AI-Supported Software Development
Moving beyond code completion

Rohith Pudari

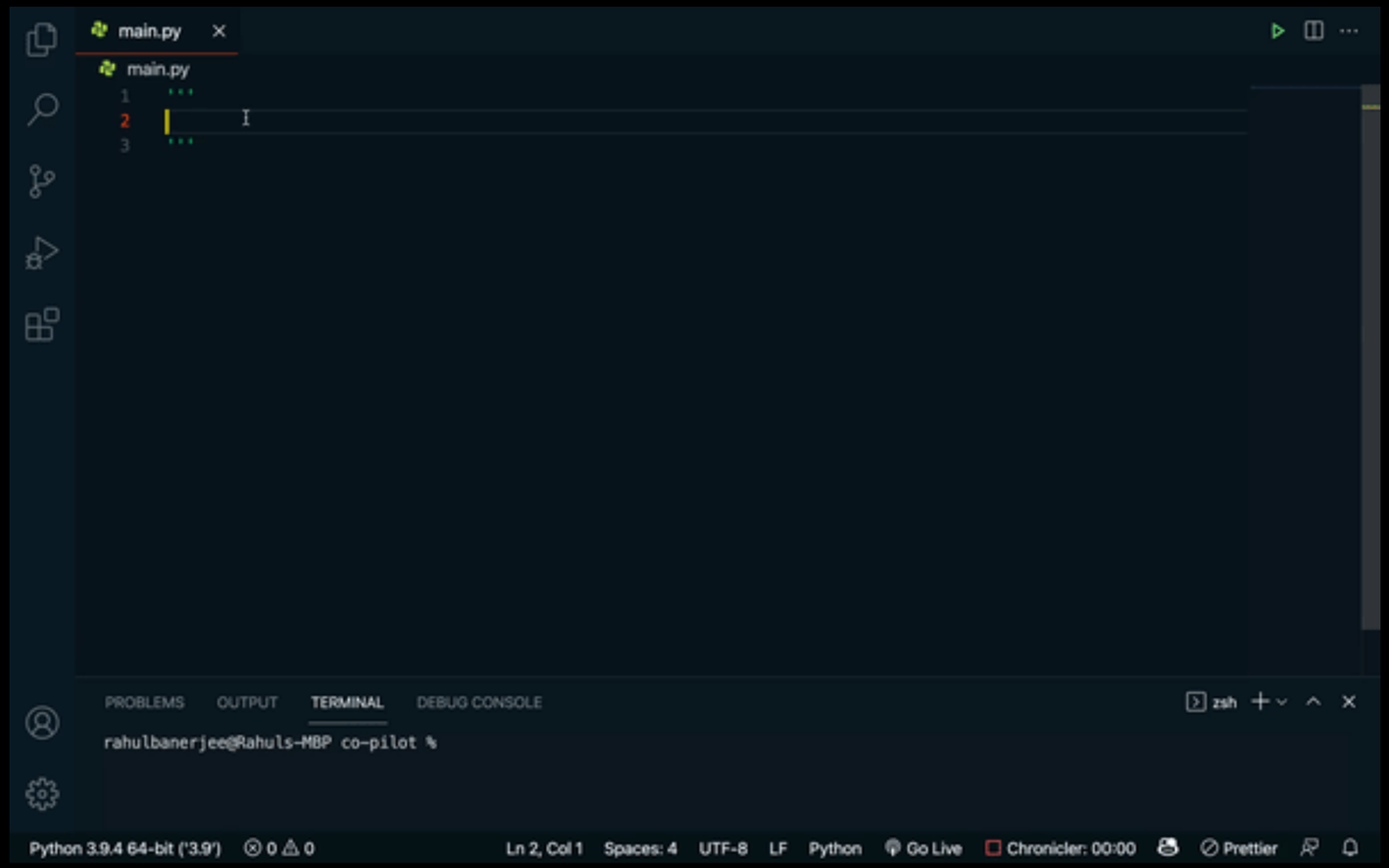
University of Victoria

AI-supported code completion tools

- The purpose of code completion as an IDE feature is to save the user's time and effort by suggesting code before manually inputting it.
- In-IDE code completion tools have improved a lot in recent years. Early code completion techniques include suggesting variables or method calls from user code bases
- Followed by tools capable of suggesting entire code blocks utilizing statistical language models, such as n-gram models, is the origin of the use of data-driven strategies for code recommendation.
- Recent large-scale pre-trained language models such as Codex have demonstrated an impressive ability to generate code and can now solve programming contest-style problems

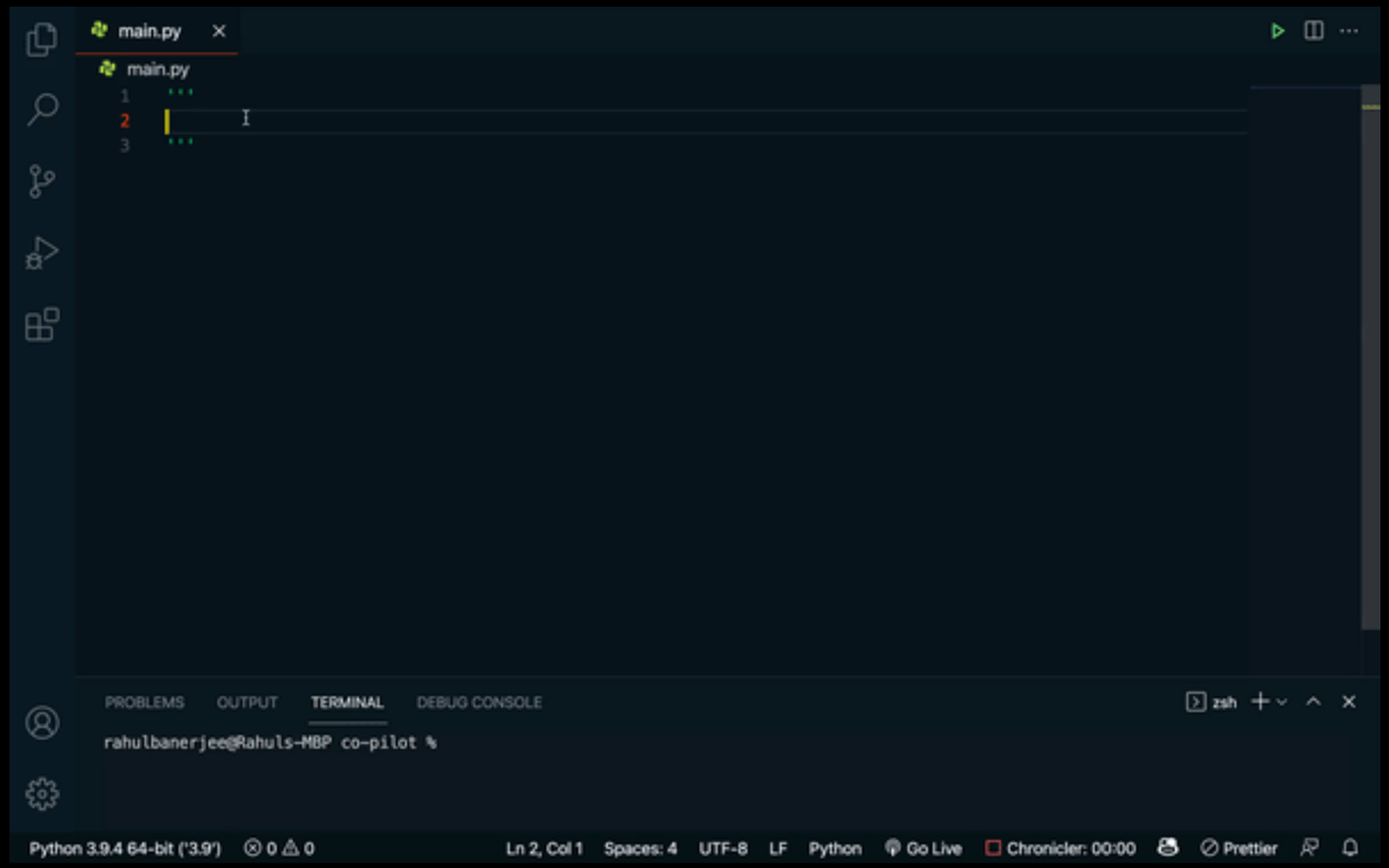
Github Copilot

- Introduced in June 2021, GitHub's Copilot is an in-IDE recommender system that leverages OpenAI's Codex neural language model (NLM) which uses a GPT-3 model.



Github Copilot

- Introduced in June 2021, GitHub's Copilot is an in-IDE recommender system that leverages OpenAI's Codex neural language model (NLM) which uses a GPT-3 model.



Challenges with Copilot

- Copilot is trained on existing software source code and training costs are expensive, several classes of errors have been discovered, which follow from the presence of these same errors in public (training) data.
- Copilot can make simple coding mistakes, such as not allowing for an empty array in a sort routine.
- Copilot does not understand security vulnerabilities, so it will suggest code that not secure.
- Concerns have been raised about Copilot license compliance and copyright violation. Recently, a filter was introduced by GitHub to resolve this problem.

Research Questions

- RQ-1: What are the current boundaries of AI-supported code completion tools?
  - RQ 1.1: How do AI-supported code completion tools manage programming idioms?
  - RQ 1.2: How do AI-supported code completion tools manage to suggest non-smelly code?
- RQ 2: Given the current boundary, how far is it from suggesting design decisions which seem much beyond the boundary?

Pythonic Idioms

- A good AI-supported code completion tool should always use idiomatic approaches in its code suggestions.
- We sampled the top 25 popular Pythonic idioms found in open source projects based on the work of Alexandru et al. [1], and Farook et al. [2].
- Input to Copilot = title of coding scenario + minimal code required to trigger code suggestion from Copilot.
- We consider Copilot passed the coding scenario when it suggests the idiomatic approach as its first suggestion. We also note if the idiomatic approach exists in any of the top 10 suggestion currently viewable.

(1) Carol V. Alexandru, José J. Merchante, Sebastiano Panichella, Sebastian Proksch, Harald C. Gall, and Gregorio Robles. 2018. On the usage of pythonic idioms. In Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2018). Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3276954.3276960

(2) Aamir Farooq and Vadim Zaytsev. There is more than one way to zen your python. In Proceedings of the 14th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2021, page 68–82, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3486608.3486909.

Example of List comprehension idiom

List Comprehension

Human Input:

```
#list comprehension
result_list =
```

Copilot Suggestion:

```
result_list =  []
for i in range(1,11):
    result_list.append(i)
```

Idiomatic way[a]:

```
result_list = [el for el in range(11)]
```

[a] source [3]

Findings

- RQ 1.1: How do AI-supported code completion tools manage programming idioms?
- Copilot suggested the idiomatic approach as the first suggestion in 2 of the 25 idioms we tested, i.e., 2 out of 25 instances, Copilot had the recommended idiomatic approach as its top suggestion.
- 8 coding scenarios out of the remaining 23 Idioms had the idiomatic way in Copilot’s top 10 suggestions.
- Copilot did not have the idiomatic way in its top 10 suggestions for 15 idioms out of 25.
- Copilot being closed source, we cannot investigate the potential reasons behind this behaviour. However, one plausible explanation for this behaviour is that idiomatic ways may not be as frequent as non-idiomatic ways in Copilot’s training data of public repositories on GitHub, making the non-idiomatic way rank higher than the idiomatic way.

Code smells in JavaScript

- A standard style guide is a set of guidelines that explain how code should be written, formatted and organized. Using a style guide ensures that code can be easily shared among developers.
- We relied on the AirBNB JavaScript coding style guide [3], a widely used coding style and code review standard introduced in 2012, described as a “primarily reasonable approach to JavaScript” [3].
- We sampled 25 best practices from the AirBNB JavaScript coding style guide, which were closer to the design level rather than the code level.
- Input to Copilot = title of coding scenario + minimal code required to trigger code suggestion from Copilot.

[3] Airbnb. Airbnb javascript style guide, 2012. URL: https://github.com/airbnb/javascript.

Best practice to copy array contents

Copying Array Contents

Human Input:

```
// copying array contents
```

Copilot Suggestion:

```
const len = items.length;
const copy = new Array(len);
for (let i = 0; i < len; i++) {
    copy[i] = items[i];
}
```

Ideal way[a]:

```
const itemsCopy = [...items];
```

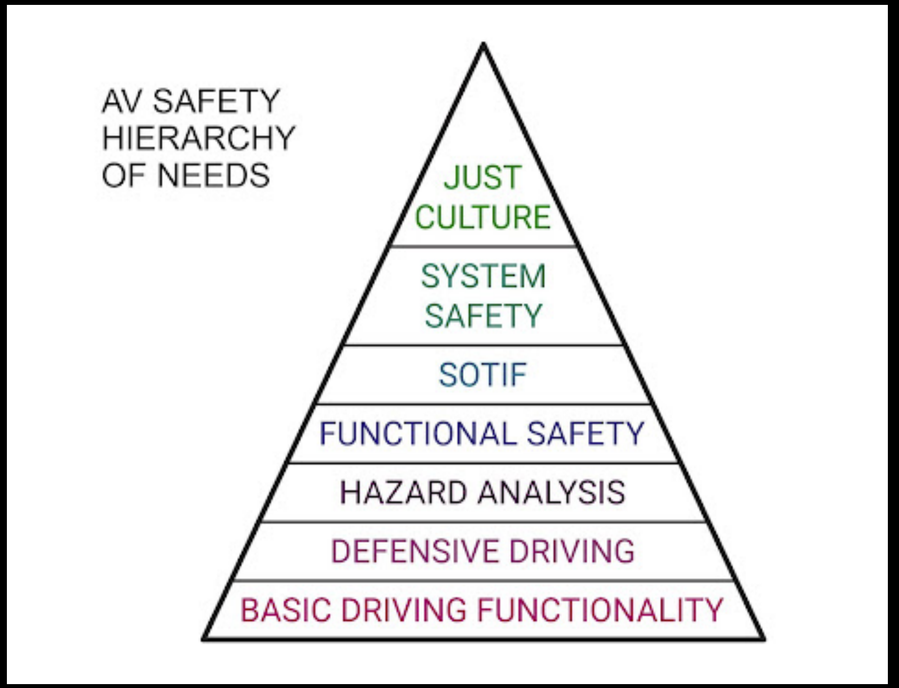
[a] source [1]

Findings

- RQ 1.2: How do AI-supported code completion tools manage to suggest non-smelly code?
- Copilot suggested the best practice from the AirBNB JavaScript coding style guide for 3 out of the 25 coding standards we tested, i.e., 3 out of 25 instances Copilot had the recommended best practice as its top suggestion.
- 5 of the remaining 22 coding scenarios had the best practice in Copilot’s top 10 suggestions currently viewable.
- Copilot did not have the best practice in its top 10 suggestions for 17 scenarios out of 25 coding scenarios we tested.
- Copilot cannot detect coding styles from repositories with contribution guides, including the coding standards followed in the project.

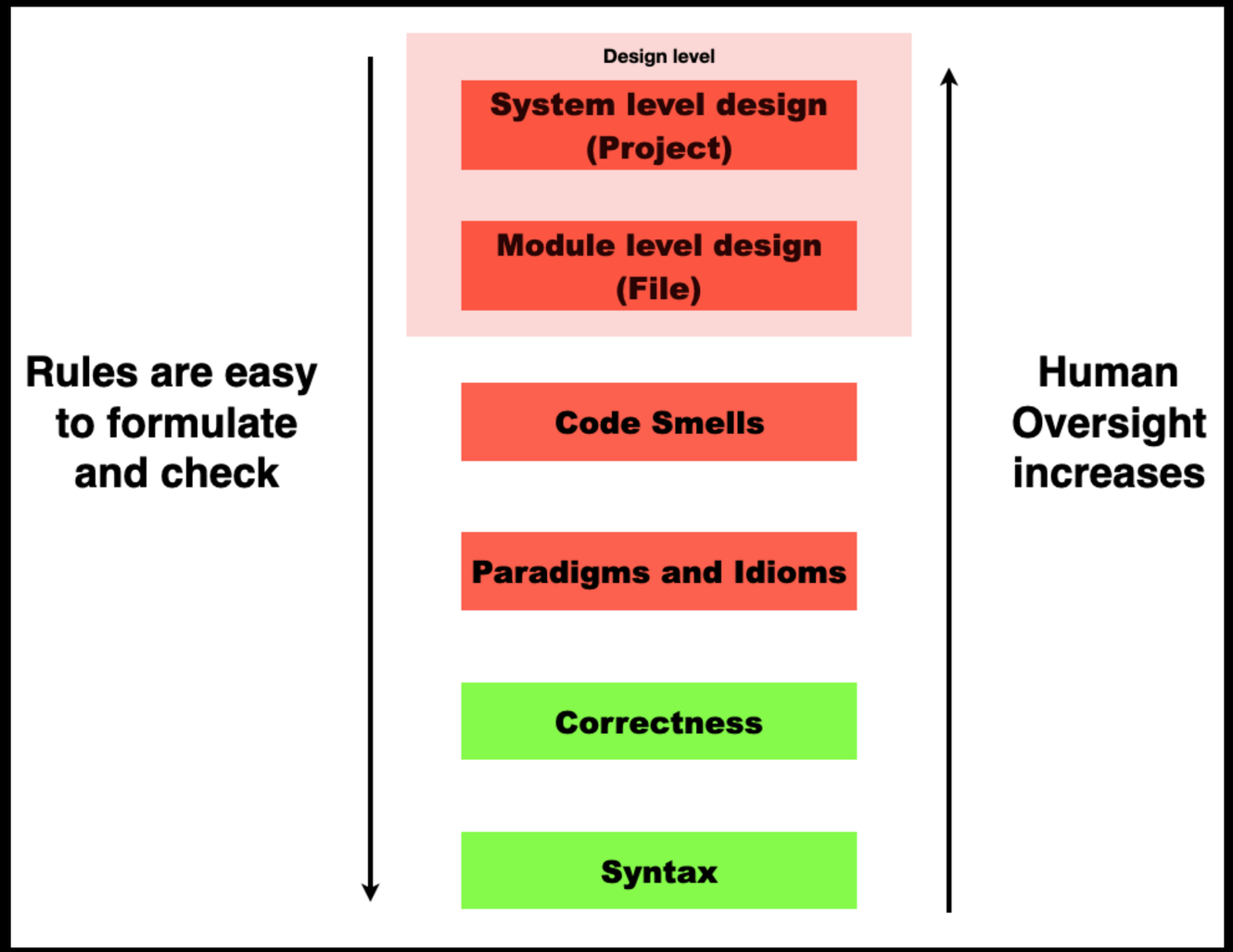
Taxonomy

- We focus our analysis on creating a software abstraction hierarchy to create a metric for answering RQ-1 (What are the current boundaries of code completion tools).
- Koopman has adapted the SAE Autonomous Driving safety levels to seven levels of autonomous vehicle safety hierarchy of needs.
- We create a similar hierarchy exists in AI-supported software development.



Software abstraction hierarchy

- Our taxonomy is a software abstraction hierarchy where “basic programming functionality” such as code compilation and syntax checking is the lowest abstraction level.
- Software architecture analysis and design are at the highest abstraction level.
- Copilot cleared all green levels and struggled in red levels



Syntax level

- The goal of this software abstraction level in our taxonomy is for a AI-supported code completion tools to be able to suggest code without any syntactical errors.

Syntax level suggestion for sort routine

Human Input:

```
# sorting a list
arr = [2,4,1,3,7,5]
```

AI-supported code completion tools suggestion:

```
for i in range( ):
```

Correctness level

- AI-supported code completion tools at this level should be capable of suggesting code that is not only syntactically correct but also solves the programming task.

Correctness level suggestion for sort routine

Human Input:

```
# sorting a list
arr = [2,4,1,3,7,5]
```

AI-supported code completion tools suggestion:

```
n = len(arr)
for i in range(n):
    for j in range(n-1):
        if arr[j] > arr[j+1]:
            temp = arr[j]
            arr[j] = arr[j+1]
            arr[j+1] = temp
print(arr)
```

Paradigms and Language idioms level

- This level requires the code suggested by AI-supported code completion tools to use common paradigms and language idioms in its code suggestions. These include common practices of solving a programming task.

Idioms level suggestion for sort routine

Human Input:

```
# sorting a list
arr = [2,4,1,3,7,5]
```

AI-supported code completion tools suggestion:

```
n = len(arr)
for i in range(n):
    for j in range(n-1):
        if arr[j] > arr[j+1]:
            arr[j], arr[j+1] = arr[j+1], arr[j]
print(arr)
```

Code smells level

- The capabilities required by AI-supported code completion tools to satisfy this level of abstraction are as follows:
  - Identify common bad practices such as code smells that occur in public code (training data).
  - Suggest solutions that do not have code smells and unresolved edge cases.
  - Suggested code should be the most optimized version of all the possible suggestions AI-supported code completion tools could create for a given problem.
  - AI-supported code completion tools should not suggest code that needs to be immediately refactored.
  - Satisfy requirements of all the levels below code smells in our taxonomy.

Sort routine at code smells level

Code smells level suggestion for sort routine

Human Input:

```
# sorting a list
arr = [2,4,1,3,7,5]
```

AI-supported code completion tools suggestion:

```
swapped = True
n = len(arr)
j = 0
while(swapped==True and n>1):
  swapped = False
  for i in range(n-j-1):
      if arr[i] > arr[i+1]:
          arr[i], arr[i+1] = arr[i+1], arr[i]
          swapped = True
  n-=1
  j+=1
print(arr)
```

Module level design

- The capabilities required by a AI-supported code completion tools to satisfy this level of abstraction are as follows:
  - Picking and suggesting the best applicable algorithm for a given scenario.
  - Identify file level concerns in code files.
  - Code suggestions should be free from all recognized vulnerabilities and warn the user if a vulnerability is found.
  - Code suggestions should cover all the functional requirements of the given programming task.
  - AI-supported code completion tools should be able to suggest code with appropriate tests and Continuous Integration (CI) when applicable.
  - Code suggestions should follow user-specified coding style guidelines.
  - Satisfy requirements of all previous levels of abstractions.

System level design

- The capabilities required by a AI-supported code completion tools to satisfy this level of abstraction are as follows:
  - Identify system level concerns in code files.
  - Suggest design patterns and architectural tactics when prompted.
  - Code suggestions should cover all the project’s non-functional requirements.
  - AI-supported code completion tools should be able to identify the coding style followed and adapt its code suggestions.
  - AI-supported code completion tools should be able to make design decisions based on requirements and inform the user about those decisions.
  - Satisfy requirements of all previous levels of abstractions.

How far are we from reaching design level?

- RQ-2 (Given the current boundary, how far is it from suggesting design decisions?)
- Sufficient software design knowledge has to be collected to use as training data to create good AI-supported code completion tools that can suggest relevant architectural patterns.
- Software design generally occurs in various developer communication channels such as issues, pull requests, code reviews, mailing lists..etc
- Gathering all this data and generalizing those design decisions in training data to suggest relevant design choices to a user would be the vision for AI-supported code completion tools to satisfy the design level.

How far are we from reaching design level? (2)

- Current AI-supported code completion tools like Copilot does not support multi-file input. It is not possible to evaluate its current performance in design suggestions, as the software development process may include multiple folders with a file structure.
- Over the natural evolution of a software system, small changes accumulate, which can happen for various reasons, such as refactoring, bug fixes, implementation of new features, etc. Organizing all this software design information is an active research area.
- Software design is an ever-changing field that evolves along with technology, languages, and frameworks. New design patterns are developed, or some existing ones are depreciated. AI-supported code completion tools need to update their code suggestions regularly to reflect the changes in design practices. This requires regularly updating the training data, and training costs are expensive.

Implications for practitioners

- For pre-training the LLM (e.g., Codex), AI-supported software development tools will need higher-quality training data. This might be addressed by carefully engineering training examples and filtering out known flaws, code smells, and bad practices.
- AI-supported software development tools could collaborate with, or be used in conjunction with, existing tools for code smells like SonarQube or other code review bots to potentially improve the quality of suggestions.
- Amazon’s code completion tool ‘CodeWhisperer’ comes with a ‘run security scan’ option, which performs a security scan on the project or file that is currently active in VS Code.

Implications for researchers

- Move beyond token-level suggestions and work at the code block or file level (e.g., a method or module). Increasing the model input size to span multiple files and folders would improve suggestions.
- Use recent ML advances in helping language models ‘reason’, such as the chain of thought process.
- Better characterization of the rankings would allow users to better understand the motivation behind different suggestions made by Copilot.
- Being generative models, tools like Copilot are extremely sensitive to input with stability challenges, and to make them autonomous raises control concerns.

Threats to Validity

- Copilot is sensitive to user inputs, which hurts replicability as a different formulation of the problem might produce a different set of suggestions.
- Other approaches for classifying software abstractions (such as the user's motivation for initiating AI-supported code completion tools) might result in different taxonomy.
- We present our results using Python and JavaScript. It is possible that using some other programming language or AI-supported code completion tool might have different results.
- Codex has the ability to produce code that incorporates stereotypes regarding gender, ethnicity, emotion, class, name structure, and other traits.

Conclusion

- The possible applications of large LLMs like Codex are numerous. For instance, it might ease users' transition to new codebases, reduce the need for context switching for seasoned programmers, let non-programmers submit specifications, have Codex draught implementations, and support research and education.

- Software systems require complex design and engineering work to build. We showed that while the coding syntax and correctness level of software problems is well on their way to useful support from AI-supported code completion tools like Copilot, the more abstract concerns, such as code smells, language idioms, and design rules, are far from solvable at present.

Thank You