

Developing the Future with ChatGPT



In this chapter, we will discuss how developers can leverage ChatGPT. The chapter focuses on the main use cases ChatGPT addresses in the domain of developers, including code review and optimization, documentation generation, and code generation. The chapter will provide examples and enable you to try the prompts on your own.

After a general introduction about the reasons why developers should leverage ChatGPT as a daily assistant, we will focus on ChatGPT and how it can do the following:

Why ChatGPT for developers?

Generate, optimize, and debug code

Generate code-related documentation and debug your code

Explain **machine learning (ML)** models to help data scientists and business users with model interpretability

Translate different programming languages

By the end of this chapter, you will be able to leverage ChatGPT for coding activities and use it as an assistant for your coding productivity.

Why ChatGPT for developers?

Personally, I believe that one of the most mind-blowing capabilities of ChatGPT is that of dealing with code. Of any type. We've already seen in [Chapter 4](#) how ChatGPT can act as a Python console. However, ChatGPT capabilities for developers go way beyond that example. It can be a daily assistant for code generation, explanation, and debugging.

Among the most popular languages, we can certainly mention Python, JavaScript, SQL, and C#. However, ChatGPT covers a wide range of languages, as disclosed by itself:

V list all the programming languages you can understand and generate

As an AI language model, I can understand and generate code in many programming languages. Here are some of the most popular ones:

1. Python
2. Java
3. C++
4. JavaScript
5. Ruby
6. PHP
7. Swift
8. Kotlin
9. TypeScript
10. C#
11. Go
12. Rust
13. R
14. MATLAB
15. Lua
16. SQL
17. Bash
18. Perl
19. Objective-C
20. Scala
21. F#
22. Haskell
23. Dart
24. Groovy
25. PowerShell
26. Visual Basic
27. COBOL
28. Fortran
29. Assembly

This is not an exhaustive list, but it includes many of the most commonly used programming languages.

Regenerate response

Figure 6.1 – ChatGPT lists the programming languages it is able to understand and generate

Whether you are a backend/frontend developer, a data scientist, or a data engineer, whenever you work with a programming language, ChatGPT can be a game changer, and we will see how in the several examples in the next sections.

From the next section onward, we will dive deeper into concrete examples of what ChatGPT can achieve when working with code. We will see end-to-end use cases covering different domains so that we will get familiar with using ChatGPT as a code assistant.

Generating, optimizing, and debugging code

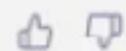
The primary capability you should leverage is ChatGPT code generation. How many times have you been looking for a pre-built piece of code to start from? Generating the `utils` functions, sample datasets, SQL schemas, and so on? ChatGPT is able to generate code based on input in natural language:

V

generate a python function that add a line to all my csv files saved within "my_folder". The line should be the name of the csv file.



Here's a Python function that should do what you're looking for:



```
python

import os
import csv

def add_line_to_csv_files(folder_path):
    # loop through all the files in the folder
    for file_name in os.listdir(folder_path):
        if file_name.endswith('.csv'):
            file_path = os.path.join(folder_path, file_name)
            # open the file in append mode
            with open(file_path, 'a', newline='') as csv_file:
                writer = csv.writer(csv_file)
                # write the file name as a new row in the CSV file
                writer.writerow([file_name])
```

Copy code

This function takes a folder path as an argument and loops through all the files in the folder. If the file has a ".csv" extension, the function opens it in append mode and uses the csv module to write the file name as a new row in the CSV file.

You can call this function like this:

```
python

add_line_to_csv_files('my_folder')
```

Copy code

Make sure to replace "my_folder" with the actual path to your folder.

Figure 6.2 – Example of ChatGPT generating a Python function to write into CSV files

As you can see, not only was ChatGPT able to generate the function, but also it was able to explain what the function does, how to use it, and what to substitute with generic placeholders such as **my_folder**.

Another example could be of creating schemas for structured tables. Namely, imagine you are a data engineer in the process of creating relationships between two standard tables in an SAP ERP system: **MARA** and **VBAP**, storing, respectively, general material data and sales document data.

Let's ask ChatGPT to create the schema for these tables by giving it the names of the columns as context:

V

I have to SAP tables with the following columns.

MARA table: MATNR ERSDA ERNAM AENAM AEDAT LAEDA LVORM PRDHA STPRS
PEINH BWTAR MEINS BSPGE BKLAS SERNP SOBSL SPART MBRSH MTPOS_MARA

VBAP table: VBELN POSNR MATNR KDMAT VRKME LFIMG WMENG VRKMG
NETWR WAERK KZWI1 KZWI2 KZWI3 KZWI4 KZWI5 FKIMG NTGEW BRGEW GEWEI
VOLUM

generate a SQL schema for the above VBAP and MARA SAP table making relationships



Based on the given columns, the SQL schema for the MARA and VBAP tables can be defined



```
scss                                     Copy code

CREATE TABLE MARA (
    MATNR VARCHAR(18) NOT NULL,
    ERSDA DATE,
    ERNAM VARCHAR(12),
    AENAM VARCHAR(12),
    AEDAT DATE,
    LAEDA DATE,
    LVORM CHAR(1),
    PRDHA CHAR(2),
    STPRS DECIMAL(13, 2),
    PEINH DECIMAL(13, 3),
    BWTAR CHAR(4),
    MEINS CHAR(3),
    BSPGE DECIMAL(13, 2),
    BKLAS CHAR(9),
    SERNP CHAR(10),
    SOBSL CHAR(1),
    SPART CHAR(2),
    MBRSH CHAR(2),
    MTPOS_MARA INTEGER,
    PRIMARY KEY (MATNR)
);

CREATE TABLE VBAP (
    VBELN VARCHAR(10) NOT NULL,
    POSNR VARCHAR(6) NOT NULL,
    MATNR VARCHAR(18) NOT NULL,
    KDMAT VARCHAR(18),
    VRKME CHAR(3),
    LFIMG DECIMAL(13, 3),
    WMENG DECIMAL(13, 3),
    VRKMG DECIMAL(13, 3),
    NETWR DECIMAL(15, 2),
    WAERK CHAR(5),
    KZWI1 DECIMAL(13, 2),
    KZWI2 DECIMAL(13, 2),
    KZWI3 DECIMAL(13, 2),
    KZWT4 DECIMAL(13, 2)
);
```

```
KZWI4 DECIMAL(13, 2),  
KZWI5 DECIMAL(13, 2),  
FKIMG DECIMAL(13, 3),  
NTGEW DECIMAL(13, 3),  
BRGEW DECIMAL(13, 3),  
GEWEI CHAR(3),  
VOLUM DECIMAL(13, 3),  
PRIMARY KEY (VBELN, POSNR),  
FOREIGN KEY (MATNR) REFERENCES MARA(MATNR)  
);
```

In this schema, a primary key is defined for each table, and a foreign key relationship is established between the MATNR column of the MARA table and the MATNR column of the VBAP table.

Figure 6.3 – Example of ChatGPT generating a SQL schema for the SAP tables

Note that ChatGPT was able to make relationships because it knows SAP taxonomy and column names interpretation, which is something anyone who doesn't know SAP taxonomy (including data engineers) struggles with. As confirmation, let's ask ChatGPT to rename all those columns with more human-understandable names:

V rename the above columns with more human-understandable names



Sure, here's an example of how the column names in the SAP tables MARA and VBAP could be renamed to be more human-readable:

MARA table:

- MATNR: Material Number
- ERSDA: Created On Date
- ERNAM: Created By User
- AENAM: Last Changed By User
- AEDAT: Last Changed On Date
- LAEDA: Deletion Date
- LVORM: Deletion Flag
- PRDHA: Material Group
- STPRS: Standard Price
- PEINH: Price Unit
- BWTAR: Valuation Class
- MEINS: Base Unit of Measure
- BSPGE: Minimum Remaining Shelf Life
- BKLAS: Material Class
- SERNP: Serial Number Profile
- SOBSL: Special Stock Indicator
- SPART: Division
- MDRSH: Industry Key
- MTPOS_MARA: Material Type

VBAP table:

- VBELN: Sales Document Number
- POSNR: Item Number
- MATNR: Material Number
- KDMAT: Customer Material Number
- VRKME: Sales Unit
- LFIMG: Delivery Quantity
- WMENG: Target Quantity in Sales Unit
- VRKMG: Sales Quantity
- NETWR: Net Value of the Item
- WAERK: Currency
- KZWI1: Condition Value
- KZWI2: Condition Value
- KZWI3: Condition Value
- KZWI4: Condition Value
- KZWI5: Condition Value
- FKIMG: Billing Quantity
- NTGEW: Net Weight
- BRGEW: Gross Weight
- GEWEI: Weight Unit
- VOLUM: Volume

Figure 6.4 – SAP tables columns renamed by ChatGPT

And, finally, let's recreate the schema with the new columns, making sure to avoid duplicates (the new name, **Condition Value**, is indeed repeated five times):

V recreate the sql schema for the two tables with renamed columns. If there are columns with same name, add a numeric indicator with an underscore at the end of the name

Here's the recreated SQL schema for the two tables with renamed columns and any duplicate column names appended with a numeric indicator and an underscore:

```
SCSS Copy code

CREATE TABLE MARA (
    MATERIAL_NUMBER VARCHAR(50) PRIMARY KEY,
    CREATED_ON DATE,
    CREATED_BY VARCHAR(50),
    LAST_CHANGED_BY VARCHAR(50),
    LAST_CHANGED_ON DATE,
    VALID_TO DATE,
    DELETION_INDICATOR CHAR(1),
    PRODUCT_HIERARCHY VARCHAR(50),
    STANDARD_PRICE DECIMAL(18,2),
    PRICE_UNIT DECIMAL(18,2),
    WEIGHT_UNIT CHAR(10),
    BASE_PRICE_LIST CHAR(10),
    STOCK_ITEM_STATUS CHAR(10),
    SERIAL_NUMBER_PROFILE CHAR(10),
    OBSOLETE_REASON CHAR(10),
    DIVISION CHAR(10),
    MEMBER_OF_GROUP CHAR(10),
    MATERIAL_TYPE_POSITION VARCHAR(50)
);

CREATE TABLE VBAP (
    SALES_DOCUMENT VARCHAR(50) NOT NULL,
    ITEM_NUMBER INT NOT NULL,
    MATERIAL_NUMBER VARCHAR(50),
    CUSTOMER_MATERIAL_NUMBER VARCHAR(50),
    SALES_UNIT CHAR(10),
    QUANTITY DECIMAL(18,2),
    DELIVERY_QUANTITY DECIMAL(18,2),
    NET_PRICE DECIMAL(18,2),
    CURRENCY CHAR(10),
    CONDITION_TYPE_1 DECIMAL(18,2),
    CONDITION_TYPE_2 DECIMAL(18,2),
    CONDITION_TYPE_3 DECIMAL(18,2),
    CONDITION_TYPE_4 DECIMAL(18,2),
    CONDITION_TYPE_5 DECIMAL(18,2),
    BILLED_QUANTITY DECIMAL(18,2),
    NET_WEIGHT DECIMAL(18,2),
    GROSS_WEIGHT DECIMAL(18,2),
    WEIGHT_UNIT CHAR(10),
    VOLUME DECIMAL(18,2)
);
ALTER TABLE VBAP ADD CONSTRAINT FK_MATERIAL_NUMBER_VBAP
    FOREIGN KEY (MATERIAL_NUMBER) REFERENCES MARA(MATERIAL_NUMBER);
```

Figure 6.5 – SQL schema with renamed columns

The preceding example is key to reducing the time to create schemas for further data analysis.

ChatGPT can also be a great assistant for code optimization. In fact, it might save us some running time or compute power to make optimized scripts starting from our input. This capability might be compared, in the domain of natural language, to the writing assistance feature we saw in [Chapter 5](#) in the *Improving writing skills and translation* section.

For example, imagine you want to create a list of odd numbers starting from another list. To achieve the result, you wrote the following Python script (for the purpose of this exercise, I will also track the execution time with the `timeit` and `datetime` libraries):

[Copy](#)

[Explain](#)

```
from timeit import default_timer as timer
from datetime import timedelta
start = timer()
elements = list(range(1_000_000))
data = []
for el in elements:
    if not el % 2:
        # if odd number
        data.append(el)
end = timer()
print(timedelta(seconds=end-start))
```

The execution time was **00.141185** seconds. What happens if we ask ChatGPT to optimize this script?

V

I wrote the following script to filter a list only for odd numbers.

```
elements = list(range(1_000_000))
data = []
for el in elements:
    if el % 2 != 0:
        data.append(el)
```

Can you create a script to achieve the same result in lower execution time?



Yes, using a list comprehension can be faster and more concise than a for loop. Here's an example script that achieves the same result as your original code but with a list comprehension:

python

Copy code

```
data = [el for el in range(1_000_000) if el % 2 != 0]
```

This code creates a list "data" containing only the odd numbers from the original list "range(1_000_000)", using a list comprehension. The "if" statement inside the list comprehension filters out even numbers, and the resulting list contains only odd numbers.

The list comprehension is generally faster than a "for" loop because it is optimized for Python's internal C implementation. This means that it can run much faster than a Python loop for large lists like this.

Note that the resulting list "data" will contain the same elements as your original "data" list created by the "for" loop, but the list comprehension should execute more quickly.

Figure 6.6 – ChatGPT generating optimized alternatives to a Python script

ChatGPT provided me with an example to achieve the same results with lower execution time. It also elaborated on the alternative with a clear explanation of why the comprehension method is faster than a **for** loop.

Let's see the performance of this new script compared to the previous one:

```

from timeit import default_timer as timer
from datetime import timedelta

start = timer()

#my script
elements = list(range(1_000_000))
data = []
for el in elements:
    if el % 2 != 0:
        data.append(el)

end = timer()
print(timedelta(seconds=end-start))

start = timer()

#ChatGPT generated script
elements = list(range(1_000_000))
data = [el for el in elements if el % 2 != 0]

end = timer()
print(timedelta(seconds=end-start))

```

0:00:00.141185
0:00:00.073695

Figure 6.7 – Comparison of user and ChatGPT scripts' execution times

As you can see, the second method (the one generated by ChatGPT) provides a reduction in execution time of about 47.8%.

On top of code generation and optimization, ChatGPT can also be leveraged for *error explanation and debugging*. Sometimes, errors are difficult to interpret; hence a natural language explanation can be useful to identify the problem and drive you toward the solution.

For example, while running a `.py` file from my command line, I get the following error:

[Copy](#)[Explain](#)

```
2023-03-25 11:27:10.270 Uncaught app exception
Traceback (most recent call last):
  File "C:\Users\vaalt\Anaconda3\lib\site-
  packages\streamlit\runtime\scriptrunner\script_runner.py", line 565, in _run_script
    exec(code, module.__dict__)
  File "C:\Users\vaalt\OneDrive\Desktop\medium articles\llm.py", line 129, in <module>
    user_input = get_text()
  File "C:\Users\vaalt\OneDrive\Desktop\medium articles\llm.py", line 50, in get_text
    input_text = st.text_input("You: ", st.session_state['input'], key='input',
placeholder = 'Your AI assistant here! Ask me anything...', label_visibility =
'hidden')
  File "C:\Users\vaalt\Anaconda3\lib\site-packages\streamlit\runtime\metrics_util.py",
line 311, in wrapped_func
    result = non_optional_func(*args, **kwargs)
  File "C:\Users\vaalt\Anaconda3\lib\site-
  packages\streamlit\elements\text_widgets.py", line 174, in text_input
    return self._text_input(
  File "C:\Users\vaalt\Anaconda3\lib\site-
  packages\streamlit\elements\text_widgets.py", line 266, in _text_input
    text_input_proto.value = widget_state.value
TypeError: [] has type list, but expected one of: bytes, Unicode
```

Let's see whether ChatGPT is able to let me understand the nature of the error. To do so, I simply provide ChatGPT with the text of the error and ask it to give me an explanation.

V

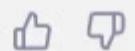
I got the following error while running a .py file:

```
2023-03-25 11:27:10.270 Uncaught app exception
Traceback (most recent call last):
  File "C:\Users\vaalt\Anaconda3\lib\site-
  packages\streamlit\runtime\scriptrunner\script_runner.py", line 565, in _run_script
    exec(code, module.__dict__)
  File "C:\Users\vaalt\OneDrive\Desktop\medium articles\llm.py", line 129, in <module>
    user_input = get_text()
  File "C:\Users\vaalt\OneDrive\Desktop\medium articles\llm.py", line 50, in get_text
    input_text = st.text_input("You: ", st.session_state['input'], key='input', placeholder = 'Your
AI assistant here! Ask me anything...', label_visibility = 'hidden')
  File "C:\Users\vaalt\Anaconda3\lib\site-packages\streamlit\runtime\metrics_util.py", line
311, in wrapped_func
    result = non_optional_func(*args, **kwargs)
  File "C:\Users\vaalt\Anaconda3\lib\site-packages\streamlit\elements\text_widgets.py", line
174, in text_input
    return self._text_input(
  File "C:\Users\vaalt\Anaconda3\lib\site-packages\streamlit\elements\text_widgets.py", line
266, in _text_input
    text_input_proto.value = widget_state.value
TypeError: [] has type list, but expected one of: bytes, unicode
```

Explain me the error and possible resolutions



The error message is indicating that there is a `TypeError` being raised while running your Python script. Specifically, the issue is with the `get_text()` function on line 50 of the `llm.py` file. The `st.text_input()` function is expecting a string value, but it is receiving a list object instead.



The list object is most likely coming from the `st.session_state['input']` variable, which is expected to hold a user's input from a previous session. It seems that this variable is being initialized as a list instead of a string, which is causing the `st.text_input()` function to fail.

To resolve the issue, you can modify the `st.session_state['input']` variable to hold a string value instead of a list. You can either initialize it as an empty string by using `st.session_state['input'] = ''`, or you can modify the code that sets its value to ensure that it is always a string.

Alternatively, you can modify the `st.text_input()` function to handle the list input by converting it to a string first, like this: `input_text = st.text_input("You: ",
str(st.session_state['input']), key='input', placeholder = 'Your AI assistant here!
Ask me anything...', label_visibility = 'hidden')`. This will ensure that the `text_input()` function receives a string value even if the `st.session_state['input']` variable holds a list object.

Figure 6.8 – ChatGPT explaining a Python error in natural language

Finally, let's imagine I wrote a function in Python that takes a string as input and returns the same string with an underscore after each letter.

In the preceding example, I was expecting to see the **g_p_t_** result; however, it only returned **t_** with this code:

```
def add_underscores(word):
    new_word = "_"
    for i in range(len(word)):
        new_word = word[i] + "_"
    return new_word

phrase = "gpt"
print(add_underscores(phrase))
```

t_

Figure 6.9 – Bugged Python function

Let's ask ChatGPT to debug this function for us:

V

I wrote this python function to add an underscore after each letter of a word.
The expected result for the input word should be `w_u_s_d_`.

```
def add_underscores(word):
    new_word = "_"
    for i in range(len(word)):
        new_word = word[i] + "_"
    return new_word

phrase = "gpt"
print(add_underscores(phrase))

can you debug this code?
```



Sure, here is a corrected version of the `'add_underscores'` function:



```
python

def add_underscores(word):
    new_word = ""
    for i in range(len(word)):
        new_word += word[i] + "_"
    return new_word
```

The bug in the original version of the function was that the `'new_word'` variable was initialized to an underscore character instead of an empty string. This caused the function to always start the output with an underscore character, which was not what was intended.

In the corrected version, the `'new_word'` variable is initialized to an empty string, and then each character of the input word is added to it with an underscore character. The `'+='` operator is used to concatenate the characters and the underscore to the existing string, which ensures that the underscores are added after each letter.

You can call the corrected function with a word, like this:

```
python

phrase = "gpt"
print(add_underscores(phrase))
```

This should output `'g_p_t_'`, which is the expected result for the input word.

Figure 6.10 – Example of ChatGPT debugging a Python function

Impressive, isn't it? Again, ChatGPT provided the correct version of the code, and it helped in the explanation of where the bugs were and why they led to an incorrect result. Let's see whether it works now:

```
def add_underscores(word):
    new_word = ""
    for i in range(len(word)):
        new_word += word[i] + "_"
    return new_word

phrase = "gpt"
print(add_underscores(phrase))
```

g_p_t_

Figure 6.11 – Python function after ChatGPT debugging

Well, it obviously does!

These and many other code-related functionalities could really boost your productivity, shortening the time to perform many tasks.

However, ChatGPT goes beyond pure debugging. Thanks to the incredible language understanding of the GPT model behind, this **artificial intelligence (AI)** tool is able to generate proper documentation alongside the code, as well as explain exactly what a string of code will do, which we will see in the next section.

Generating documentation and code explainability

Whenever working with new applications or projects, it is always good practice to correlate your code with documentation. It might be in the form of a docstring that you can embed in your functions or classes so that others can invoke them directly in the development environment.

For example, the following Python class has 10 different methods for basic mathematical operations:

[Copy](#)[Explain](#)

```
class Calculator:
    def add(self, x, y):
        return x + y
    def subtract(self, x, y):
        return x - y
    def multiply(self, x, y):
        return x * y
    def divide(self, x, y):
        try:
            return x / y
        except ZeroDivisionError:
            print("Error: division by zero")
            return None
    def power(self, x, y):
        return x ** y
    def square_root(self, x):
        try:
            return x ** 0.5
        except ValueError:
            print("Error: square root of a negative number")
            return None
    def logarithm(self, x, base):
        try:
            return math.log(x, base)
        except ValueError:
            print("Error: invalid logarithm arguments")
            return None
    def factorial(self, x):
        if x < 0:
            print("Error: factorial of a negative number")
            return None
        elif x == 0:
            return 1
        else:
            return x * self.factorial(x-1)
    def fibonacci(self, n):
        if n < 0:
            print("Error: fibonacci sequence index cannot be negative")
            return None
        elif n == 0:
            return 0
        elif n == 1:
            return 1
        else:
            return self.fibonacci(n-1) + self.fibonacci(n-2)
```

You can initialize the class and test it as follows (as an example, I will use the addition method):

```
calc = Calculator()  
print(calc.add(3, 4))
```

7



Figure 6.12 – Initializing a Python class and using one of its methods

Now, let's say I want to be able to retrieve the docstring documentation using the **Calculator?** convention. By doing so with Python packages, functions, and methods, we have full documentation of the capabilities of that specific object, as follows (an example with the **pandas** Python library):

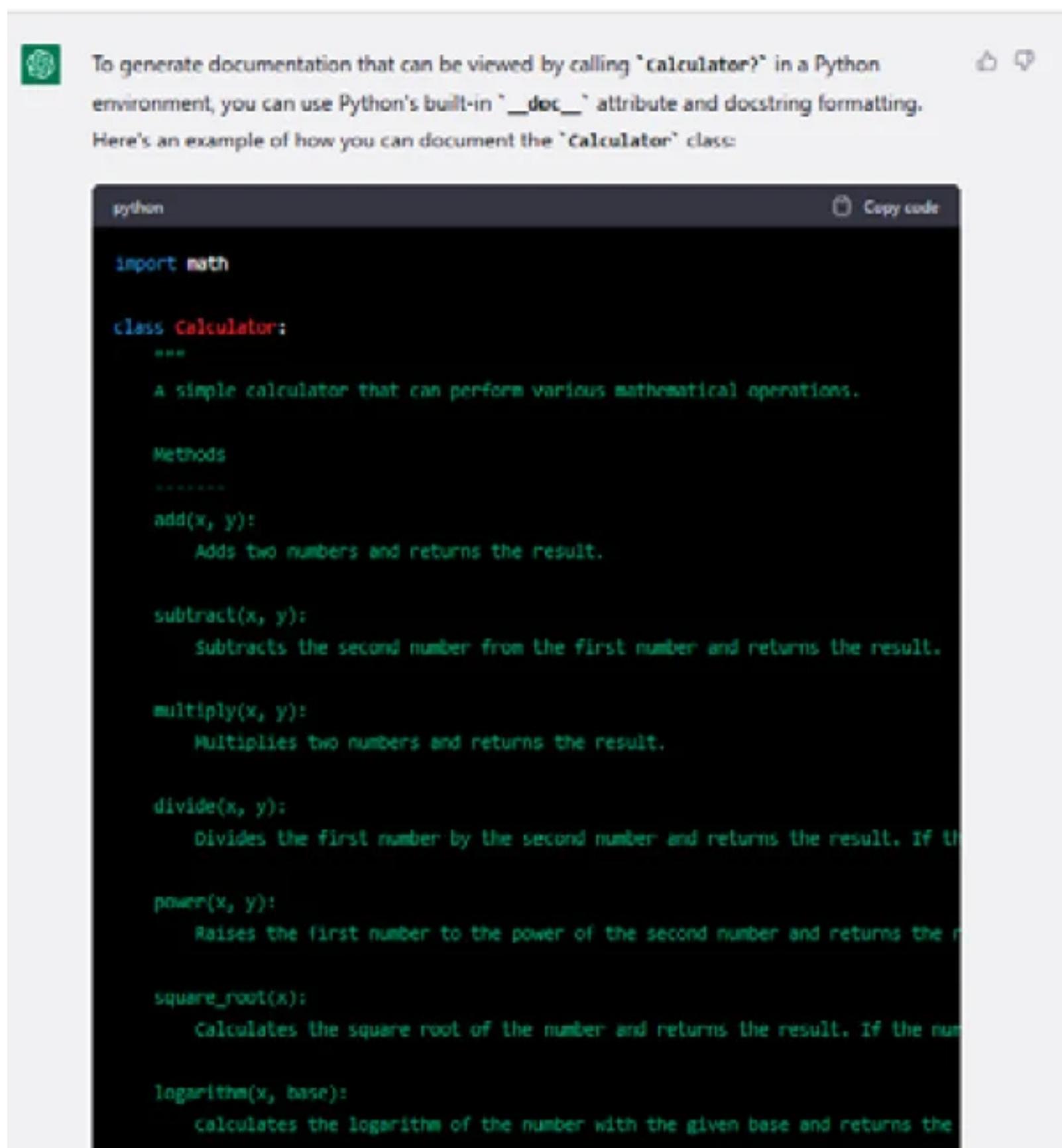
```
import pandas as pd  
pd?  
  
Type:      module  
String form: <module 'pandas' from 'C:\\\\Users\\\\vaalt\\\\Anaconda3\\\\lib\\\\site-packages\\\\pandas\\\\__init__.py'>  
File:      c:\\users\\vaalt\\anaconda3\\lib\\site-packages\\pandas\\__init__.py  
Docstring:  
pandas - a powerful data analysis and manipulation library for Python  
-----  
  
**pandas** is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python. Additionally, it has the broader goal of becoming **the most powerful and flexible open source data analysis / manipulation tool available in any language**. It is already well on its way toward this goal.  
  
Main Features  
  
Here are just a few of the things that pandas does well:  
  
- Easy handling of missing data in floating point as well as non-floating point data.  
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects  
- Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let 'Series', 'DataFrame', etc. automatically align the data for you in computations.  
- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data.  
- Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects.  
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets.  
- Intuitive merging and joining data sets.  
- Flexible reshaping and pivoting of data sets.  
- Hierarchical labeling of axes (possible to have multiple labels per tick).  
- Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving/loading data from the ultrafast HDF5 format.  
- Time series-specific functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging.
```

Figure 6.13 – Example of the pandas library documentation

So let's now ask ChatGPT to produce the same result for our **Calculator** class.

Henceforth, after giving the **Calculator** class as context to ChatGPT, I asked it to generate the relative documentation as follows:

Generate documentation about the **Calculator** class above so that developers can have a look at it by calling "**Calculator?**" in their python environment



To generate documentation that can be viewed by calling "calculator?" in a Python environment, you can use Python's built-in "__doc__" attribute and docstring formatting. Here's an example of how you can document the "calculator" class:

```
python
import math

class Calculator:
    """
    A simple calculator that can perform various mathematical operations.

    Methods
    -------

    add(x, y):
        Adds two numbers and returns the result.

    subtract(x, y):
        Subtracts the second number from the first number and returns the result.

    multiply(x, y):
        Multiplies two numbers and returns the result.

    divide(x, y):
        Divides the first number by the second number and returns the result. If the

    power(x, y):
        Raises the first number to the power of the second number and returns the result.

    square_root(x):
        Calculates the square root of the number and returns the result. If the number

    logarithm(x, base):
        calculates the logarithm of the number with the given base and returns the result.
    """

    def add(self, x, y):
        return x + y

    def subtract(self, x, y):
        return x - y

    def multiply(self, x, y):
        return x * y

    def divide(self, x, y):
        if y == 0:
            raise ValueError("Division by zero")
        return x / y

    def power(self, x, y):
        return x ** y

    def square_root(self, x):
        if x < 0:
            raise ValueError("Square root of negative number")
        return x ** 0.5

    def logarithm(self, x, base):
        return math.log(x, base)
```

Figure 6.14 – ChatGPT updating the code with documentation

You can find the whole ChatGPT response in the GitHub repository of this book at <https://github.com/PacktPublishing/Modern-Generative-AI-with-ChatGPT-and-OpenAI-Models/blob/main/Chapter%206%20-%20ChatGPT%20for%20Developers/prompts>.

As a result, if we update our class as shown in the preceding code and call **Calculator?**, we will get the following output:

Calculator?

```
Init signature: Calculator()
Docstring:
A simple calculator that can perform various mathematical operations.

Methods
-----
add(x, y):
    Adds two numbers and returns the result.

subtract(x, y):
    Subtracts the second number from the first number and returns the result.

multiply(x, y):
    Multiplies two numbers and returns the result.

divide(x, y):
    Divides the first number by the second number and returns the result. If the second number is zero, returns None.

power(x, y):
    Raises the first number to the power of the second number and returns the result.

square_root(x):
    Calculates the square root of the number and returns the result. If the number is negative, returns None.

logarithm(x, base):
    Calculates the logarithm of the number with the given base and returns the result. If the number is negative or the base is invalid, returns None.

factorial(x):
    Calculates the factorial of the number and returns the result. If the number is negative, returns None.

fibonacci(n):
    Generates the n-th number in the Fibonacci sequence and returns the result. If n is negative, returns None.

Type:      type
Subclasses:
```

Figure 6.15 – The new Calculator class documentation

Finally, ChatGPT can also be leveraged to explain what a script, function, class, or other similar things do in natural language. We have already seen many examples of ChatGPT enriching its code-related response with clear explanations. However, we can boost this capability by asking specific questions in terms of code understanding.

For example, let's ask ChatGPT to explain to us what the following Python script does:

```
V from azureml.opendatasets import NycTlcYellow  
from datetime import datetime  
from dateutil import parser
```

```
start_date = parser.parse('2018-05-01')  
end_date = parser.parse('2018-05-07')  
nyc_tlc = NycTlcYellow(start_date=start_date, end_date=end_date)  
nyc_tlc_df = nyc_tlc.to_pandas_dataframe()  
nyc_tlc_df.info() // Explain this code
```



This code imports the `NycTlcYellow` class from the `azureml.opendatasets` module, which is used to retrieve data from the New York City Taxi and Limousine Commission (TLC) yellow taxi trip data.



The code then sets a start and end date for the data that will be retrieved by creating `datetime` objects from strings using the `parser.parse()` method from the `dateutil` module. In this case, the start date is set to May 1, 2018, and the end date is set to May 7, 2018.

An instance of the `NycTlcYellow` class is then created with the specified start and end dates, and the data is retrieved using the `to_pandas_dataframe()` method. This method converts the retrieved data to a pandas DataFrame, which is assigned to the `nyc_tlc_df` variable.

Finally, the `info()` method is called on the DataFrame to print information about its columns, including data types and the number of non-null values. This is a useful way to inspect the structure of the DataFrame and check for missing values.

Figure 6.16 – Example of ChatGPT explaining a Python script

Code explainability can also be part of the preceding mentioned documentation, or it can be used among developers who might want to better understand complex code from other teams or (as sometimes happens to me) remember what they wrote some time ago.

Thanks to ChatGPT and the capabilities mentioned in this section, developers can easily keep track of the project life cycle in natural language so that it is easier for both new team members and non-technical users to understand the work done so far.

We will see in the next section how code explainability is a pivotal step for ML model interpretability in data science projects.

Understanding ML model interpretability

Model interpretability refers to the degree of ease with which a human can comprehend the logic behind the ML model's predictions. Essentially, it is the capability to comprehend how a model arrives at its decisions and which variables are contributing to its forecasts.

Let's see an example of model interpretability using a deep learning **convolutional neural network (CNN)** for image classification. I built my model in Python using Keras. For this purpose, I will download the CIFAR-10 dataset directly from `keras.datasets`: it consists of 60,000 32x32 color images (so 3-channels images) in 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck), with 6,000 images per class. Here, I will share just the body of the model; you can find all the related code in the book's GitHub repository for data preparation and pre-processing at <https://github.com/PacktPublishing/Modern-Generative-AI-with-ChatGPT-and-OpenAI-Models/tree/main/Chapter%206%20-%20ChatGPT%20for%20Developers/code>:

```
model=tf.keras.Sequential()
model.add(tf.keras.layers.Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(32,32,1)))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(1024,activation='relu'))
model.add(tf.keras.layers.Dense(10,activation='softmax'))
```

Copy

Explain

The preceding code is made of several layers that perform different actions. I might be interested in having an explanation of the structure of the model as well as the purpose of each layer. Let's ask ChatGPT for some help with that:

```
V model=tf.keras.Sequential()  
model.add(tf.keras.layers.Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=  
(32,32,1)))  
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))  
model.add(tf.keras.layers.Flatten())  
model.add(tf.keras.layers.Dense(1024,activation='relu'))  
model.add(tf.keras.layers.Dense(10,activation='softmax'))
```

can you explain this code to me?

Certainly! This code defines a neural network model using the Keras API within TensorFlow, which is a popular open-source machine learning framework. The model is used for image classification tasks, specifically on the CIFAR-10 dataset, which contains 10 classes of 32x32 pixel images. Here's what each line of the code does:

1. `model=tf.keras.Sequential()` : Creates a sequential model, which is a linear stack of layers.
2. `model.add(tf.keras.layers.Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(32,32,1)))` : Adds a convolutional layer to the model, with 32 filters of size 3x3, using the ReLU activation function. The input shape is set to 32x32 pixels with a single color channel.
3. `model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))` : Adds a max pooling layer to the model, with a pooling window size of 2x2. This helps to downsample the input and reduce the number of parameters in the model.
4. `model.add(tf.keras.layers.Flatten())` : Adds a flattening layer to the model, which takes the output of the previous layer and flattens it into a 1-dimensional vector. This is necessary to connect the output of the convolutional layers to the fully connected layers.
5. `model.add(tf.keras.layers.Dense(1024,activation='relu'))` : Adds a fully connected layer with 1024 neurons and a ReLU activation function.
6. `model.add(tf.keras.layers.Dense(10,activation='softmax'))` : Adds a final fully connected layer with 10 neurons (one for each class) and a softmax activation function, which produces a probability distribution over the classes.

Overall, this model has a convolutional layer for feature extraction, followed by a max pooling layer for downsampling, a flattening layer to reshape the output, and two fully connected layers for classification.

Figure 6.17 – Model interpretability with ChatGPT

As you can see in the preceding figure, ChatGPT was able to give us a clear explanation of the structure and layers of our CNN. It also adds some comments and tips, such as the fact that using the max pooling layer helps reduce the dimensionality of the input.

I can also be supported by ChatGPT in interpreting model results in the validation phase. So, after splitting data into training and test sets and training the model on the training set, I want to see its performance on the test set:

```
model.fit(np.resize(x_train, (50000,32,32,1))/255.0,  
         tf.keras.utils.to_categorical(y_train),  
         batch_size=batch_size,  
         shuffle=True,  
         epochs=epochs,  
         validation_data=(np.resize(x_test, (10000,32,32,1))/255.0,  
                         tf.keras.utils.to_categorical(y_test))  
     )
```

↑ ↓ ± ⌂

```
Epoch 1/3  
1563/1563 [=====] - 208s 133ms/step - loss: 2.3021 - accuracy: 0.1038 - val_loss: 2.3029 - val_accuracy: 0.1039  
Epoch 2/3  
1563/1563 [=====] - 209s 133ms/step - loss: 2.2964 - accuracy: 0.1104 - val_loss: 2.3070 - val_accuracy: 0.1001  
Epoch 3/3  
1563/1563 [=====] - 909s 582ms/step - loss: 2.2743 - accuracy: 0.1312 - val_loss: 2.3173 - val_accuracy: 0.1006
```

Figure 6.18 – Evaluation metrics

Let's also ask ChatGPT to elaborate on our validation metrics:

V

Epoch 1/3

1563/1563 [=====] - 208s 133ms/step - loss: 2.3021

- accuracy: 0.1038 - val_loss: 2.3029 - val_accuracy: 0.1039

Epoch 2/3

1563/1563 [=====] - 209s 133ms/step - loss: 2.2964

- accuracy: 0.1104 - val_loss: 2.3070 - val_accuracy: 0.1001

Epoch 3/3

1563/1563 [=====] - 909s 582ms/step - loss: 2.2743

- accuracy: 0.1312 - val_loss: 2.3173 - val_accuracy: 0.1006

I received as output of the model testing the above results. Can you elaborate those metrics and explaining what do they mean?



These are the results of training a neural network model for image classification using the CIFAR-10 dataset. The training is done over 3 epochs, and for each epoch, the model's performance on both the training set and validation set is reported.



- Epoch: The number of passes over the entire training dataset during training.
- Loss: A measure of how well the model is able to predict the correct class. The loss function is used to compute the difference between the predicted class and the actual class. The goal of training is to minimize this loss. In this case, the loss is measured using the categorical cross-entropy loss function.
- Accuracy: The percentage of images that are correctly classified by the model. The accuracy metric is used to evaluate the performance of the model. In this case, it is reported as a decimal value between 0 and 1.
- Val_loss: The loss on the validation set. The validation set is a subset of the dataset that is used to evaluate the model's performance during training. It is important to use a validation set to detect overfitting, which is when the model becomes too specialized to the training set and performs poorly on new data.
- Val_accuracy: The accuracy on the validation set. The validation accuracy is used to measure the generalization performance of the model. A high validation accuracy indicates that the model is able to generalize well to new data.

In summary, the training results show that the model is improving with each epoch, as the loss is decreasing and the accuracy is increasing. However, the validation results show that the model is not generalizing well, as the validation accuracy is much lower than the training accuracy. This could be an indication of overfitting, and further investigation is needed to improve the model's performance on new data.

Figure 6.19 – Example of ChatGPT explaining evaluation metrics

Once again, the result was really impressive, and it provided clear guidance on how to set up ML experiments in terms of training and test sets. It explains how important it is for the model to be sufficiently generalized so that it does not overfit and is able to predict accurate results on data that it has never seen before.

There are many reasons why model interpretability is important. A pivotal element is that it reduces the gap between business users and the code behind models. This is key to enabling business users to understand how a model behaves, as well as translate it into code business ideas.

Furthermore, model interpretability enables one of the key principles of responsible and ethical AI, which is the transparency of how a model behind AI systems thinks and behaves. Unlocking model interpretability means detecting potential biases or harmful behaviors a model could have while in production and consequently preventing them from happening.

Overall, ChatGPT can provide valuable support in the context of model interpretability, generating insights at the row level, as we saw in the previous example.

The next and last ChatGPT capability we will explore will be yet another boost for developers' productivity, especially when various programming languages are being used within the same project.

Translation among different programming languages

In [Chapter 5](#), we saw how ChatGPT has great capabilities for translating between different languages. What is really incredible is that natural language is not its only object of translation. In fact, ChatGPT is capable of translating between different programming languages while keeping the same output as well as the same style (namely, it preserves docstring documentation if present).

There are so many scenarios when this could be a game changer.

For example, you might have to learn a new programming language or statistical tool you've never seen before because you need to quickly deliver a project on it. With the help of ChatGPT, you can start programming in your language of preference and then ask it to translate to the desired language, which you will be learning alongside the translation process.

Imagine that the project needs to be delivered in MATLAB (a proprietary numerical computing and programming software developed by MathWorks), yet you've always programmed in Python. The project consists of classifying images from the **Modified National Institute of Standards and Technology (MNIST)** dataset (the original dataset description and related paper can be found [here](#) at <http://yann.lecun.com/exdb/mnist/>). The dataset contains numerous handwritten digits and is frequently utilized to teach various image processing systems.

To start, I wrote the following Python code to initialize a deep-learning model for classification:

```
Copy Explain

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
# Preprocess the data
x_train = x_train.reshape(-1, 28*28) / 255.0
x_test = x_test.reshape(-1, 28*28) / 255.0
y_train = keras.utils.to_categorical(y_train)
y_test = keras.utils.to_categorical(y_test)
# Define the model architecture
model = keras.Sequential([
    layers.Dense(256, activation='relu', input_shape=(28*28,)),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Train the model
history = model.fit(x_train, y_train, validation_split=0.2, epochs=10, batch_size=128)
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print('Test accuracy:', test_acc)
```

Let's now see what happens if we give the preceding code as context to ChatGPT and ask it to translate it into MATLAB:

V Translate the preceding code in Matlab



Sure, here's the translation of the preceding Python code into MATLAB:

The screenshot shows a MATLAB code editor window with the title 'matlab'. The code is a direct translation of the provided Python code into MATLAB syntax. It includes loading the MNIST dataset, preprocessing the data (resizing and normalizing), defining a neural network architecture with six layers, specifying training options (using Adam optimizer, setting epochs to 10, mini-batch size to 128, validation data from the first 10,000 samples, and plotting training progress), training the network, and finally evaluating it on the test set to get a 92.93% accuracy.

```
% Load the MNIST dataset
[Xtrain, Ytrain] = mnist_train;
[Xtest, Ytest] = mnist_test;

% Preprocess the data
XTrain = double(reshape(XTrain, [size(XTrain, 1), 28*28])) / 255;
xtest = double(reshape(xtest, [size(xtest, 1), 28*28])) / 255;
Ytrain = categorical(Ytrain);
YTest = categorical(Ytest);

% Define the model architecture
model = [
    fullyConnectedLayer(256)
    reluLayer
    fullyConnectedLayer(128)
    reluLayer
    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer
];

% Specify the training options
options = trainingOptions('adam', ...
    'maxEpochs', 10, ...
    'miniBatchSize', 128, ...
    'validationData', {XTrain(1:10000,:), YTrain(1:10000)}, ...
    'plots', 'training-progress');

% Train the model
net = trainNetwork(Xtrain, Ytrain, model, options);

% Evaluate the model
YPred = classify(net, Xtest);
accuracy = sum(YPred == Ytest) / numel(Ytest);
fprintf('Test accuracy: %.2f%%\n', accuracy * 100);
```

Figure 20 – ChatGPT translates Python code into MATLAB

Let's also see whether it is capable of translating it into other languages such as JavaScript:

V

translate the preceding deep learning model written in Python to JavaScript



To translate the preceding deep learning model from Python to JavaScript, we can use TensorFlow.js, a library for machine learning in JavaScript. Here's the translation of the Python code to JavaScript:



```
javascript Copy code

// Load the MNIST dataset
const data = tf.data.generator(function() {
  return {value: tf.nextFrame(tf.tensor([mnist.nextTrainBatch(128)]))};
});

// Preprocess the data
const trainData = data.take(10000).map(x => {
  return {x: x.value.reshape([128, 28 * 28]).div(255), y: tf.oneHot(x.value)};
});
const testData = data.skip(10000).map(x => {
  return {x: x.value.reshape([128, 28 * 28]).div(255), y: tf.oneHot(x.value)};
};

// Define the model architecture
const model = tf.sequential({
  layers: [
    tf.layers.dense({units: 256, inputShape: [28 * 28], activation: 'relu'}),
    tf.layers.dense({units: 128, activation: 'relu'}),
    tf.layers.dense({units: 10, activation: 'softmax'})
  ]
});

// Specify the training options
const optimizer = tf.train.adam();
const loss = 'categoricalCrossentropy';
const metrics = ['accuracy'];
const batchSize = 128;
const epochs = 10;
const validationSplit = 0.1;
const shuffle = true;
const verbose = 1;
const fitArgs = {optimizer, loss, metrics, batchSize, epochs, validationSplit};

// Train the model
model.fitDataset(trainData, fitArgs);

// Evaluate the model
const evalArgs = {batchSize: 128};
const evalResult = model.evaluateDataset(testData, evalArgs);
console.log(`Test loss: ${evalResult[0].dataSync()[0].toFixed(2)}, accuracy: ${evalResult[1].toFixed(2)})`);
```

This code has the same functionality as the Python code, with some differences in syntax and usage. We first load the MNIST dataset using the TensorFlow.js data API, preprocess it, and define the model architecture using the `tf.sequential` API. We then specify the training options, including the optimizer, loss function, metrics, batch size, and number of epochs, and train the model using the `model.fitDataset` method. Finally, we evaluate the model on the test set using the `model.evaluateDataset` method and print the test loss and accuracy.

Figure 6.21 – ChatGPT translates Python code into JavaScript

Code translation could also reduce the skill gap between new technologies and current programming capabilities.

Another key implication of code translation is **application modernization**. Indeed, imagine you want to refresh your application stack, namely migrating to the cloud. You could decide to initiate with a simple lift and shift going toward **Infrastructure-as-a-Service (IaaS)** instances (such as Windows or Linux **virtual machines (VMs)**). However, in a second phase, you might want to refactor, rearchitect, or even rebuild your applications.

The following diagram depicts the various options for application modernization:

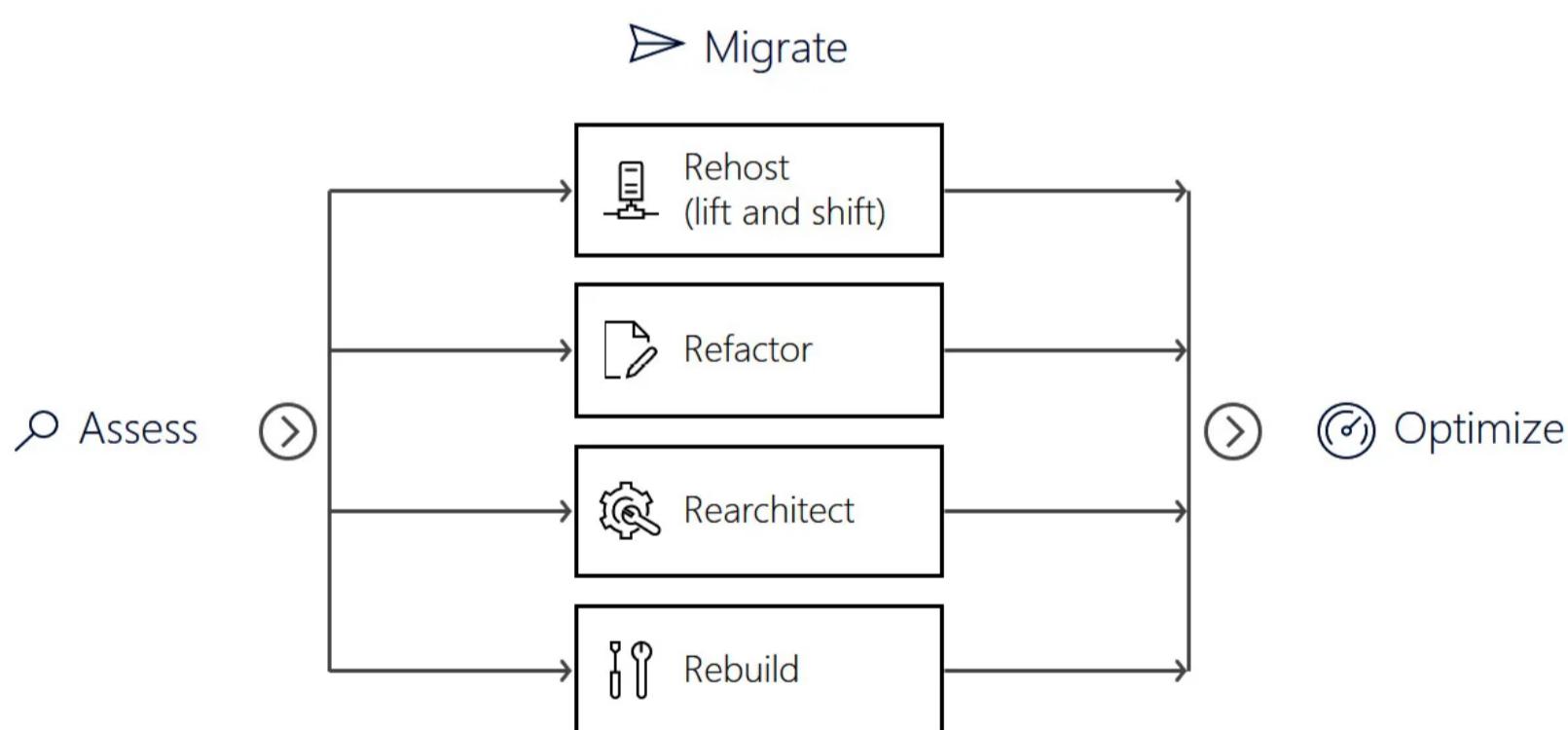


Figure 6.22 – Four ways you can migrate your applications to the public cloud

ChatGPT and OpenAI Codex models can help you with the migration. Consider mainframes, for example.

Mainframes are computers that are predominantly employed by large organizations to carry out essential tasks such as bulk data processing for activities such as censuses, consumer and industry statistics, enterprise resource planning, and large-scale transaction processing. The application programming language of the

mainframe environment is **Common Business Oriented Language (COBOL)**. Despite being invented in 1959, COBOL is still in use today and is one of the oldest programming languages in existence.

As technology continues to improve, applications residing in the realm of mainframes have been subject to a continuous process of migration and modernization aimed at enhancing existing legacy mainframe infrastructure in areas such as interface, code, cost, performance, and maintainability.

Of course, this implies translating COBOL to more modern programming languages such as C# or Java. The problem is that COBOL is unknown to most of the new-generation programmers; hence there is a huge skills gap in this context.

Let's consider a COBOL script that reads a file of employee records and writes a report of employees who have been with the company for more than 10 years:

IDENTIFICATION DIVISION.

PROGRAM-ID. EMPLOYEEREPORT.

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT EMPLOYEE-FILE ASSIGN TO 'EMPLOYEE.DAT'

ORGANIZATION IS LINE SEQUENTIAL.

SELECT REPORT-FILE ASSIGN TO 'EMPLOYEEREPORT.TXT'

ORGANIZATION IS LINE SEQUENTIAL.

DATA DIVISION.

FILE SECTION.

FD EMPLOYEE-FILE.

01 EMPLOYEE-RECORD.

 05 EMPLOYEE-ID PIC X(10).

 05 HIRE-DATE PIC 9(6).

 05 EMPLOYEE-NAME PIC X(30).

FD REPORT-FILE.

01 REPORT-LINE PIC X(80).

WORKING-STORAGE SECTION.

01 WS-CURRENT-DATE PIC 9(6).

01 WS-EMPLOYEE-YEARS PIC 9(2).

01 WS-REPORT-HEADER.

 05 FILLER PIC X(30) VALUE 'EMPLOYEES WITH 10+ YEARS OF SERVICE'.

 05 FILLER PIC X(10) VALUE SPACES.

 05 WS-REPORT-DATE PIC X(6).

PROCEDURE DIVISION.

MAIN-PARAGRAPH.

 OPEN INPUT EMPLOYEE-FILE

 OPEN OUTPUT REPORT-FILE

 MOVE FUNCTION CURRENT-DATE TO WS-CURRENT-DATE

 MOVE WS-CURRENT-DATE(1:6) TO WS-REPORT-DATE

 WRITE REPORT-LINE FROM WS-REPORT-HEADER

 WRITE REPORT-LINE FROM SPACES

 PERFORM READ-EMPLOYEE UNTIL EMPLOYEE-FILE-STATUS = '10'

 CLOSE EMPLOYEE-FILE

 CLOSE REPORT-FILE

 STOP RUN.

READ-EMPLOYEE.

 READ EMPLOYEE-FILE

 AT END MOVE '10' TO EMPLOYEE-FILE-STATUS

 END-READ

 COMPUTE WS-EMPLOYEE-YEARS = FUNCTION INTEGER-OF-DATE(WS-CURRENT-DATE) - HIRE-
 DATE(1:6)

 IF WS-EMPLOYEE-YEARS >= 10

 WRITE REPORT-LINE FROM EMPLOYEE-RECORD

 END-IF.

I then passed the previous COBOL script to ChatGPT so that it can use it as context to formulate its response. Let's now ask ChatGPT to translate that script into JavaScript:

The preceding code is written in COBOL, a programming language commonly used for business and administrative applications in the 1960s and 1970s. It reads data from an input file called "EMPLOYEE.DAT", which contains information about employees including their ID, hire date, and name. It then writes a report to an output file called "EMPLOYEEREPOR.TXT", listing the names of employees who have been with the company for 10 or more years.

Here is a translation of the COBOL code into JavaScript:

```
javascript
const fs = require('fs');

const employeeData = fs.readFileSync('EMPLOYEE.DAT', 'utf-8');
const employeeRecords = employeeData.split('\n').filter(record => record.length > 6);

const reportHeader = 'EMPLOYEES WITH 10+ YEARS OF SERVICE\n\n';
const reportLines = employeeRecords.filter(record => {
  const hireDate = Number.parseInt(record.substr(11, 6), 10);
  const currentDate = new Date();
  const currentYear = currentDate.getFullYear();
  const currentMonth = currentDate.getMonth() + 1;
  const currentDay = currentDate.getDate();
  const currentDateString = `${currentYear}${currentMonth < 10 ? '0' : ''}${currentDay}`;
  const yearsOfService = Math.floor((Number.parseInt(currentDateString, 10) - hireDate) / 10);
  return yearsOfService >= 10;
}).map(record => record.substr(0, 10));

const report = reportHeader + reportLines.join('\n');

fs.writeFileSync('EMPLOYEEREPOR.TXT', report);
```

This code reads the contents of the "EMPLOYEE.DAT" file, splits it into individual employee records, and filters out any empty records. It then calculates the years of service for each employee by subtracting their hire date from the current date, and filters out any employees who have been with the company for less than 10 years. Finally, it creates a report string consisting of the report header and the names of the qualifying employees, separated by newlines, and writes it to a file called "EMPLOYEEREPOR.TXT".

Figure 6.23 – Example of ChatGPT translating COBOL to JavaScript

Tools such as ChatGPT can help in reducing the skill gap in this and similar scenarios by introducing a layer that knows both the past and the future of programming.

In conclusion, ChatGPT can be an effective tool for application modernization, providing code upgrading in addition to valuable insights and recommendations for enhancing legacy systems. With its advanced language processing capabilities and

extensive knowledge base, ChatGPT can help organizations streamline their modernization efforts, making the process faster, more efficient, and more effective.

Note

OpenAI codex models are the engines behind GitHub Copilot. This new capability encompasses many of the use cases we will see in this chapter, such as code generation, autofill, and code optimization.

Summary

ChatGPT can be a valuable resource for developers looking to enhance their skills and streamline their workflows. We started by seeing how ChatGPT can generate, optimize, and debug your code, but we also covered further capabilities such as generating documentation alongside your code, explaining your ML models, and translating between different programming languages for application modernization.

Whether you're a seasoned developer or just starting out, ChatGPT offers a powerful tool for learning and growth, reducing the gap between code and natural language.

In the next chapter, we will dive deeper into another domain of application where ChatGPT could be a game changer: marketing.

[Previous Chapter](#)[Next Chapter](#)