IA Générative pour le développement logiciel

Data collection and training

Data collection

Programming language: C

Extract code files from Github: Automate the process for multiple repositories to extract only files of a certain language.

Collection of Github issues and commits (Filter auto-generated text).

Data diversity/Deduplication: Training a model with duplicate data doesn't provide additional information; it's redundant and can lead to overfitting. (MinHash)

Personally identifiable information (PII) issue (name, email, username, IP, …).

Data collection and training

Training

A pre-trained model: StarCoderBase (available on HuggingFace) trained on 1 Trillion tokens.

Data Formatting: the process of structuring and organizing the input data in a specific way before feeding it into the model.

Tokenization: breaking up a piece of text into many pieces such as sentences and words. StarCoder uses the Hugging Face Tokenizers library.

```
# Example
java_code = "public class HelloWorld { public static void main(String[] args) { System.out.println(\"Hello, World!\"); } }"
result_tokens = tokenize_java_code(java_code)
print(result_tokens)
```

```
['utf-8', 'public', 'class', 'HelloWorld', '{', 'public', 'static', 'void', 'main', '(', 'String', '[', ']', 'args', ')', '{', 'System', '.', 'out', '.', 'println', '(', '"Hello, World!"', ')', ';', '}', '}', '', '']
```

Data collection and training

Training

Embeddings: A numerical representation for tokens (for example vectors). In general, they are used to match text to text and text to images. This representation captures the semantic meaning of what is being embedded. (In our case Java keywords)

An embedded dataset allows quick search, grouping and sorting…

We can use the Hugging Face Inference API which allows dataset embedding quickly with a POST request.

Prompt engineering

The inputs of the model depend on the task we want to generate:

- Function: We can provide the header of the function with a description of what it does or we can simply provide the documentation of the function.
- Script: We can provide a description of what it has to do.
- Class: We can provide the documentation of the class or a description of the class (attributes, methods, relationship with another class…)

LangChain

LangChain serves as a generic interface for LLMs, providing a centralized development environment to build LLM applications and integrate them with external data sources and software workflows.

Chatbots are one of the central LLM use-cases: Our code generator chatbot must be able to access some window of past messages directly (memory). So, we have to implement a conversational memory. In the context of LangChain, it is built on top of the ConversationChain.

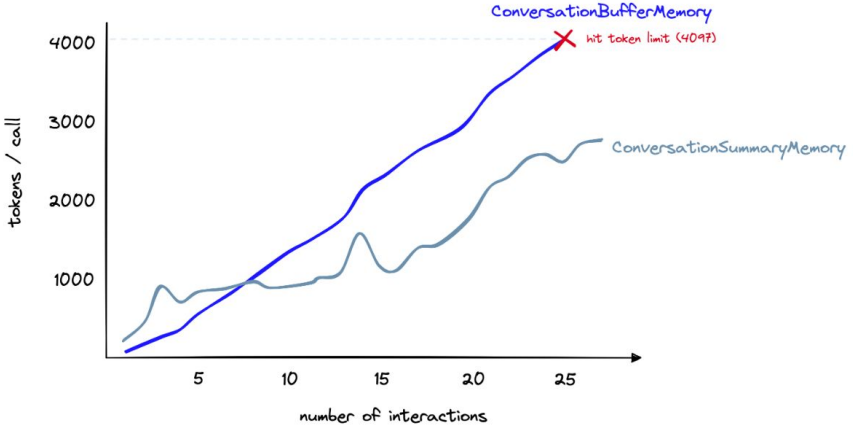
For that we have 2 possibilities that are available on LangChain:

- **ConversationBufferMemory** : this form of memory allows the LLM to clearly remember the history of the conversation.
- **ConversationSummarizeMemory** : this form summarizes the conversation history.

LangChain

The problem with the ConversationBufferMemory form is that if we use a lot of tokens, we can hit the LLM token limit (4096 for GPT-3.5 for example).

ConversationSummarizeMemory is a perfect solution to this problem. But, it is important to know that the memorization of conversation the history is reliant on the summarization ability of the intermediate summarization LLM.



Production Chain of our chatbot:

1st step: Getting the pre-trained Model, we will use stacoder for its availability, flexibility and numerous uses.

2nd step: Fine tuning the pretrained model.

3rd step: Integrating the resulting model to LangChain.

4th step-if possible: Integrating the whole system to a website.

Getting the Pre-trained Model:

An instance of use

Fine Tuning

Input	Output

Input: *Who was the 35th President of the United States?*

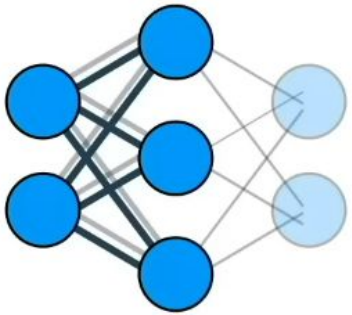
Output: *John F. Kennedy*

```
"""Please answer the following question.

Q: {Question}

A: {Answer}"""
```

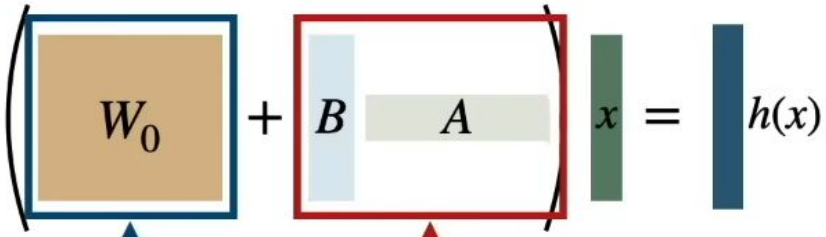
Low-Rank Adaptation (LoRA)

Fine-tunes model by adding new trainable parameters

$$h(x) = W_0 x + \Delta W x \qquad \Delta W = BA$$
$$= W_0 x + BAx$$



$x \rightarrow h(x)$



Frozen

Trainable

$d = 1{,}000$
$k = 1{,}000$
$r = 2$

$\Longrightarrow$ $(d \times r) + (r \times k)$ = **4,000 trainable parameters**

$W_0, \Delta W \in R^{d \times k}$

$B \in R^{d \times r}$

$A \in R^{r \times k}$

$h(x) \in R^{d \times 1}$

Fine-tuning with LoRA

```
peft_config = LoraConfig(task_type="SEQ_CLS", # sequence classification
                         r=4, # intrinsic rank of trainable weight matrix
                         lora_alpha=32, # this is like a learning rate
                         lora_dropout=0.01, # probablity of dropout
                         target_modules = ['q_lin']) # we apply lora to query layer
```

```
model = get_peft_model(model, peft_config)
model.print_trainable_parameters()

# trainable params: 1,221,124 || all params: 67,584,004 || trainable%: 1.8068239934
```

Integration into LangChain: