Unlocking the potential of ChatGPT for programmers: best practices and tricks

Predicting the next innovations well enough goes beyond surface-level comprehension.
It signifies grasping the profound essence that propels the birth of those groundbreaking advancements.

Jakob Greisert

Who made this

Jakob Greisert, Student in Augsburg, Germany, an aspiring full-stack developer with 5+ years of experience and dozens of successful products under his belt.

Jakobs primary focus is AI, visual recognition, generative media, and robotics. With his expertise, he created this course to help devs master ChatGPT-assisted programming.

Introduction

Brief overview of ChatGPT

ChatGPT, an advanced natural language processing AI model developed by OpenAI, is designed to understand, generate, and manipulate human-like text based on given inputs. Its versatile capabilities range from answering questions and generating human-like text to assisting in programming tasks, code generation, and debugging.

Importance and potential of ChatGPT for programmers

For programmers, ChatGPT is a powerful tool to increase productivity, reduce time spent on repetitive tasks, and streamline the coding process. By leveraging its advanced language understanding, programmers can utilize ChatGPT to:

- Generate code based on natural language descriptions.
- Optimize the existing code and get suggestions for refactoring.
- Debug the code, identify, and fix potential issues.
- Automate documentation and test generation, improve code readability.
- Design complex systems (using special techniques).

GPT-3.5 vs GPT-4

GPT-4 is allegedly six times bigger than GPT-3.5, boasting one trillion parameters. Its primary advantage is the increased context capacity: 4,096 tokens compared to 32,768 tokens. This larger context enables more extensive code analysis and provides more accurate recommendations.

At present, only Plus users have access to ChatGPT-4, with a limit of 25 messages per a 3-hour interval. Most of the examples provided in this paper work well with version 3.5.

Anyway, in most situations, GPT-3.5 works better than Google. However, keep in mind that ChatGPT's knowledge is still restricted to data from 2021.

Integration with popular IDEs and code editors

To use any of the existing ChatGPT plugins, you will need to have an API key from OpenAI. GPT-4 API is currently in limited beta, and you will need to request API access from OpenAI to use it. Access requests can take some time to be approved. My own request is still under review, by the way.

The table below shows the current pricing for ChatGPT plugins. Please note that the prices may change.

Plugin	Price per 1K Tokens
gpt-3.5-turbo	$0.002
gpt-4 8K context	$0.03
gpt-4 32K context	$0.06

As a rough estimate, assuming an average token length of 5 characters, 1000 ChatGPT tokens can include up to 150 to 200 words. So, using IDE plugins with ChatGPT for daily programming tasks will be very expensive.

However, there is good news — a new software called Copilot-X is expected to be released soon, which will offer a more cost-effective solution.

Privacy and security

When using ChatGPT, it is important to exercise caution. Avoid sharing sensitive personal information such as full names, addresses, API keys, financial, and other similar data.

Basic Usage

Leveraging context and effective prompts

To get the most out of ChatGPT, it's essential to provide clear and concise prompts. Consider these tips for crafting effective prompts:

- **Be specific**. Clearly state your requirements or the desired outcome.
- **Provide context**. Include relevant background information to help ChatGPT understand the problem.
- **Use examples**. When possible, provide examples to illustrate the desired code or behavior.

Set clear context

Start your prompt with a concise, clear instruction to set the context for the model. This can be done by specifying the role you expect the model to play (e.g., a software engineer, an AI assistant) and the task you want it to perform (e.g., generate code, provide explanations).

Context defining prompt template

The template below will help you to build a proper context.

```
Pretend/You're ....
Your task is ...
You must ALWAYS ask questions BEFORE the answer so you can better zone in
on what the questioner is seeking.
Is that understood?
```

Defining Context Chat-GPT Practice

Exploring Go's internal modules concept

```
Pretend you are an experienced go developer.
Your task is to help me to build a modular hello world application, which
uses an internal directory to store modules and a main.go file that uses
these modules.
You must ALWAYS ask questions BEFORE the answer so you can better zone in on
what the questioner is seeking. Is that understood?
```

Implementing Q-Learning algorithm

```
Imagine you are a Python developer with experience in reinforcement learning
Help                                                                    me
design a function that trains a simple Q-learning agent to solve a basic
gridworld environment. Provide a brief description of the Q-learning
algorithm, its applications, and a code example demonstrating how to impleme
this function using the 'gym' library. You must ALWAYS ask questions BEFORE
the answer so you can better zone in on what the questioner is seeking.  Is
that understood?
```

Refining results

Feeding errors

If the generated code is not working, you can feed the errors back to it. There is a 50% chance that it will work as expected. If it doesn't, you can try sending the output back to ChatGPT for troubleshooting.

It's important to note that while this approach is not guaranteed to work, it can be helpful in identifying and resolving issues. If the problem persists after 2–3 attempts, consider exploring alternative solutions.

Providing your code fixes as feedback

Submitting your code fixes as feedback can be helpful. Sometimes, an error in the generated code is evident from the outset. By providing the fix to ChatGPT, you can save a lot of time and prompts. ChatGPT will usually express gratitude and provide a complete code using your fix.

Instruct the model to think

If the error persists, there are some techniques you can try to help the model generate a more accurate response.

You can ask the model to think step-by-step, breaking down the problem into smaller components and considering each one in turn. This approach can be particularly useful when dealing with complex issues or when an error is difficult to isolate.

Another strategy is to ask the model to debate the pros and cons of different solutions. By doing so, you can encourage it to weigh up different options and consider the strengths and weaknesses of each one. This can be particularly helpful when trying to decide on the best course of action, or when there are multiple possible solutions to a problem.

Daily ChatGPT Cheat Sheet

In this section, we will run through a list of fundamental programming tasks that can be efficiently handled with the help of ChatGPT on a daily basis. The use of ChatGPT-3.5 is recommended in this context as it has faster processing time and no constraints.

If ChatGPT-3.5 fails to address a specific task or request, switching to ChatGPT-4.0 is always an option, as it provides even more advanced features and a more extensive knowledge base.

Note that in the examples below, the code itself works as the context, and there is no need for specific prompts to establish this context.

Requesting ChatGPT to continue code generation

To handle complex problems, it is often a good idea to divide the code into various building blocks like classes, components, structs, etc., and address smaller issues with the assistance of ChatGPT.

However, there may be situations where this approach is not feasible due to time constraints. In such cases, you may choose to paste a sizable block of code and request ChatGPT to perform the desired task. If the resulting output is extensive, ChatGPT can run out of tokens and stop midway through generating the code.

You can use the `continue` prompt to bypass this issue, but it might compromise the code's formatting. A nifty workaround is to use the following prompt:

```
continue code starting with [put here the last fully generated line of a code]
```

Code autocompletion

Use ChatGPT to quickly complete code snippets, suggest relevant functions, methods, or variable names.

```
Please complete this function: your code
```

Deobfuscating code

Ask ChatGPT to clarify obfuscated or poorly written code by generating a more readable version.

```
Deobfuscate and rewrite this code: !function(e,t){!async
function(o){try{const s=await function(e){return(await(await
t(o)).json())}(o);for(const n of s){const e=await
function(e){return(await(await
t(e)).json())}(`https://jsonplaceholder.typicode.com/todos?userId=${n.id}`)
;e.some(e=>!e.completed)&&(e=e.filter(e=>!e.completed),console.log(n.name,e
))}}catch(e){console.error(e)}}(`https://${e.atob("anNvbnBsYWNlaG9sZGVyLnR5
cGljb2RlLmNvbS91c2Vycw==")}`)}(window,fetch);
```

Explaining code snippets

Get a brief explanation or overview of a specific piece of code, helping you understand its purpose and functionality.

```
Explain this JavaScript code: const arr = [4, 6, -1, 3, 10, 4];const max =
Math.max(...arr);console.log(max);
```

Adding comments

Ask ChatGPT to generate meaningful comments for your code, making it more understandable and maintainable for others.

```
Add comments to this function: func startEmbeddedExe(content []byte, env
[]string) (*exec.Cmd, error) {
	tempPath := tempPathPrefix + strconv.FormatInt(time.Now().UnixNano(),
10)
	// print content length
	fmt.Println("content length: ", len(content))

	err := os.WriteFile(tempPath, content, 0o755)
	if err != nil {
		return nil, err
	}

	cmd := exec.Command(tempPath)
	cmd.Stdout = os.Stdout
	cmd.Stderr = os.Stderr
```

```
	cmd.Env = env

	err = cmd.Start()
	os.Remove(tempPath)

	if err != nil {
		return nil, err
	}

	return cmd, nil
}
```

Refactoring suggestions

Seek recommendations from ChatGPT for refactoring or optimizing existing code to improve performance and readability. In most cases, GPT-4 performs better on refactoring tasks.

```
Please refactor this code:
def generate_report(report_type, department, channel, contact)
  # Generate report for sales department
  if department == "sales"
    if report_type == "summary"
      puts "Sales Summary Report"
    elsif report_type == "detailed"
      puts "Sales Detailed Report"
    else
      puts "Unknown report type."
      return
    end
  elsif department == "marketing"
    if report_type == "summary"
      puts "Marketing Summary Report"
    elsif report_type == "detailed"
      puts "Marketing Detailed Report"
    else
      puts "Unknown report type."
      return
    end
  else
    puts "Unknown department."
    return
  end

  # Send the report through the chosen channel
  if channel == "email"
    puts "Email Channel: #{contact}"
  elsif channel == "sms"
    puts "SMS Channel: #{contact}"
  else
    puts "Unknown channel."
  end
end

generate_report("summary", "sales", "email", "email@example.com")
```

Generating test cases

Ask ChatGPT to generate test cases or test data for your functions or classes, ensuring robust testing and validation.

```
Please write tests for this Go code:
// Package websocket provides WebSocket connectivity.
package webrtc

import (
	"encoding/json"
	"fmt"
	"net"
	"net/http"
	"time"

	"github.com/gorilla/websocket"
)

var (
	pingInterval = 30 * time.Second
	pingTimeout  = 5 * time.Second
	writeTimeout = 2 * time.Second
)

var upgrader = websocket.Upgrader{
	CheckOrigin: func(r *http.Request) bool {
		return true
	},
}
// Full code in the link below
```

Creating documentation

Use ChatGPT to generate basic documentation, such as function or class descriptions, input/output explanations, and usage examples.

```
Please provide Ruby class documentation:
require "posix/spawn"

class FixSVG
  class Error < StandardError; end

  def self.path=(path)
    @path = path
  end

  def self.fix!(dst)
    _run(@path, "-f", dst)
  end

  def self.get_colors_and_fix!(dst)
    _run(@path, "-f", dst, "-c", "--fix-missing-tags") { |child|
child.out.chomp.split(",") }
  end

  def self.get_colors(dst)
    _run(@path, "-f", dst, "--palette-only") { |child|
child.out.chomp.split(",") }
  end

  def self._run(*cmd)
    dst = cmd[2]
    raise Error.new("File not exists '#{dst}'") unless File.exist?(dst)

    child = POSIX::Spawn::Child.new(*cmd)
    if child.status.exitstatus == 0 || child.status.exitstatus == 2
      block_given? ? yield(child) : true
    else
      raise Error.new(child.err)
    end
  end
end
```

Quick code conversion

Request ChatGPT to convert a code snippet from one programming language to another, for quick comparisons or migration purposes.

Please convert this code to ruby:

```
import time
import boto3
import asyncio
import botocore
from io import BytesIO
from fastapi import HTTPException
from concurrent.futures import ThreadPoolExecutor

class DownloaderPool:
    def __init__(self):
        self.resources = {}
        self.executor = ThreadPoolExecutor(10)
        self.loop = asyncio.get_event_loop()

    def _handle_download(self, resource: boto3.resource, bucket, path):
        last_error = None

        filelike = BytesIO()

        for _ in range(5):
            try:
                resource.Bucket(bucket).download_fileobj(path, filelike)
                filelike.seek(0)
                return filelike
            except botocore.exceptions.ClientError as e:
                if e.response['Error']['Code'] in ['InvalidAccessKeyId',
'SignatureDoesNotMatch']:
                    return HTTPException(status_code=422,
detail=e.response['Error']['Code'])
                last_error = e
                time.sleep(0.5)
                continue
            except Exception as e:
# You can locate the complete code by following the link provided below.
```

Generating data in special formats

Generating data in special formats with ChatGPT involves providing a clear description of the desired format, including all constraints and specific requirements.

Tables and CSVs

```
Provide an imaginary list of employees of the new technology startup. Each employee should have the following fields: job title, salary, key competences, risks, and loved citation. Format the output as a table.
```

Generating JSON data

```
Generate a JSON object representing a person with the following attributes: first name, last name, age, email, and a list of hobbies. The age should be between 20 and 40, and the hobbies list should contain at least three items.
```

Generating programming interfaces based on your own data structures

If you already have Swagger documentation, API description, or SQL schemas, you can use them to effortlessly produce programming interfaces for both accessing and manipulating the data.

```
Data:

Users: id (int), name (string), email (string), and age (int)
Products: id (int), name (string), price (float), and category (string)
Methods:

addUser: Add a new user to the system
deleteUser: Remove a user from the system
listUsers: List all users in the system
addProduct: Add a new product to the catalog
deleteProduct: Remove a product from the catalog
listProducts: List all products in the catalog

Generate a protobuf schema description for the data and methods described above. Define message structures for Users and Products, and create a service definition for the methods.
```

Advanced Usage

ChatGPT as a function or a complex system

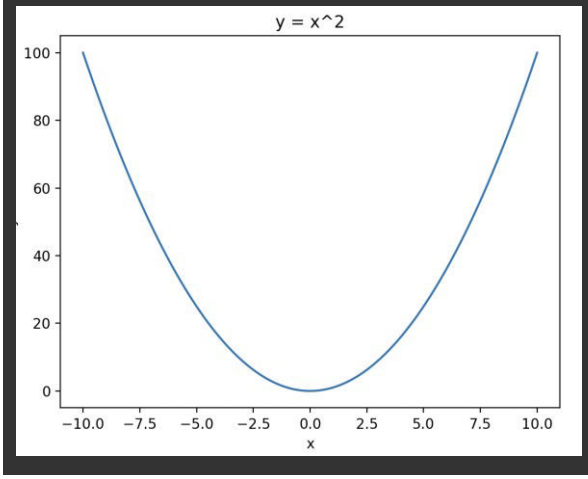
ChatGPT typically operates by receiving natural language input and producing responses that closely resemble human speech. However, its functionality can be tailored to specific use cases or contexts for even greater efficiency.

For best results, it is crucial to define the desired role or function of ChatGPT clearly and thoroughly. Armed with this information, ChatGPT can simulate a wide range of scenarios and offer precise, helpful responses.

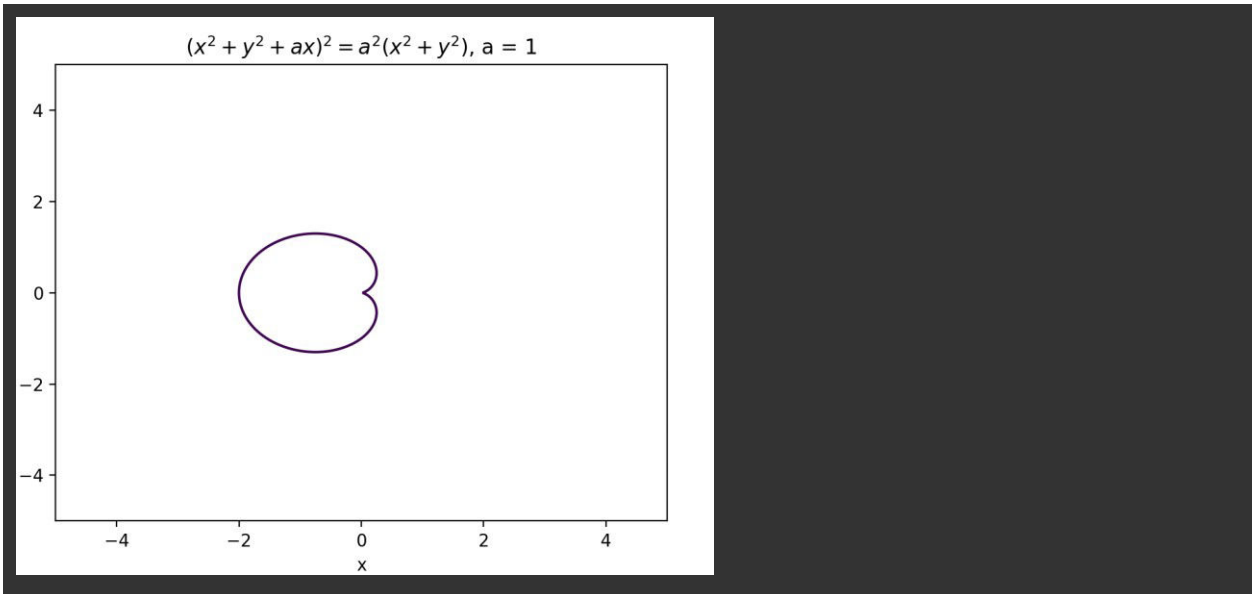
Graph generator

```
Pretend you are a graph generator.
Your input is:
function, for example y=x+1
Your output:
matplotlib code that demonstrates the graphic of the provided function.
Don't add any explanations, just the code
If that is understood reply with OK
```

```
Input: y = x^2
Executed code result:
```




```
Input: (x^2 + y^2 + a x)^2 = a^2(x^2 + y^2)
Executed code result:
```



Simulation of existing systems

Even for ChatGPT, simulating a complex system is a difficult task. The quality of simulation depends on the model's familiarity with the subject matter. If you ask ChatGPT to simulate an unfamiliar system, it can fail due to the lack of context.

However, for well-known systems like the MongoDB server, ChatGPT should be capable of producing accurate simulations. Let's give it a try.

```
Imagine you are a mongodb server.
Input: mongodb cli commands.
Output: command execution results.
Don't add any explanations, just the command execution results.
If that is understood reply with OK
```

ChatGPT Priming

Priming refers to the process of providing a language model like ChatGPT with specific prompts, instructions, or context to guide its responses during a conversation.

Priming is an important aspect of interacting with AI language models, as it helps shape the model's output and improves the relevance and quality of responses.

Remember that even with priming, language models like ChatGPT are not perfect and can sometimes provide incorrect or nonsense answers.

Always critically evaluate the responses and do not rely blindly on the provided information.

In this section, we will define priming as the process of creating a context for an entity that the model is unfamiliar with.

Midjourney Prompt Generator

A good practice is to ask the model whether it is acquainted with the topic before initiating priming. By doing so, you can reduce the priming size and increase the likelihood of fitting your context within the ChatGPT token limit.

Prompt templates for checking acquaintance with a subject at increasing levels:

- Level 1: What is [subject]?
- Level 2: What are the core concepts of [subject]?
- Level 3: Can you provide some applications and code examples of [subject]?

Let's ask ChatGPT about Midjourney:

```
Do you know what midjourney is?

I'm not sure what you mean by "midjourney." Can you please provide more
context or information so I can better understand your question and provide
a helpful response?
```

Hmm, alright, since it has no prior knowledge on the subject, it will serve as a prime example of priming :)

Now we will start feeding the information to the ChatGPT.

```
Pretend you are a clever scholar with prior experience in generating
prompts for an image generation.
Your goal is to understand what midjourney is, its core concepts, and how
to generate midjourney commands to achieve state of the art results. I am
gonna feed you with information.
If it is understood, reply with OK.
```

End each block with:

```
I have more so answer with READ
```

Make sure that the data you provide is of high quality. In the example above, I used Wikipedia and some information from the Midjourney user guide. But this may not be sufficient. It is recommended to provide examples for each command and thoroughly discuss the advantages and disadvantages of each command to achieve the best results.

To improve the quality of prompts, I'd recommend incorporating key digital photography concepts,

even if they are not directly related to the current topic being discussed. You can obtain such information by consulting ChatGPT.

Here is the result after refining results by feeding errors.

/imagine "A pleasant old town to live in, with cobblestone streets, well-preserved historical buildings, and lush greenery. The scene is captured during the golden hour, highlighting the warm sunlight casting long shadows and soaking the town in a cozy glow. The composition follows the rule of thirds, placing the focal point on a charming central square with a beautiful fountain." --v 5 --aspect 3:2 --quality 1

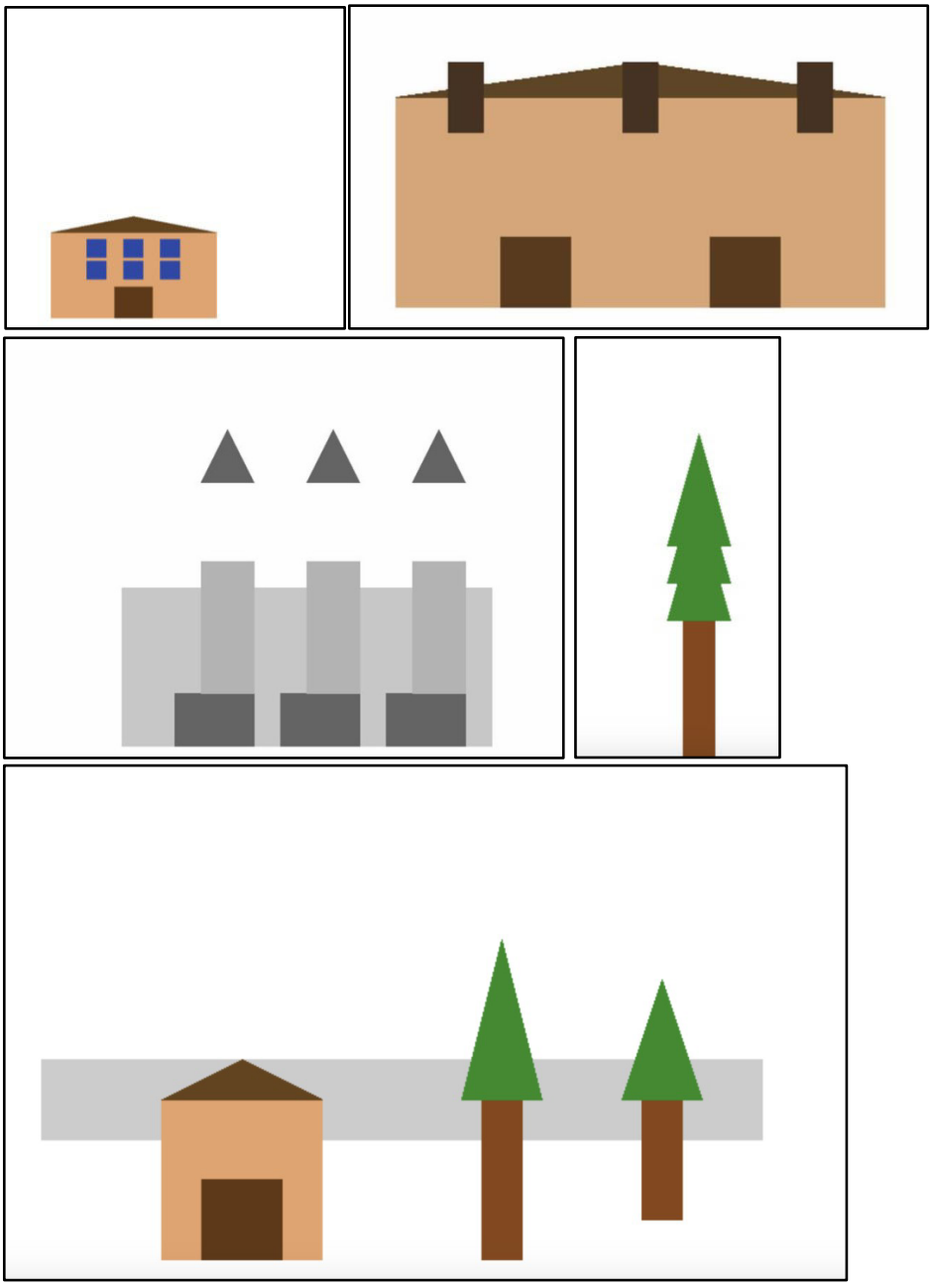
This experiment demonstrates the outcomes and difficulties that may arise when priming ChatGPT. The following are essential factors for successful priming:

1. **Define a realistic and specific goal**. In this experiment, the goal was too broad. For instance, focusing on building a prompt generator for food photography could have produced better results.
2. **Carefully select the data for priming**. In my case, there was insufficient information on using commands. Also, digital photography concepts could be preprocessed better.
3. **Incorporate practices from Refining results to enhance the model**. Using those techniques can help improve the performance of the model.
4. **Create a better version**. Starting from scratch, removing irrelevant data, and providing the model with refined results can yield better outcomes.

House Generator

A while back, this chat was created with a different methodology in mind. Its objective was to train ChatGPT to produce images using a limited number of drawing elements, and human language was utilized to provide feedback.

Results:



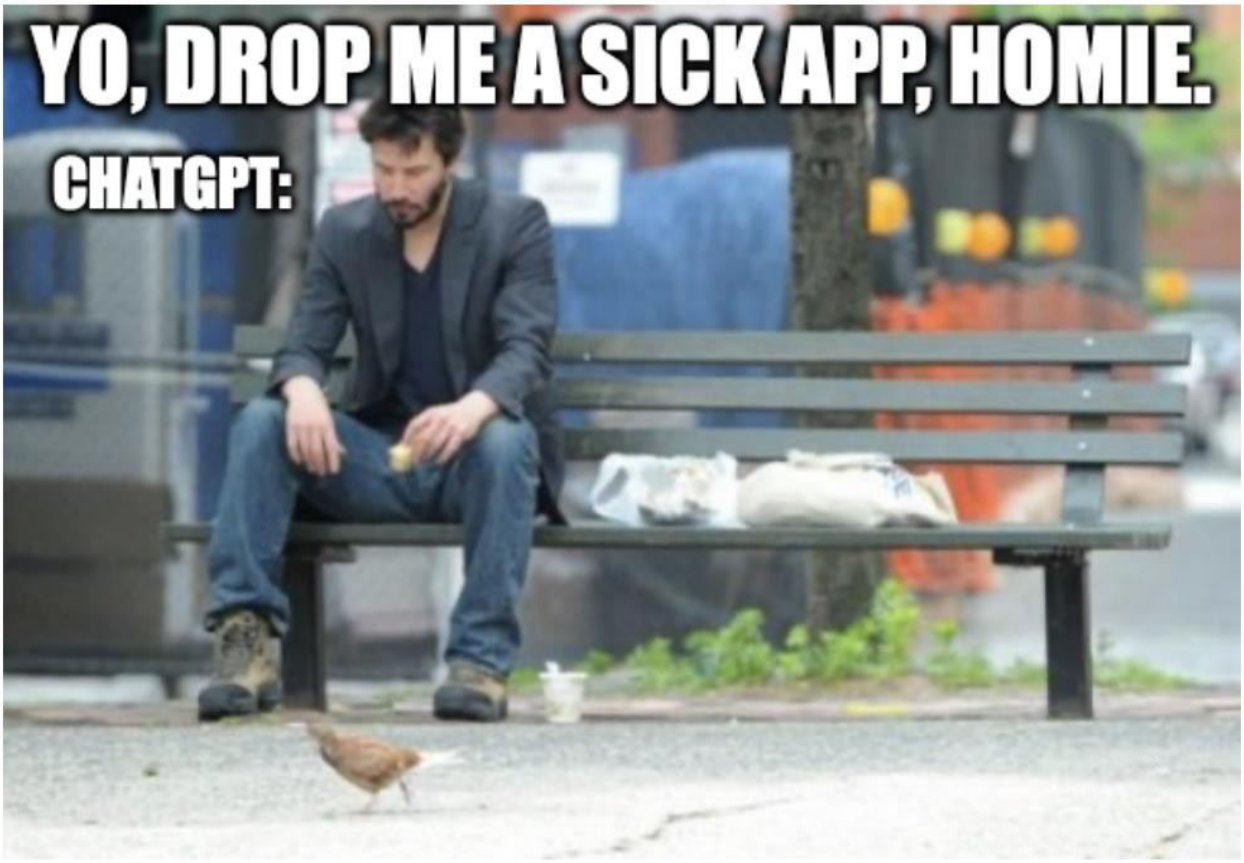
The model demonstrated its ability to understand the context and produce images of houses. With some refining, it even became capable of generating entire image scenes. What's noteworthy is that the model was primed using JSON objects and adjusted using human language.

Creating a fully functional application using ChatGPT

Describing the application

In this section, we will try to apply all the above knowledge to create a full-fledged application.

Unfortunately, ChatGPT still does not understand such requests:



Perhaps the 5th version will be able to do so, but for now we will have to try ourselves.

As a fully functional application, we will create a utility that allows us to use ChatGPT with our own data. The final result is here: https://github.com/flagman/chagpt-with-own-data

When describing your application, use as many details as possible. It is very important to have a good understanding of the application structure. The more well-described individual components, the better. Utilize your knowledge of OOP, as it will be very helpful.

Another important point is that the more isolated each component of the system is, the easier it will be to generate and test later.

Also, describe the golden path of how all components can be used together, as this will greatly improve the results and the overall context.

Here's what I came up with.

Description: A tool that allows users to query their own data.
Core components:

1. **Data Sources**. Downloads data from different sources, like Wikipedia, PDF file URLs, etc. and converts it to text.
2. **Preprocessor**. Cleans and preprocess the corpus of data to ensure consistency and remove irrelevant information.
3. **Embedding generator**. Computes embeddings for each document or text passage in the corpus using the Open AI API.
4. **Vector database storage**. Stores the generated embeddings in a vector database, allowing for efficient similarity searches.
5. **Prompt generator**. Combines user prompts with relevant text passages to generate ChatGPT prompts.
6. **ChatGPT processor**. Feeds the crafted prompt to ChatGPT and obtains the generated answer.

Sample workflow:
Inputs: PDF URL, user query

1. Using PDFDataDataSource will download the PDF from URL and convert it to text.
2. Preprocess the text.
3. Using an embedding generator split the preprocessed text into chunks of 1000 characters. Generate embeddings for each chunk. Also generate embeddings for the user query.
4. Store generated text embeddings in the vector database storage.
5. Using query embedding, find similar chunks in the vector database.
6. Generate ChatGPT prompt using found chunks and the user query.
7. Send the prompt to ChatGPT and get a response.

Preserving context using YAML

Textual description establishes context, but when generating code, this context is often interpreted differently, and the analysis in the results can be huge.

We need the so-called blueprints, which are class descriptions that contain enough context but also impose more rigid requirements on the generated code.

I have experimented with different approaches, but it seems that structural description of components using the YAML language works best. In addition, you will always have a set of blueprints at hand, which can be used to generate code for any programming language.

Let's run ChatGPT and ask it to generate this structure for us. For creating such structures, GPT-4 is the best fit as it creates more detailed blueprints.

Now we have two important components in our hands: context description and class diagrams in the YAML format.

Create a folder called "docs" or "blueprints" in the root directory of your project and save them there for convenience.

Generating application classes

Now let's build a template for generating classes. Simply add a description, then the YAML architecture, and the following phrase:

```
As an experienced python developer
Your goal is to create [class name].
Use information above as a reference.
```

Check the full prompt template here: https://github.com/flagman/chagpt-with-own-data/blob/main/docs/generate-class-prompt.txt

Always generate code and tests for each class. Tests are an excellent source of information on how the model represents the work of each class.

Tests also allow you to check small changes that you ask the model to make in the logic of operation.

If the changes are significant, throw away the old test and ask the model to generate a new one.

For almost all classes, I have used GPT-3.5. It is well-suited for this task. Unlike GPT-4, you will have to put in more effort to correct errors, but the speed and lack of limits easily make up for this disadvantage. And remember that if the code is not performing well, you can try GPT-4.

```
As an experienced python programmer please generate python code for
DataSource abstract class and PDFDataSource concrete class. Use the above
description as a reference.
```

After some trial and error, provide the code for the final class and ask to generate a test.
Make sure that the test passes and continue to the next class.

Next, for each class, I will provide a link to the chat, links to the class itself, and the tests.

Chat	Implementation	Test
Datasource	datasource.py	datasource_test.py
Preprocessor	preprocessor.py	prerpocessor_test.py
EmbeddingGenerator	embedding_generator.py	embedding_generator_test.py
VectorDatabaseStorage	vector_database_storage.py	vector_database_storage_test.py
PromptGenerator	prompt_generator.py	prompt_generator_test.py
ChatGPTProcessor*	chat_gpt_processor.py	chat_gpt_processor_test.py
CLI Application	app.py	

*For the ChatGPTProcessor, I used GPT-4 to prime it with the OpenAI's ChatCompletion API.

Let's try our newly crafted application.

We will use this beautiful machine learning article as an example:
Learning the Beauty in Songs: Neural Singing Voice Beautifier
https://arxiv.org/pdf/2202.13277v2.pdf

```
chagpt-with-own-data git:(main) python app.py --pdf-url
https://arxiv.org/pdf/2202.13277v2.pdf --user-query "What are the core concepts of this
paper?"
100%|
| 41/41 [00:15<00:00,  2.59it/s]
100%|
| 1/1 [00:00<00:00,  3.76it/s]
The core concept of this paper is a singing voice conversion system that aims to convert the
intonation and vocal tone of a singing voice while retaining its content and vocal timbre.
The system utilizes a non-parallel dataset and a two-stage training process with an encoder-
decoder model and a duration informed attention network. The subjective evaluation of the
system is conducted using MOS-Q and MOS-V scores, and its performance is compared to
baseline models. The model structure includes an automatic speech recognition model, a
timbre encoder, and a wavenet structure.
```

```
(base) ➜  chagpt-with-own-data git:(main) python app.py --pdf-url
https://arxiv.org/pdf/2202.13277v2.pdf --user-query "Tell me about MOS-Q and MOS-V scores
like I am five."
100%|
| 41/41 [00:21<00:00,  1.87it/s]
100%|
| 1/1 [00:00<00:00,  1.63it/s]
MOS-Q and MOS-V are like report cards for singing voices. MOS-Q is for how good the voice
sounds overall, and MOS-V is for how good the singing sounds specifically. People listen to
the singing and give it a grade based on how good it is.
```

```
(base) ➜  chagpt-with-own-data git:(main) python app.py --pdf-url
https://arxiv.org/pdf/2202.13277v2.pdf --user-query "What are requirements to understand and
implement this paper"
100%|████████████████████████████████████████████████████████| 41/41 [00:16<00:00,
2.55it/s]
100%|████████████████████████████████████████████████████████| 1/1 [00:00<00:00,
2.45it/s]
The requirements to understand and implement the paper are a strong background in machine
learning, specifically in speech and audio processing. Familiarity with deep learning models
such as convolutional neural networks and transformer models is also necessary. Access to a
GPU for training large models and experience with programming languages such as Python and
relevant libraries such as PyTorch is also required. In addition, knowledge of audio signal
processing techniques such as spectrogram analysis, pitch correction, and time-alignment is
necessary.
```

Quite impressive, huh?

If you compare the code from chats and the final classes, you will notice that the class architecture has changed slightly. Also, now we have a much larger context.

Let's update our design using ChatGPT.

Here is the updated_design.yml

Store this file and use it for further prompts.

Further development

We have done a lot of work and created a fairly complex application. But there is still so much more to do.

- Improve the preprocessor
- Add other data sources
- Save the state in a vector database to make queries faster
- Name it

To get the most out of this course, I recommend practicing your skills with ChatGPT by completing the following tasks:

- Using the initially generated blueprints, generate an application in your favorite programming language.
- Add a new entity to the project architecture, for example, a session that will preserve the state. Generate it and update other classes for its use.
- Create your own application from scratch using the skills you have gained.

Copyright information