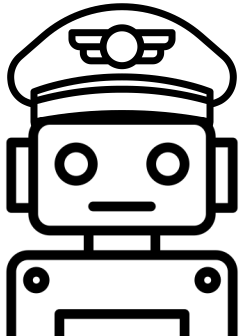
AI assistance systems in the digital humanities and research software engineering

Legacy code and copilot technologies: AI-supported software maintenance



Thesis

AI copilots force us to change our engagement with code in general.

- Possible **productivity gains** are huge → will shape the future division of labor
- The more the AI copilot “understands”, the more it will help us solve our problems and **change our problem solving** (e.g. switch → documentation performed by the consumer instead of the producer of code?)

AI copilots → the **end of prompt engineering**?

Enterprise AI Copilot - **changing everyday work life**

Structure

1. Background / motivation / news
2. Project work (Julia)
    a. Definition/Distinction of ChatGPT and AI copilots
    b. Enterprise AI Copilots
    c. Machine-supported documentation
3. Research Software Engineering - “AI copilots and software engineering” (Sebastian)
    a. How does Github Copilot work? - Why and how can it help us understand code?
    b. How can I use AI copilots for my problem solving / software engineering?
    c. Demos / examples for specific use cases.

Core questions

What is an AI copilot in contrast to other AI tools? (like ChatGPT)

- What are the intended use-cases?
- “From specialised to universal tools?” - Enterprise Copilots

Can AI copilots (github copilot) help us understand existing code?

- How do we understand code?
- How does an AI copilot “understand” code?

Motivation

- DERLA project (https://gams.uni-graz.at/derla)
- Problem: Future developers need to understand the code
  - Julia's work → document / share knowledge → using AI supported workflows.
- But:
  - Huge amount of legacy code (python, xslt, javascript etc.)
  - Lack of documentation
  - Pressure of commitment:
    - Expandability of the project
    - Additional data provided by project partners

DERLA | VERFOLGUNG UND WIDERSTAND IM NATIONALSOZIALISMUS DOKUMENTIEREN UND VERMITTELN

DIGITALE ERINNERUNGSLANDSCHAFT

News

- Publication of github copilot chat (from beta phase):
  - https://github.blog/2023-12-29-github-copilot-chat-now-generally-available-for-organizations-and-individuals/
- AI developer tools / different copilots
  - Github Copilot
  - Tabnine
  - CodeWhisper
  - https://medium.com/@aroshelova.tech/ai-developer-tools-copilot-tabnine-and-codewhisper-8dd052142ae3#:~:text=If%20you%20value%20real%2Dtime,Tabnine%20is%20a%20good%20option.
- More AI copilot systems / integrations:
  - Jetbrains IDEs
  - …

AI copilot - definition

An AI copilot is a conversational interface that uses large language models (LLMs) **to automate tasks and retrieve information.** By leveraging LLMs, copilots understand and respond to human language effectively, making it easier for users to interact with and navigate digital platforms.

https://www.moveworks.com/us/en/resources/blog/ai-copilot-strategy#first-things-first-what-is-an-ai-copilot

Examples:

- Github Copilot
- Microsoft's Copilot for Microsoft 365
- Jasper’s copywriting assistant
- Salesforce’s Einstein GPT

Definition - universal tool vs. specialised assistant

ChatGPT (“tools like” / “universal tools”)

- (built on top of LLMs)
- universal tool
- no specialised domain (from user’s perspective)
- excels in Natural Language Processing, therefore very applicable for everything text-related

AI Copilots (GitHub Copilot, Microsoft Copilot, …)

- (built on top of LLMs)
- specialised application(s)
- generalized for one domain
- (often) integrated in already used tools

https://support.microsoft.com/en-us/topic/chatgpt-vs-microsoft-copilot-what-s-the-difference-8fdec864-72b1-46e1-afcb-8c12280d712f
https://www.uctoday.com/unified-communications/chatgpt-vs-microsoft-copilot-the-major-differences/

Use Cases - ChatGPT vs. AI Copilot

ChatGPT

- Content Creation (“Write me a tweet about XYZ…”)
- Language Translation
- Informational Queries
- Educational Tool (“Explain like I’m five…”)

AI Copilot

- Coding assistant
- Documentation
- Integration with IDEs
- Automating mundane/repetitive tasks

https://redresscompliance.com/microsoft-copilot-and-chatgpt-a-comparative-analysis/

Enterprise AI Copilot

“One secure conversational interface for employees to take action, search for information, query data, receive notifications, and create content across hundreds of enterprise applications.”
https://www.moveworks.com/us/en/platform/enterprise-copilot

- enterprise = company setting
- single point of entry in work space
- one “catch-all” interface for all AI Copilot used

https://www.moveworks.com/us/en/resources/blog/ai-copilot-strategy

Tier-List for AI copilots

Tier I

- simple API requests against a LLM.
- no domain specific knowledge (no custom LLM)
- LLM as a service → OpenAI.
- **domain specialisation via Prompt Engineering.**

Tier II

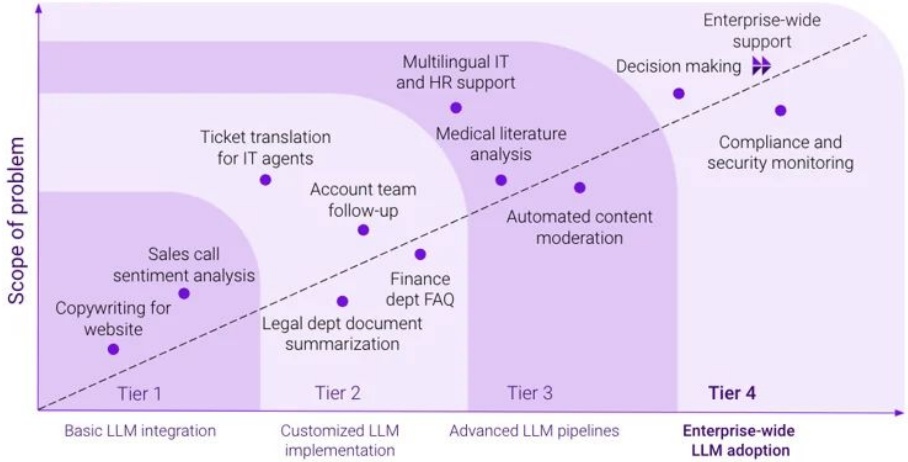
- fine tuned version of a LLM (fine-tuning + organisation owned data)
- **domain specialisation via fine tuning of the LLM** (+ prompt engineering), limited domain knowledge (only one LLM)

Tier III

- linking several LLMs → different specialisations → processing of multi-level use cases
- **domain specialisation via integration of various fine-tuned LLM pipelines**

Tier IV

- advanced LLM system specifically designed for company-wide use
- **domain specialisation** via enterprise wide **integration of different AI copilot systems**



https://www.moveworks.com/us/en/resources/blog/what-is-an-ai-copilot
https://www.moveworks.com/us/en/resources/blog/ai-copilot-strategy

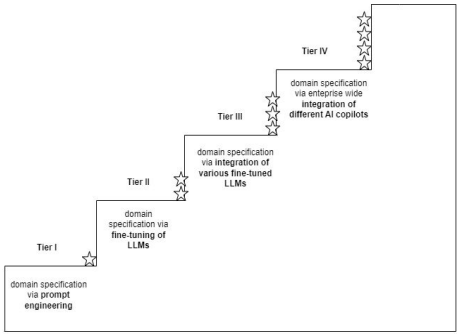
5 star model comparison - tier examples

Tier One
- generation of mails
- correction of style or grammar mistakes

Tier Two
- summarization of ZIM coding conventions
- writing a specific project proposal

Tier Three
- providing multilingual real-time support
- assisting in project proposal writing based on the available data without prompting

Tier Four
- analysis specific for ZIM structure and recommendations regarding institution wide decisions based on available data (very vague still)



(Promised) Benefits of Enterprise Copilots

- easier navigation and pooling of resources → quick access to information
- quick processing of routine requests
- utilizing full potential of existing technological systems
- seamless interaction with powerful back-end tools
- continuous learning, adaptation
- integration with various industry-specific tools such as Salesforce or Notion

→ increased efficiency, smooth information flow, new growth opportunities

- (potential) cost reduction by making previously necessary employees in support or employees who have worked on the banal activities that AI can now take over obsolete

→ What does this mean for the good majority of today's workforce, which would be rationalized away and cannot re-enter the workforce?

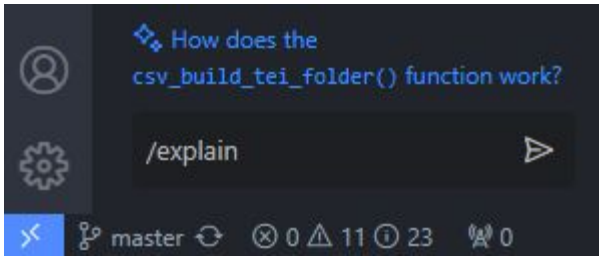
(Potential) Changes for Users

- simplified access to company data for employees
- no restriction to a specific communication channel (Slack, Microsoft Teams, mail, etc.)
- multilingual support
- backend systems can be connected across departments
- resource for employees → low-threshold to further training and development

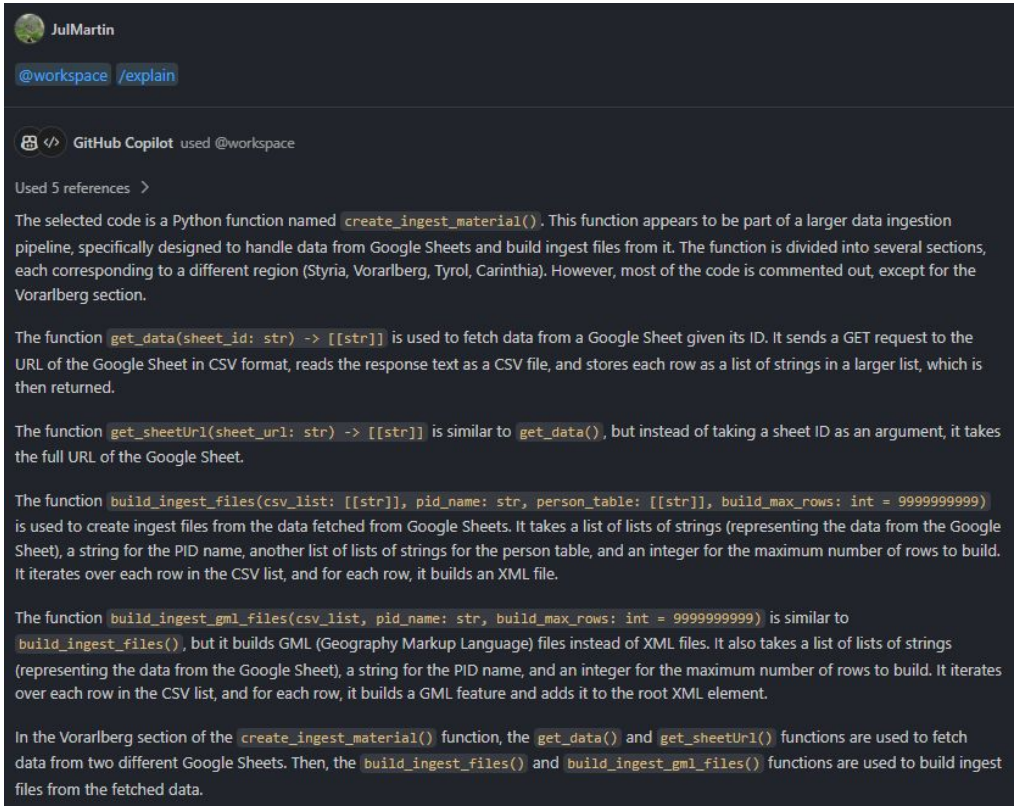
Machine Supported Documentation

Create documentation including explanations for DERLA legacy code:

- testing GitHub Copilots code explanation feature for documentation
- /explain
- /explain in simple terms
- /explain like I’m five
- /explain like I’m ten



/explain


JulMartin
@workspace /explain
GitHub Copilot used @workspace
Used 5 references
The selected code is a Python function named create_ingest_material(). This function appears to be part of a larger data ingestion pipeline, specifically designed to handle data from Google Sheets and build ingest files from it. The function is divided into several sections, each corresponding to a different region (Styria, Vorarlberg, Tyrol, Carinthia). However, most of the code is commented out, except for the Vorarlberg section.
The function get_data(sheet_id: str) -> [[str]] is used to fetch data from a Google Sheet given its ID. It sends a GET request to the URL of the Google Sheet in CSV format, reads the response text as a CSV file, and stores each row as a list of strings in a larger list, which is then returned.
The function get_sheetUrl(sheet_url: str) -> [[str]] is similar to get_data(), but instead of taking a sheet ID as an argument, it takes the full URL of the Google Sheet.
The function build_ingest_files(csv_list: [[str]], pid_name: str, person_table: [[str]], build_max_rows: int = 9999999999) is used to create ingest files from the data fetched from Google Sheets. It takes a list of lists of strings (representing the data from the Google Sheet), a string for the PID name, another list of lists of strings for the person table, and an integer for the maximum number of rows to build. It iterates over each row in the CSV list, and for each row, it builds an XML file.
The function build_ingest_gml_files(csv_list, pid_name: str, build_max_rows: int = 9999999999) is similar to build_ingest_files(), but it builds GML (Geography Markup Language) files instead of XML files. It also takes a list of lists of strings (representing the data from the Google Sheet), a string for the PID name, and an integer for the maximum number of rows to build. It iterates over each row in the CSV list, and for each row, it builds a GML feature and adds it to the root XML element.
In the Vorarlberg section of the create_ingest_material() function, the get_data() and get_sheetUrl() functions are used to fetch data from two different Google Sheets. Then, the build_ingest_files() and build_ingest_gml_files() functions are used to build ingest files from the fetched data.

Question suggested by copilot

JulMartin

How can I install the required dependencies for this project using pip?

GitHub Copilot

Used 1 reference >

To install the dependencies for a Python project using pip, you typically need a `requirements.txt` file in the project directory. This file lists all the Python packages that the project depends on.

However, the provided code does not include a `requirements.txt` file or any import statements, so it's not possible to determine the exact dependencies of this project.

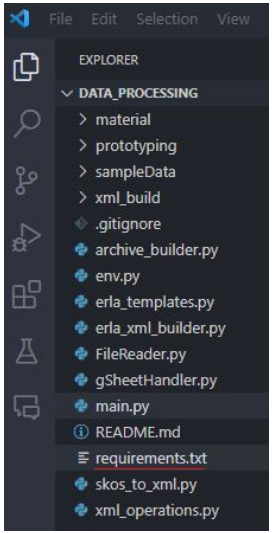
If you have a `requirements.txt` file, you can install all the dependencies with the following command in your terminal:

```
pip install -r requirements.txt
```

If you don't have a `requirements.txt` file, you'll need to manually install the packages that your project uses. For example, if your project uses the `requests` and `pandas` packages, you can install them with the following commands:

```
pip install requests
pip install pandas
```

Remember to replace `requests` and `pandas` with the actual packages that your project uses.



Provide context

Open files

Copilot looks at the current and open files in your editor to analyze the context and create appropriate suggestions. Having related files open in VS Code while using Copilot helps set this context and lets the Copilot see a bigger picture of your project.

https://code.visualstudio.com/docs/editor/github-copilot#_provide-context

/explain on a beginner level vs. /explain in simple terms

JulMartin

/explain on a beginner level

GitHub Copilot

Used 1 reference >

This Python script is part of a larger project that seems to be generating some sort of data files (possibly for data ingestion) for different regions (Styria, Vorarlberg, Tyrol, Carinthia).

The function `create_ingest_material()` is doing the following:

1. **Data Generation:** It's fetching data from Google Sheets documents. This is done using the `get_data()` and `get_sheetUrl()` methods of an object `g_handler`. The Google Sheets documents are identified by their unique IDs or URLs.
2. **Building Ingest Files:** After fetching the data, it's calling the `build_ingest_files()` and `build_ingest_gml_files()` methods of an object `erla_xml_builder`. These methods likely generate some sort of XML files for data ingestion. The methods take several parameters including the fetched data, a base name for the files to be generated, a table of person data, and a maximum number of rows to be built.

The majority of the code is commented out (lines starting with `#` are comments and are ignored by Python). Only the data generation and file building for Vorarlberg is actually being executed. The rest of the code for other regions (Styria, Tyrol, Carinthia) is not being run because it's commented out.

The `f"o:{PROJECT_ABBR}.vor"` is a formatted string in Python. It's creating a string that starts with "o:", followed by the value of `PROJECT_ABBR`, and then ".vor". This is likely creating a unique identifier for the data being ingested.

The `build_max_rows` parameter in the `build_ingest_files()` and `build_ingest_gml_files()` methods likely limits the number of rows of data that are processed.

JulMartin

/explain in simple terms

GitHub Copilot

Used 1 reference >

This Python script is part of a larger project that seems to be creating some sort of data files for different regions (like Styria, Vorarlberg, Tyrol, Carinthia).

The function `create_ingest_material()` is doing the following:

1. **Getting Data:** It's fetching data from Google Sheets documents. This is done using the `get_data()` and `get_sheetUrl()` methods of an object `g_handler`. The Google Sheets documents are identified by their unique IDs or URLs.
2. **Creating Files:** After fetching the data, it's calling the `build_ingest_files()` and `build_ingest_gml_files()` methods of an object `erla_xml_builder`. These methods likely create some sort of files using the fetched data.

The majority of the code is commented out (lines starting with `#` are comments and are ignored by Python). Only the data fetching and file creation for Vorarlberg is actually being executed. The rest of the code for other regions (Styria, Tyrol, Carinthia) is not being run because it's commented out.

The `f"o:{PROJECT_ABBR}.vor"` is creating a unique name for the data being created.

The `build_max_rows` parameter in the `build_ingest_files()` and `build_ingest_gml_files()` methods likely limits the number of rows of data that are processed.

/explain like I’m ten vs. /explain like I’m five

JulMartin

/explain like I'm ten

GitHub Copilot

Used 1 reference >

This Python script is like a toy factory. The factory is a function named `create_ingest_material()`. This function's job is to create some special toys (files) using parts (data) from different boxes (Google Sheets).

The function does the following:

1. **Getting Parts**: It's like opening a box and taking out toy parts. The function fetches data from Google Sheets documents. This is done using the `get_data()` and `get_sheetUrl()` methods of an object `g_handler`. The Google Sheets documents are like different boxes identified by their unique IDs or URLs.
2. **Creating Toys**: After fetching the parts (opening the boxes), it's calling the `build_ingest_files()` and `build_ingest_gml_files()` methods of an object `erla_xml_builder`. These methods are like the steps to assemble the toys using the parts.

Most of the code is commented out (lines starting with `#` are comments and are ignored by Python). It's like parts of the toy assembly instructions that the factory decided not to follow today. Only the parts fetching and toy creation for Vorarlberg is actually being done. The rest of the code for other regions (Styria, Tyrol, Carinthia) is not being followed because it's commented out.

The `f"o:{PROJECT_ABBR}.vor"` is creating a unique name for the toy being created.

The `build_max_rows` parameter in the `build_ingest_files()` and `build_ingest_gml_files()` methods is like a limit on how many toys to assemble.

JulMartin

/explain like I'm five

GitHub Copilot

Used 1 reference >

This Python script is like a toy factory. The factory is a function named `create_ingest_material()`. This function's job is to create some special toys (files) using parts (data) from different boxes (Google Sheets).

The function does the following:

1. **Getting Parts**: It's like opening a box and taking out toy parts. The function fetches data from Google Sheets documents. This is done using the `get_data()` and `get_sheetUrl()` methods of an object `g_handler`. The Google Sheets documents are like different boxes identified by their unique IDs or URLs.
2. **Creating Toys**: After fetching the parts (opening the boxes), it's calling the `build_ingest_files()` and `build_ingest_gml_files()` methods of an object `erla_xml_builder`. These methods are like the steps to assemble the toys using the parts.

Most of the code is commented out (lines starting with `#` are comments and are ignored by Python). It's like parts of the toy assembly instructions that the factory decided not to follow today. Only the parts fetching and toy creation for Vorarlberg is actually being done. The rest of the code for other regions (Styria, Tyrol, Carinthia) is not being followed because it's commented out.

The `f"o:{PROJECT_ABBR}.vor"` is creating a unique name for the toy being created.

The `build_max_rows` parameter in the `build_ingest_files()` and `build_ingest_gml_files()` methods is like a limit on how many toys to assemble.

Software maintenance with AI copilots

Fundamentals

- How do we understand software / code?
- How can AI systems understand software / code?

Prompt Engineering → Context!

Github copilot: Context (I)

https://github.blog/2023-05-17-how-github-copilot-is-getting-better-at-understanding-your-code/

"*To ensure that the model has the best information to make the best predictions with speed, GitHub's machine learning (ML) researchers have done a lot of work called* ***prompt engineering*** *(...) so that the* ***model provides contextually relevant responses*** *with low latency.*"

Github copilot: Context (II)

Contexts human developers use?

- Pull requests, folder in projects, issues / tickets, files, method names, filesystem context, metadata on IDE, …

- “Right now, transformers that are fast enough to power GitHub Copilot can process about 6,000 characters at a time (…) means that **not all of a developer’s code can be used as context** (…) So, our challenge is to figure out not only what data to feed the model, but also how to best order and enter it to get the best suggestions for the developer.”

  (https://github.blog/2023-05-17-how-github-copilot-is-getting-better-at-understanding-your-code/)

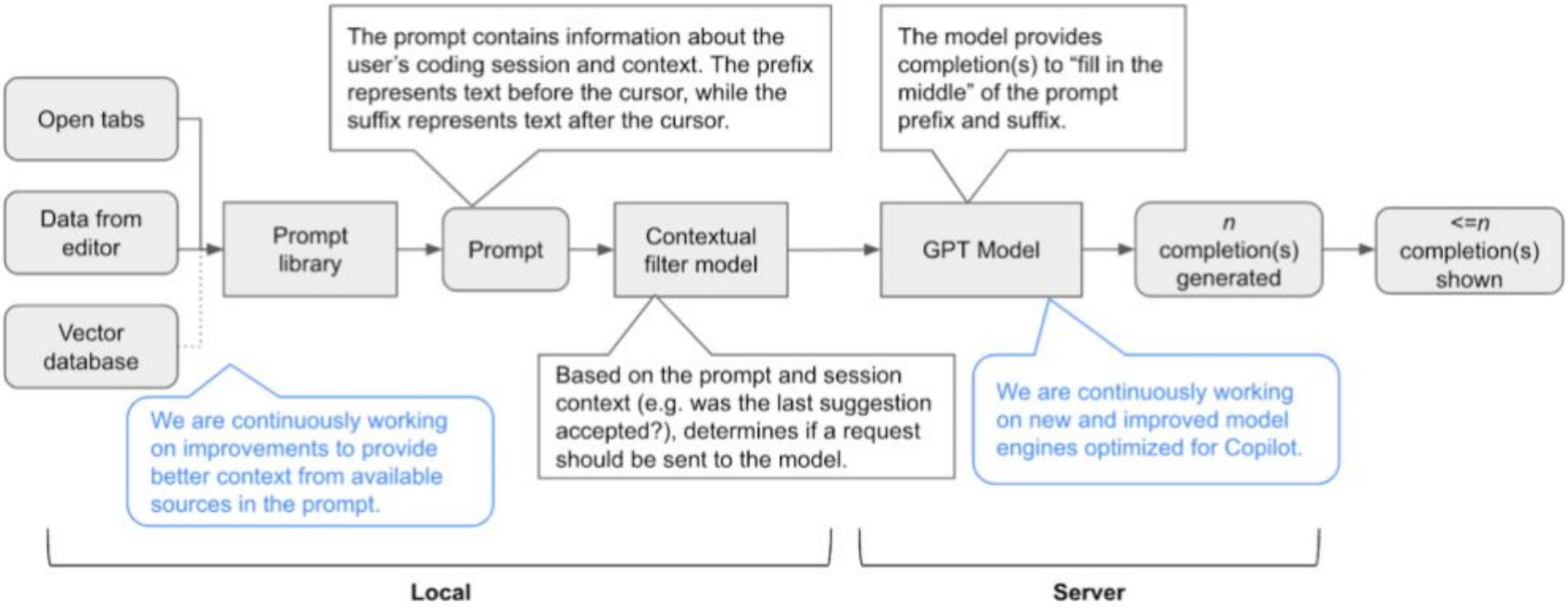
Github copilot: Context (III)

How GitHub Copilot understands your code → PROMPTS!

- **Prompts are generated at any time** in the background
  - "(...) series of algorithms first select relevant code snippets or comments from your current file and other sources (…) These snippets and comments are then **prioritized, filtered, and assembled into the final prompt**."

- **Prompt generating algorithms** first considered only the "current file" - but now:
  - 1. **Prompt Library** "(=which is where our in-house ML experts work with algorithms to extract and prioritize a variety of sources of information about the developer's context, creating the prompt that'll be processed by the GitHub Copilot model.)"
  - 2. **Neighboring Tabs**
    - "technique that allows GitHub Copilot to process all of the files open in a developer's IDE"
    - "By opening all files relevant to their project, developers automatically invoke GitHub Copilot to comb through all of the data and find matching pieces of code between their open files and the code around their cursor—and add those matches to the prompt."
  - 3. **Fill-In-the-Middle (FIM) paradigm**
    - Code before and after will is being included in the prompt

Life of a Completion



Simplified system diagram focused on model quality efforts. Made by Alice Li, machine learning researcher at GitHub.

Insights (I)

Fundamental → **Prompt Engineering**.

Need to understand **how prompts are being generated** within the AI copilot system / integrated environment.

Insights (IIa) - AI Copilot / Prompt Library

- **Prompt library**
  - Collection of elaborate, specialized prompts for the specific use cases. (generate documentation, generate code etc.)
  - Sub-prompts, partial prompts that can be used to automatically generate optimized prompts. (e.g. improve result via prepending “take a deep breath and answer step by step” to each generated prompt)
- **Code library**
  - Analyzing context (e.g. currently open files) to generate adequate prompts, using the prompt library.
  - Filtering, sorting, adding, removal of whitespaces, avoid unnecessary tokens etc.
    - E.g. filter out repetitions in a docstring, include implicit type information,

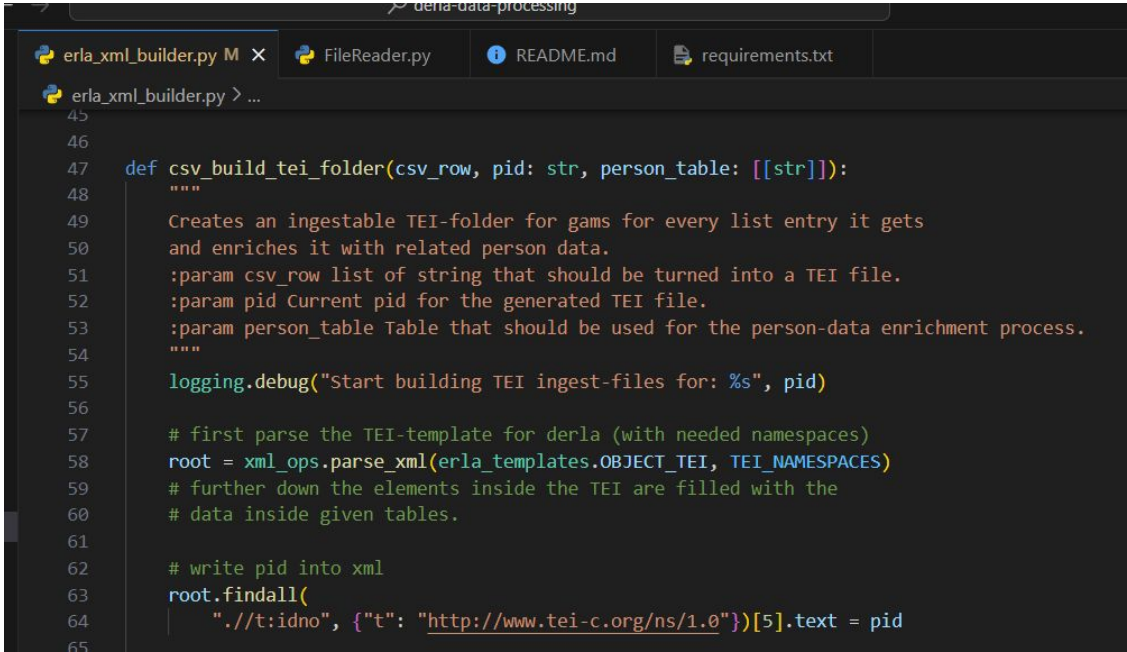
Insights (IIb) - AI Copilot / Prompt Library

- **Provide context** via comments, file-names, partly solved logic, etc.
  - … to improve generated prompts.

```
Creates an ingestable TEI-folder for gams for every list entry it gets
and enriches it with related person data.
:param csv_row list of string that should be turned into a TEI file.
:param pid Current pid for the generated TEI file.
:param person_table Table that should be used for the person-data enrichment process.
"""
logging.debug("Start building TEI ingest-files for: %s", pid)

# first parse the TEI-template for derla (with needed namespaces)
root = xml_ops.parse_xml(erla_templates.OBJECT_TEI, TEI_NAMESPACES)
# further down the elements inside the TEI are filled with the
# data inside given tables.

# write pid into xml
root.findall(
    ".//t:idno", {"t": "http://www.tei-c.org/ns/1.0"})[5].text = pid
```

Insights (IIIa) - Neighboring tabs / Github Copilot

- Open tabs → improve context of generated prompts.
- Result: “better” suggestions?

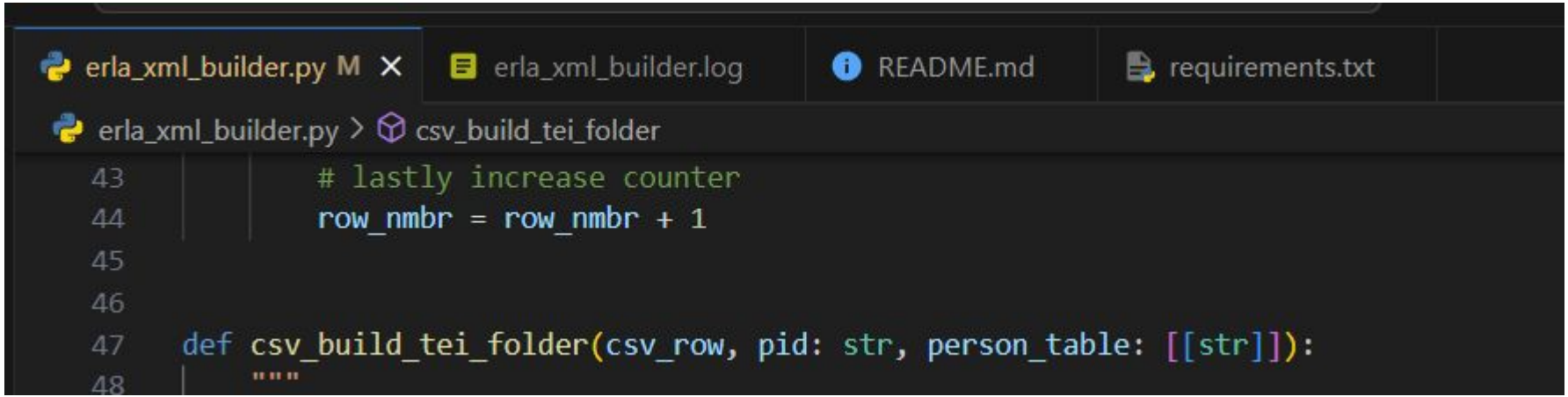


Insights (IIIb) - Neighboring tabs / Github Copilot

- Open files (in tabs) that include output of the code e.g. log files, related readmes, a result file etc.

Example:

- Code file + log-file + readme + requirements description.



RSE (I) - Effective engineering workflow?

Github copilot VS ChatGPT

ChatGPT → “top down”

- more generalized approach

AI copilot → “bottom up”

- why does the system recommend this library / concept?

Core question: Integration in software development tools?

- How can I use AI copilots tools effectively?

RSE (II) - Effective engineering workflow?

ChatGPT - “top down”

You

You are an well known expert for spring boot and the spring framework in general. You are an expert software engineer with decades of experience. Take a deep breath and answer step by step.

In my spring boot application I have the domain object "DigitalObject". When I'm requesting an overview over all digital objects - the json response will get really huge. I need to exclude / change the json when requesting a list overview to keep it small enough so that the application doesn't run out of memory. How would you approach this issue?

ChatGPT

When dealing with large JSON responses in a Spring Boot application, particularly when it involves domain objects that can lead to hefty payloads, it's essential to approach the issue methodically. Here's how you might address the need to minimize the size of your JSON responses when querying for an overview of all "DigitalObject" instances:

1. Identify the Essential Fields

Determine which fields in the "DigitalObject" are absolutely necessary for the list overview. Often, detailed information is not required at the overview level. You might need just an ID, name, or a few key attributes

RSE (III) - Effective engineering workflow?

Integration into project-management software

- Provide relevant context for AI copilots?
- E.g. gitlab, github, jira, …

Issues (support building of relevant contexts for AI copilots)

- publish ChatGPT chats
- include stackoverflow links
- provide links to documentation(s)
- Document insights / questions etc.

RSE (IV)

- include ChatGPT

  chats in the issue

Java OOM = Out of Memory problem in spring boot when ingesting a large amount of digital objects and datastreams.

Open Issue created 19 hours ago by **Stoff Sebastian**

DESC

- "Terminating due to java.lang.OutOfMemoryError: Java heap space"
- problem occured after adding the huge perssty SOLR json as datastream --> json somehow creates a huge memory impact --> when requesting all digital objects. // when to reduced to 10 digital objects -> works perfectly.
- application runs out of heap space during ingest and restarts after a while --> find out why
- occurs now also when requesting all digital objects!

INSIGHTS

- 💡 overview over all digital objects will create a **huge response (when json)** ~around 10.000 lines --> because I changed @JSONIdentity. Maybe include just the datastream-id when requesting all digital objects?
- 💡 might be a plain pagination problem
- 💡 is also a response data problem --> need to exclude extensive info about datastreams from the response --> otherwise 5 objects --> might result in very long json responses

LINKS

- https://iamvickyav.medium.com/spring-boot-dynamically-ignore-fields-while-converting-java-object-to-json-e8d642088f55
- chat gpt chat: https://chat.openai.com/share/cd042eaf-3de0-4a1b-acd8-1055f9fb56e7

TASKs

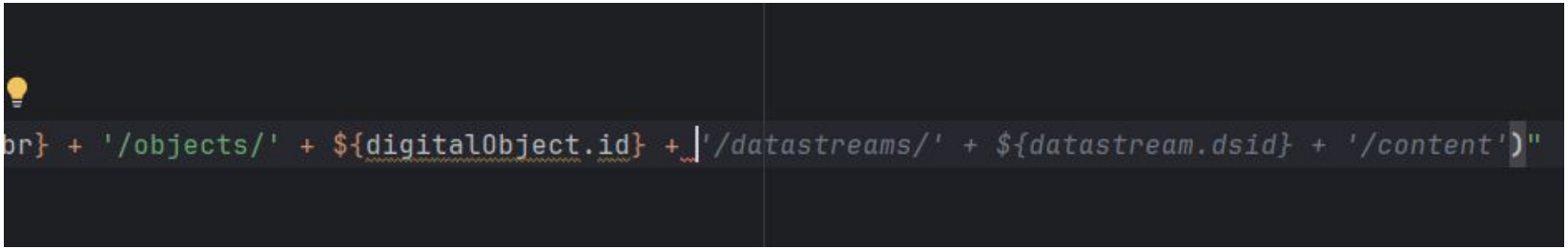
- [x] let datastream point only via id to the parent digital object
- [ ] possible solution: limit max page size of response? (e.g. 50 objects?)

Handling legacy code

- “Producing my own documentation” → consumer less dependent on the producer?
- None-code files relevant for AI copilot support:
  - reports: .log
  - test: CalculatorUnitTest.py
  - documentation: readme.md
  - …

AI copilot - context demo

- Copilot “understands” your REST-API (context of your project) - helps building requests accordingly.
- You do not need to provide extra context in comparison to a chat - your project is the context.



X-Tech - Scripting (I)

- Contrast to ChatGPT - use the current environment:
  - “This file …”, “... in my project folder …”, “... in this environment …”
  - “Why does this work?”, “How does this work?”

Demo

- “/generate I need a small python script that selects all the <graphic> elements in **this TEI file**. In the next steps rename all the IMAGE.1 and IMAGE.2 ids to IMAGE_1 and IMAGE_2.”

TEISIP.py | SIP.py | SOURCE.xml U

project > sips > vor3 > SOURCE.xml

```
43
44          <persName>
45
46              <forename>Sebastian</forename>
47
```

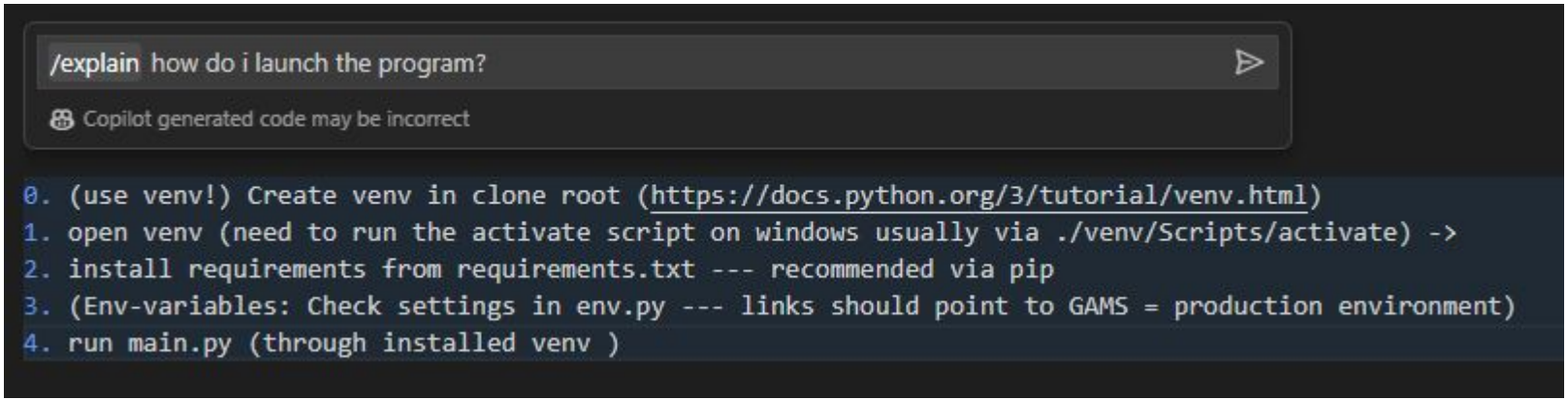
/generate I need a small python script that selects all the <graphic> elements in this TEI file. In the next steps it renames all the IMAGE.1 and IMAGE.2 ids to IMAGE_1 and IMAGE_2

Accept | Discard | Changed 19 lines

```
48              <surname>Stoff</surname>
49              import xml.etree.ElementTree as ET
50
51              # Parse the TEI file
52              tree = ET.parse('/c:/Users/sebas/Documents/programming/gams/gams5-client/project/sips/vor3/SOURCE.
53              root = tree.getroot()
54
55              # Find all graphic elements
56              graphics = root.findall('.//{http://www.tei-c.org/ns/1.0}graphic')
57
58              # Rename IMAGE.1 and IMAGE.2 ids
59              for graphic in graphics:
60                  if 'id' in graphic.attrib:
61                      if graphic.attrib['id'] == 'IMAGE.1':
62                          graphic.attrib['id'] = 'IMAGE_1'
63                      elif graphic.attrib['id'] == 'IMAGE.2':
64                          graphic.attrib['id'] = 'IMAGE_2'
65
66              # Save the modified TEI file
67              tree.write('/c:/Users/sebas/Documents/programming/gams/gams5-client/project/sips/vor3/SOURCE.xml')
68
69              <idno type="orcid">0000-0001-6941-113X</idno>
```

How do I use “this” code? (readme file)

- Mark sections of the readme → “explain this” with additional prompts
  - In VSCode: **mark text AND STRG + I + “explain”**



Stoff-Sebi

@workspace /explain how do i launch the program?

GitHub Copilot used @workspace

Used 1 reference >

The instructions provided in the README.md file outline the steps to launch the program. Here's a more detailed explanation:

1. **Create a virtual environment (venv)**: This is a self-contained environment where you can install Python packages without affecting your system's Python setup. You can create a venv in your project's root directory using the following command in your terminal:

```
python3 -m venv ./venv
```

2. **Activate the virtual environment**: This step is necessary to use the packages installed in the venv. The command to activate the venv depends on your operating system. On Windows, you can use:

```
./venv/Scripts/activate
```

On Unix or MacOS, use:

```
source venv/bin/activate
```

3. **Install the required packages**: Your project likely has a `requirements.txt` file, which lists the Python packages your project depends on. You can install these packages using pip, a package manager for Python:

```
pip install -r requirements.txt
```

4. **Set environment variables**: The `env.py` file likely contains settings that your project needs to run correctly. Make sure the settings in this file are correct for your environment.

5. **Run the program**: Finally, you can run your program. If your program's entry point is a file named `main.py`, you can run it with the following command:

Log messages

Use IDE integration

- Mark text + provide context aware question
- Use the project context! (“neighboring tabs” - github copilot)

Demo questions:

- What do these lines mean?
- Can you explain to me, what went wrong?
- Why is the code crashing after this log-message?

/explain what do these log messages mean?

Copilot generated code may be incorrect

```
DEBUG:root:Applying log file -- main.py called. Starting program
DEBUG:urllib3.connectionpool:Starting new HTTPS connection (1): docs.google.com:443
DEBUG:urllib3.connectionpool:https://docs.google.com:443 "GET /spreadsheets/d/1oL-6V0pCQ8CzbGMOFnURIXkvKb1Vj3QN5xW3zOz3Pgo/export
DEBUG:urllib3.connectionpool:Starting new HTTPS connection (1): doc-10-b0-sheets.googleusercontent.com:443
DEBUG:urllib3.connectionpool:https://doc-10-b0-sheets.googleusercontent.com:443 "GET /export/54bogvaave6cua4cdnls17ksc4/vvgbjg92u
DEBUG:urllib3.connectionpool:Starting new HTTPS connection (1): docs.google.com:443
DEBUG:urllib3.connectionpool:https://docs.google.com:443 "GET /spreadsheets/d/1oL-6V0pCQ8CzbGMOFnURIXkvKb1Vj3QN5xW3zOz3Pgo/export
DEBUG:urllib3.connectionpool:Starting new HTTPS connection (1): doc-10-b0-sheets.googleusercontent.com:443
DEBUG:urllib3.connectionpool:https://doc-10-b0-sheets.googleusercontent.com:443 "GET /export/54bogvaave6cua4cdnls17ksc4/vvgbjg92u
DEBUG:root:Start building TEI based ingest files for: 'o:derla.bur'. Only considering the first 300 rows.
DEBUG:root:Start building TEI ingest-files for: o:derla.bur3
WARNING:root:No value defined for col with number '0' at pid 'o:derla.bur3'. (Not even explicit none via '-')
INFO:root:Encountered empty title cell in places table - skipping generation of ingest files for pid (WILL NOT BE CREATED): o:der
INFO:root:Finished building of TEI-ingest files for: o:derla.bur3
DEBUG:root:Start building TEI ingest-files for: o:derla.bur4
DEBUG:root:Succesfully created term for spatial category: Stra�e
DEBUG:root:Finished building tei-xml for place: o:derla.bur4
DEBUG:root:Start victim enrichment process for pid: o:derla.bur4
DEBUG:root:Split victim row into: ['Zugeh�rige Orte']
DEBUG:root:Split victim row into: ['24', '14', '110', '']
```

Log messages

- Explain log messages

GitHub Copilot used @workspace

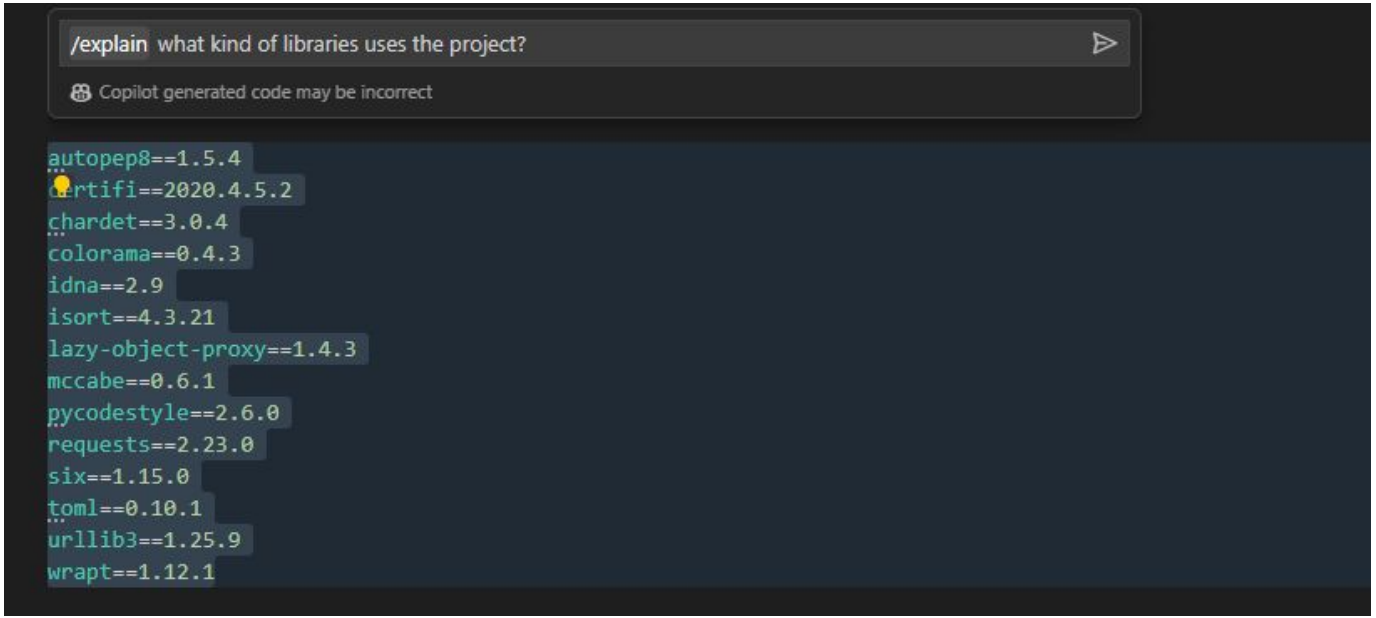
Used 1 reference

The log messages you provided are from a Python script that is interacting with Google Sheets to build TEI (Text Encoding Initiative) based ingest files. Here's a summary of what each log message means:

1. **Starting new HTTPS connection**: The script is establishing a new HTTPS connection to `docs.google.com` and `doc-10-b0-sheets.googleusercontent.com`. These are Google's servers where Google Sheets are hosted.
2. **GET /spreadsheets/... HTTP/1.1" 307 None**: The script is making a GET request to a specific Google Sheets document. The `307 None` response means that the requested resource resides temporarily under a different URI (Uniform Resource Identifier).
3. **GET /spreadsheets/... HTTP/1.1" 200 None**: The script successfully retrieved the Google Sheets document. The `200 None` response means the request was successful.
4. **Start building TEI based ingest files for: 'o:derla.bur'**: The script is starting to build TEI ingest files for the identifier 'o:derla.bur'. It's set to consider only the first 300 rows.
5. **Start building TEI ingest-files for: o:derla.bur3**: The script is starting to build TEI ingest files for the identifier 'o:derla.bur3'.
6. **No value defined for col with number '0' at pid 'o:derla.bur3'**: The script encountered an issue where there's no value defined for the column with number '0' at pid 'o:derla.bur3'. This is flagged as a warning.
7. **Encountered empty title cell in places table - skipping generation of ingest files for pid (WILL NOT BE CREATED): o:derla.bur3**: The script encountered an empty title cell in the places table. As a result, it's skipping the generation of ingest files for the pid 'o:derla.bur3'. This pid will not be created.
8. **Finished building of TEI-ingest files for: o:derla.bur3**: The script finished building the TEI ingest files for 'o:derla.bur3'.
9. **Start building TEI ingest-files for: o:derla.bur4**: The script is starting to build TEI ingest files for the identifier 'o:derla.bur4'.
10. **Succesfully created term for spatial category: Stra\ufffde**: The script successfully created a term for the spatial category 'Stra\ufffde'.
11. **Finished building tei-xml for place: o:derla.bur4**: The script finished building the TEI-XML for the place 'o:derla.bur4'.
12. **Start victim enrichment process for pid: o:derla.bur4**: The script is starting the victim enrichment process for 'o:derla.bur4'

Dependencies

- Use requirement files from software tools e.g. requirements.txt



GitHub Copilot used @workspace

Used 1 reference >

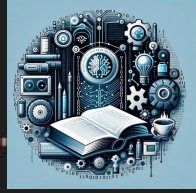
The `requirements.txt` file lists the Python libraries that your project depends on. Here's a brief description of each library:

1. **autopep8**: A tool that automatically formats Python code to conform to the PEP 8 style guide.
2. **certifi**: A Python package for providing Mozilla's CA Bundle.
3. **chardet**: A universal character encoding detector.
4. **colorama**: Makes ANSI escape character sequences work on Windows and Unix.
5. **idna**: Supports the Internationalized Domain Names in Applications (IDNA) protocol as specified in RFC 5891.
6. **isort**: A Python utility/library to sort imports alphabetically, and automatically separated into sections.
7. **lazy-object-proxy**: A fast and thorough lazy object proxy.
8. **mccabe**: A complexity checker for Python.
9. **pycodestyle**: A tool to check your Python code against some of the style conventions in PEP 8.
10. **requests**: An elegant and simple HTTP library for Python, built for human beings.
11. **six**: A Python 2 and 3 compatibility library.
12. **toml**: A library for parsing TOML files.
13. **urllib3**: A powerful, sanity-friendly HTTP client for Python.
14. **wrapt**: A module for decorators, wrappers and monkey patching.

These libraries are used for various purposes such as code formatting, HTTP requests, compatibility between Python 2 and 3, and more. The specific use of each library depends on the context of your project.

comments = writing code?

```
# sip_folder_path should not be None
if sip_folder_path is None:
    msg = f"SIP folder path must not be None."
    logging.error(msg)
    raise ValueError(msg)


# sip_folerpath should contain the os path separator
if os.path.sep not in sip_folder_path:
```



```
def _validate_sip(self, sip_folder_path: str) -> None:
    """
    Validates given sip folder path AND it's content.
    (Like must contain files + no subfolders)
    """

    # sip_folder_path should not be None
    if sip_folder_path is None:
        msg = f"SIP folder path must not be None."
        logging.error(msg)
        raise ValueError(msg)
```

Github Copilot Context - IIIF (I)

Context! → Copilot knows that I typically want to construct a related IIIF-URL from a base URL.

```
/**
 *
 */
const IIIFUtils = (() => {


    const getIIIFImageURL = (imageURL: string, width: number, height: number) => {


    return  {

    }

})();
```

Github Copilot Context - IIIF (II)

IIIF - API version?

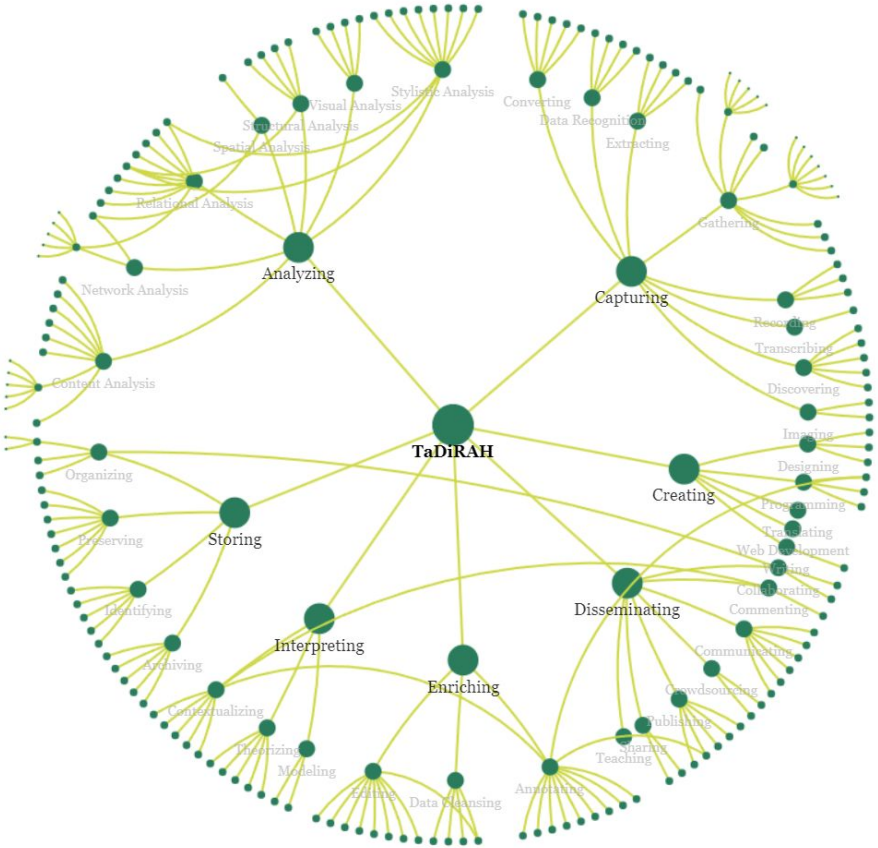
```
const getIIIFImageURL = (imageURL: string, width: number, height: number) => {
    return `${imageURL}/full/${width},${height}/0/default.jpg`;

}
```

THE TAXONOMY OF DIGITAL RESEARCH ACTIVITIES IN THE HUMANITIES

https://tadirah.info/pages/Browser.html

AI copilots supporting DH activities?

DH copilot?



Resources

- https://github.blog/2023-05-17-how-github-copilot-is-getting-better-at-understanding-your-code/
- https://github.blog/2023-04-07-what-developers-need-to-know-about-generative-ai/

- https://www.moveworks.com/us/en/resources/blog/what-is-an-ai-copilot
- https://www.moveworks.com/us/en/resources/blog/ai-copilot-strategy
- https://support.microsoft.com/en-us/topic/chatgpt-vs-microsoft-copilot-what-s-the-difference-8fdec864-72b1-46e1-afcb-8c12280d712f
- https://www.uctoday.com/unified-communications/chatgpt-vs-microsoft-copilot-the-major-differences
- https://redresscompliance.com/microsoft-copilot-and-chatgpt-a-comparative-analysis
- https://code.visualstudio.com/docs/editor/github-copilot#_provide-context