985907.
- Can we identify patterns in the prompts developers use when interacting with ChatGPT, and do these patterns correlate with the success of issue resolution?
- What is the typical structure of conversations between developers and ChatGPT? How many turns does it take on average to reach a conclusion?
- In instances where developers have incorporated the code provided by ChatGPT into their projects, to what extent do they modify this code prior to use, and what are the common types of modifications made?

Suggestions by MSR

- How does the code generated by ChatGPT for a given query compared to code that could be found for the same query on the internet (e.g., on Stack Overflow)?
- What types of quality issues (for example, as identified by linters) are common in the code generated by ChatGPT?
- How accurately can we predict the length of a conversation with ChatGPT based on the initial prompt and context provided?
- Can we reliably predict whether a developer's issue will be resolved based on the initial conversation with ChatGPT?
- If developers were to rerun their prompts with ChatGPT now and/or with different settings, would they obtain the same results?

Suggestions by LOG6307 Teaching team

- Human factor and social factors analysis
  - Sentiment analysis of the prompts. Example:
    - **Stack Overflow:** Novielli, Nicole, Fabio Calefato, and Filippo Lanubile. "Towards discovering the role of emotions in stack overflow." *Proceedings of the 6th international workshop on social software engineering*. 2014.
    - **GitHub discussion**: Pletea, D., Vasilescu, B., and Serebrenik, A. 2014. Security and emotion: sentiment analysis of security discussions on GitHub. In Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014). ACM, New York, NY, USA, 348-351.