Software Artifact Summarizer

From the Developers Perspective

INDUSTRIAL SOFTWARE ENGINEERING
DEEPAK YADAV
CS23M105
IIT TIRUPATI

● What is the problem statement?

The Problem statement was to the software Arteficat Summarization
Later, the Problem was further Refined into software issues Summarization, and that too from the perspective of the Developer

● How have you solved the problem?

First Methodology
You Can Also Run the Jupyter Notebook to Summerize the Software Artifact Issues from the Point of View of a developer
 It will first Download the List of the links of all the issues of the link Provided by THE USER
 then

- We will Extract the Individual functional Components of the Git Issues.
- They are The Title of the issue
- Body of the issue
- Description of the issue
- We will then store them in different files with names, titles, body and Description.

- Then, by using the hugging face transformers and long-chain open API pipeline, we will summarise the individual summaries.
- This describes what the individual title says similarly to the key idea from the point of the developers in the body and description.
- We then merge all the files, summarise the entire document, and append the entire document summary to new_Readme.md file

While Handling the Text Summarization Part

->As Hugging Facettransformers Was unable to handle the very large files of data, to solve this issue, we divided the text files into chunks of smaller sizes, about 1000 tokens each, then we summerise individual segments and then concatenated the final Result

-> Also Experimented with the vector word Embedding method and then summarising the text
The Accuracy (Summarizing Capability) of the Model is Resinabely low as we need the mode, and the Avelabelity the Dataset to train the model is Resinabely low, so this Did’t come out to be a very great idea.

#Alternatively, I also Explore new Methodologies to Summerize the entire Software artefact
PyDocGen: Python Generator
It makes the complete Summery for the Python code, which includes different types of Summery, including code-summarizer, auto-generation of the requirements.txt file and code visualisation.

● Technical Information

-> Hugging Face Transformer
Hugging Face Transformers is a popular package for PyTorch and TensorFlow-based natural language processing (NLP) applications. It provides easy access to state-of-the-art pre-trained models and tools. Here are some critical points about Hugging Face Transformers:

Pre-trained Models: Hugging Face Transformers offers a wide range of pre-trained models for various NLP tasks, including translation, named entity recognition, text classification, and more.
Community and Platform: Hugging Face is an online community and platform focused on machine learning and NLP. The Transformers library is one of its flagship offerings, providing access to powerful pre-trained models2.
User-Friendly: The Hugging Face Transformer Library is known for its user-friendliness. It’s built on both PyTorch and TensorFlow, making it versatile and powerful for NLP tasks3.
Modalities Supported: Transformers support different modalities, such as text, vision, and audio. You can use these models for tasks

like text generation, image classification, and automatic speech recognition.

->LangCahain
LangChain is a framework designed to simplify the creation of applications using large language models. As a language model integration framework, LangChain's use cases largely overlap with language models in general, including document analysis and summarisation, chatbots, and code analysis.

● Techniques libraries used.

Hugging Face Transformers
Langchain large Language Model
Also Experimented with the Openapi GPT-3.5 MODEL

In the ALternative Method, while summarising the Entire Software we used
Django, React, Python(including a various other python modules) django djangorestframework django-cors-headers openai networkx pipreqs pygithub pylint

● Results (Output)

The Model was successful enough to summarise the entire Soteware and the issues of the software artefact on the git hub from the developer's point of view.

Summary
One model output

What is SAM?
Segment Anything Model (SAM) is a new AI model developed by Meta AI. Its standout feature is its ability to “cut out” any object in an image with a single click.
SAM is a promptable segmentation system that doesn’t require extensive training with labeled data. This means it can segment objects without the need for additional training, even for unfamiliar objects and images1.
Zero-Shot Generalization:
SAM has learned a general notion of what objects are. This understanding enables zero-shot generalization to unfamiliar objects and images without requiring additional training.
In other words, SAM can segment objects it hasn’t seen before, making it highly versatile1.
How SAM Works:
SAM takes input prompts specifying what to segment in an image. These prompts allow for a wide range of segmentation tasks without additional training.
You can prompt SAM with interactive points boxes, or even automatically segment everything in an image.
It generates multiple valid masks for ambiguous prompts, making it robust and flexible1.
Applications and Outputs:
SAM’s output masks can be used in various ways:
Tracked in videos
Enable image editing applications
Lifted to 3D
Used for creative tasks like collaging1.
Training and Data Engine:
SAM’s advanced capabilities come from training on millions of images and masks.
Researchers used SAM and its data to interactively annotate images and update the model. This cycle was repeated to improve both the model and the dataset.
The final dataset includes more than 1.1 billion segmentation masks collected from ~11 million licensed and privacy-preserving images1.
Efficiency and Flexibility:
SAM’s promptable design allows for flexible integration with other systems.
It’s efficient and groundbreaking in its ability to perform complex image segmentation tasks with unprecedented precision2.
If you’d like to explore SAM further, you can try the demo or read more about it in this Medium article3. Image segmentation is a fascinating field, and SAM is pushing the boundaries!

- What new things have you learned?
    - Tools
        - Hugginface Transformers
        - Openai API gpt-3
        - Langchain
        - GitHub API calls

    - New Summarization Tools and Techniques
    - How to sense the data that is collected from different numbers of (i.e. numerous sources when there is no common concise among them)
    - How to refine a problem in a Multiple Stepwise manner: Break a more significant development PROJECT into more minor tasks and subtask and solve them individually
    - I will call this method personally divide-and-conquer method of Software Development.

First Methodology
You Can Also Run the Jupyter Notebook to Summerize the Software Artifact Issues from the Point of View of a developer
 It will first Download the List of the links of all the issues of the link Provided by THE USER
 then

- We will Extract the Individual functional Components of the Git Issues.

- They are The Title of the issue
- Body of the issue
- Description of the issue
- We will then store them in different files with names, titles, body and Description.
- Then, by using the hugging face transformers and long-chain open API pipeline, we will summarise the individual summaries.
- This describes what the individual title says in a similar way to the key idea from the point of the developers in the body and description.
- We then merge all the files and summarise the entire document, and we also append the entire summary of the document to new_Readme.md file

While Handling the Text Summarization Part

->As Hugging Facettransformers Was unable to handle the very large files of data, to solve this issue, we divided the text files into chunks of smaller sizes, about 1000 tokens each, then we summerise individual segments and then concatenated the final Result

-> Also Experimented with the vector word Embedding method and then summarising the text
The Accuracy (Summarizing Capability) of the Model is Resinabely low as we need the mode, and the Avelabelity the Dataset to train the model is Resinabely low, so this Did’t come out to be a very great idea.

#Alternatively, I also Explore new Methodologies to Summerize the entire Software artefact
PyDocGen: Python Generator
It makes the complete Summery for the Python code, which includes different types of Summery, including code-summarizer, auto-generation of the requirements.txt file and code visualisation.

``` bash
git clone https://github.com/sanskarbharti/Summery_generator
```

Tech Stack used: Django, React, Python(including a various other python modules)

Installation and Running Backend:
Django
To install Django and run Backend, follow the steps below:
1. Change the directory to django_bk:
``` bash
cd django_bk
```

2. Install the following packages
``` bash
pip install Django django rest framework django-cors-headers open networkx pipreqs pygithub pylint
```

3. Change the directory to django_test:
``` bash
cd django_test
```
4. Start the backend using the following commands:
``` bash
python manage.py runserver
```

Installation and Running Frontend:
React
To install React and run Frontend, follow the steps below:
1. Start another terminal in the Summery_Generator directory
2. Install all the dependencies with:
``` bash
npm install
```
3. Install the axios dependency in case not installed using the following command:
``` bash
pip install axios
```
4. Start the Frontend with the following command:
``` bash
npm start
```

Django: https://docs.djangoproject.com/en/4.1/topics/install/
https://www.geeksforgeeks.org/django-introduction-and-installation/

Different python modules used includes ast, networkx, openai. We can install this modules using the pip install commands.

Different django-react modules used include axios, corsheaders. "npm install axios" can be used for the installation of the axios.

-> Also Experimented with using the openai gpt-3.5 model in that model we pass the entire git Repo to the openapi model and then we can chat with with the git Repo helping in summerizing the entire Git Repo if we Raise any command to summerize the git issues from the Developers Preaspective
This method worked Resinabely well beyond my Expecitaions as this was already trained on very Dataset So the Results were quite accurate