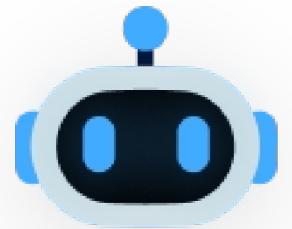


- Niccolò Cetoloni
- Mirko Bruttini
- Lorenzo Lombrichi

A.S 2023/2024
PROGETTO INFORMATICA

Progetto ChatBot - AI



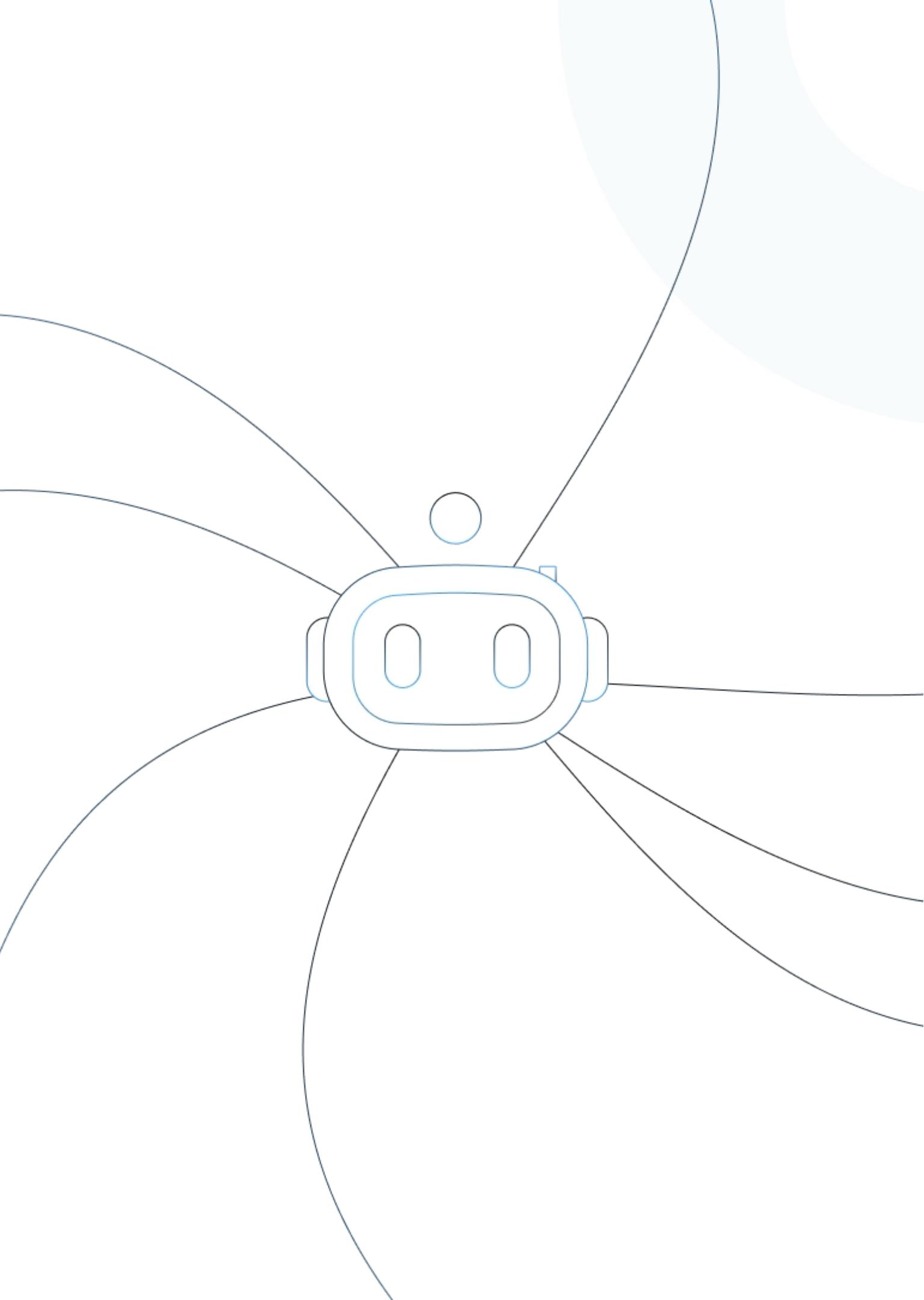


Chi è Michele?

Michele è un chatbot sviluppato in Python, con il quale puoi comunicare riguardo al tuo piano telefonico.

Si tratta di un modello classificatore, cioè assegna una categoria o un'etichetta in base alla domanda classificata in input, e sceglie la risposta più appropriata da un elenco.

 [Documentazione Completa](#)



Indice

01

NLP Concepts

I concetti del Natural Language Processing

02

Struttura del Progetto

Come comunicano i vari file

03

Strumenti Utilizzati

Quali sono i linguaggi e gli strumenti adottati

04

Analisi del Codice

- NLTK_Utils.py
- Intents.json
- Train.py
- Model.py
- Chat.py

05

GitHub & GIT

Strumenti collaborativi

Strumenti e Linguaggi adottati

Linguaggi

- Python
- HTML
- CSS



Strumenti di supporto

- GitHub
- Youtube
- ChatGPT/GitHub Copilot



Librerie

- PyTorch
- NLTK
- Numpy



Framework

- Streamlit

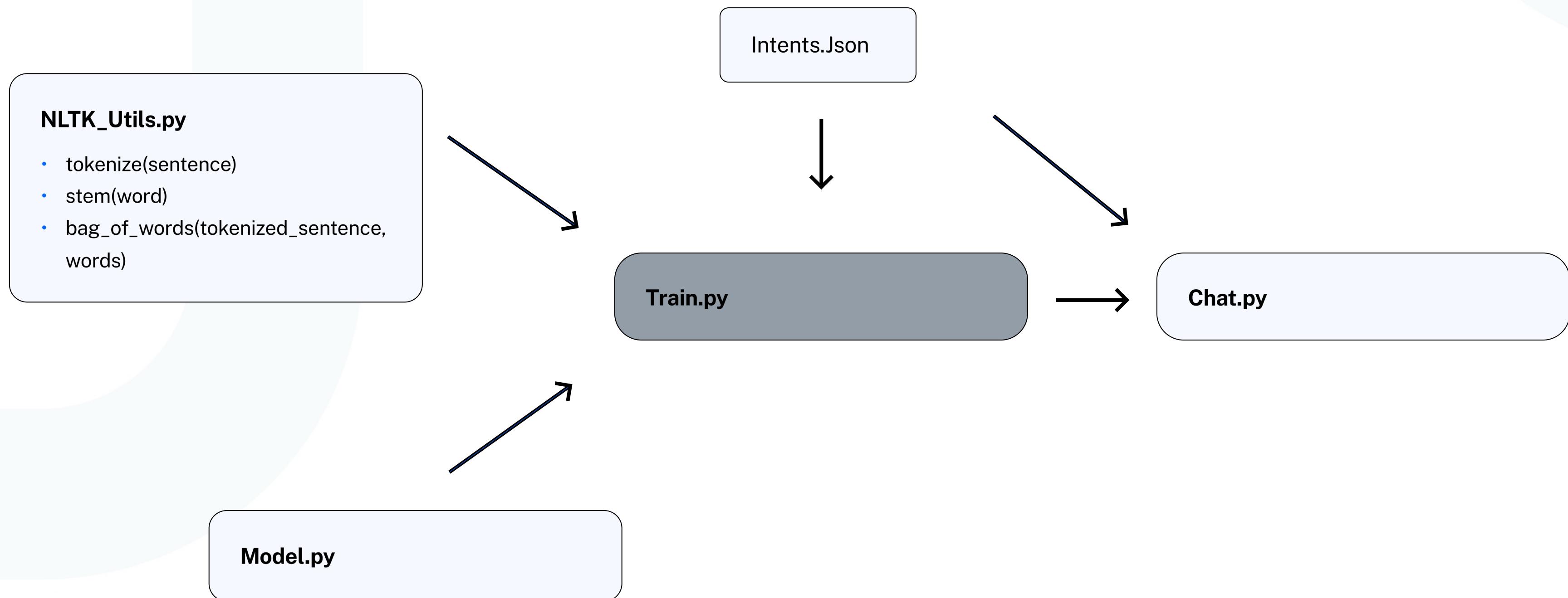


Risorse del Progetto 

L'intero progetto è disponibile, scaricabile e testabile. Visita

[GitHub](#)

Struttura del Progetto



Concetti NLP



NLP Concepts

1. Cos'è il Natural-Language-Processing?

E' lo studio tramite metodi computazionali del linguaggio naturale (cioè quello parlato dagli umani).



2. Perchè è necessario trattarlo?

Contiene i concetti logici base da cui il progetto prende forma.

3. Task riguardanti l'NLP

- Analisi del sentimento
- Riconoscimento di persone, oggetti e luoghi
- Role labeling (Sapere chi ha fatto cosa in un certo testo)
- Analisi sintattica
- Analisi grammaticale

Tokenizzazione, Stemming e BoW

NLP concepts

Tokenization

E' il processo di suddivisione di un testo in unità più piccole, aventi un token.

ES:

Ciao io sono Michele!

—> ["Ciao","io","sono","Michele","!"]

Stemming (o Lemmatization)

Fa parte del processo di analisi morfologica.

E' una tecnica utilizzata per ridurre le parole alle loro radici o forme base, chiamate *lemma*.

["running", "runs", "runner"] —> run

Bag of Words

Rappresentazione vettoriale della frequenza delle parole nella frase rispetto alla lista di tutte le parole.

Diz. [Ciao,io,oggi,aiuto,sono,Michele]

Frase: Ciao io sono Michele

BoW = [1, 1, 0, 0, 1, 1]

Analisi del Codice

- NLTK_Utils.py
- Intents.json
- Train.py
- Model.py
- Chat.py



NLTK_Utils

- Importazione delle librerie esterne NLTK e NUMPY

Nota: La libreria Numpy viene importata "as np".

Quando si esegue "stemmer = PorterStemmer()", si crea un oggetto stemmer che può essere utilizzato per stemmare le parole in un testo.

Definizione dei metodi:

Metodo Tokenize

Data la frase come parametro, verrà restituita la frase tokenizzata.

Metodo Stem

Data una parola come parametro, verrà effettuata la pratica di stemming tramite il metodo stem() importato dalla libreria nltk.stem.porter.

Metodo Bag_of_Words

Prende una frase tokenizzata e un insieme di parole, e restituisce una rappresentazione binaria della frase in forma di bag-of-words rispetto a quel set di parole.



```
1 import nltk
2 import numpy as np
3
4 from nltk.stem.porter import PorterStemmer
5 stemmer = PorterStemmer()
```



```
7 def tokenize(sentence):
8     return nltk.word_tokenize(sentence)
9
10 def stem(word):
11     return stemmer.stem(word.lower())
12
13 def bag_of_words(tokenized_sentence, words):
14
15     sentence_words = [stem(w) for w in tokenized_sentence]
16     bag = np.zeros(len(words), dtype=np.float32)
17     for idx, w in enumerate(words):
18         if w in sentence_words:
19             bag[idx] = 1.0
20
21     return bag
```

```
1  {
2    "intents": [
3      {
4        "tag": "greeting",
5        "patterns": [
6          "Ciao",
7          "Ciaone",
8          "Ciao Michele",
9          "Ciao come stai?",
10         "Ciao, come va?",
11         "Ciao, come stai?",
12         "Ciao, come'è?",
13         "Hey",
14         "Ciao Michele, come stai?",
15         "Ciao Michele, come va?",
16         "Come stai?",
17         "Buongiorno",
18         "Buonasera",
19         "Buon pomeriggio",
20         "Buonanotte"
21       ],
22       "responses": [
23         "Hey :-)",
24         "Ciao, grazie per avermi contattato, come posso esserti utile?",
25         "Ciao, io sono Michele, l'assistente virtuale per risolvere i tuoi
problemi di linea elettrica, come posso esserti utile?",
26         "Ciao, come posso aiutarti?"
27       ]
28     },
29     {
30       "tag": "goodbye",
31       "patterns": ["Bye", "See you later", "Goodbye"],
32       "responses": [
33         "See you later, thanks for visiting",
34         "Have a nice day",
35         "Bye! Come back again soon."
36       ]
37     },
38   ]
```

Intents - JSON

Cos'è un file .json?

JSON (JavaScript Object Notation) è un formato leggero per lo scambio di dati e viene comunemente utilizzato per la trasmissione e lo storage di dati strutturati.

Cos'è Intents.json?

Questo file JSON rappresenta un insieme di "intents" (intenzioni) utilizzate per addestrare un modello di chatbot.

intent è associato a un determinato tipo di domanda o input dell'utente

patterns (modelli) che rappresentano possibili varianti di come l'utente può esprimere quell'intenzione.

Ogni intent contiene anche una lista di "responses" (risposte) che il chatbot può fornire quando riconosce quella specifica intenzione.



Train

Fasi di esecuzione del file Train.py

1. Importazione Librerie

2. Lettura dei dati dal file JSON:

Carica i dati da un file JSON chiamato intents.json e li memorizza in una variabile chiamata intents.

3. Preparazione dei dati di addestramento:

- Estrazione dei pattern e i relativi tag dai dati di intents.
- Tokenizzazione dei pattern, eliminazione delle parole e applicazione dello stemming.
- Creazione di un elenco di tutte le parole presenti nei pattern.
- Crea una lista di coppie (pattern, tag) per l'addestramento.

```
14 with open('intents.json','r') as f:  
15     intents = json.load(f)  
16
```

```
22 for intent in intents['intents']:  
23  
24     # Nella variabile tag viene inserito il 'tag' di intent  
25     tag = intent['tag']  
26  
27     # Aggiunge il tag alla lista dei tags  
28     tags.append(tag)  
29  
30     # Ciclo for che itera per ogni pattern nella lista patterns di intent  
31     for pattern in intent['patterns']:  
32         w = tokenize(pattern)  
33  
34         # Aggiunge tutte le parole alla lista all_words  
35         all_words.extend(w)  
36  
37         # Aggiunge il pattern e il tag alla lista xy  
38         xy.append((w,tag))
```

```
49 for (pattern_sentence, tag) in xy:  
50     bag = bag_of_words(pattern_sentence,all_words)  
51     X_train.append(bag)  
52  
53     label = tags.index(tag)  
54     y_train.append(label) # CrossEntropyLoss
```

Model

```
● ● ●  
1 import torch  
2 import torch.nn as nn  
3  
4  
5 class NeuralNet(nn.Module):  
6     def __init__(self, input_size, hidden_size, num_classes):  
7         super(NeuralNet, self).__init__()  
8         self.l1 = nn.Linear(input_size, hidden_size)  
9         self.l2 = nn.Linear(hidden_size, hidden_size)  
10        self.l3 = nn.Linear(hidden_size, num_classes)  
11        self.relu = nn.ReLU()  
12  
13    def forward(self, x):  
14        out = self.l1(x)  
15        out = self.relu(out)  
16        out = self.l2(out)  
17        out = self.relu(out)  
18        out = self.l3(out)  
19        # no activation and no softmax at the end  
20        return out
```

Import delle librerie:

Importa le librerie PyTorch che forniscono funzionalità per la creazione e l'addestramento di reti neurali.

Definizione della classe NeuralNet:

Definisce una classe chiamata NeuralNet, che eredita dalla classe nn.Module di PyTorch. Questo è il modo standard per definire reti neurali in PyTorch.

Metodo __init__:

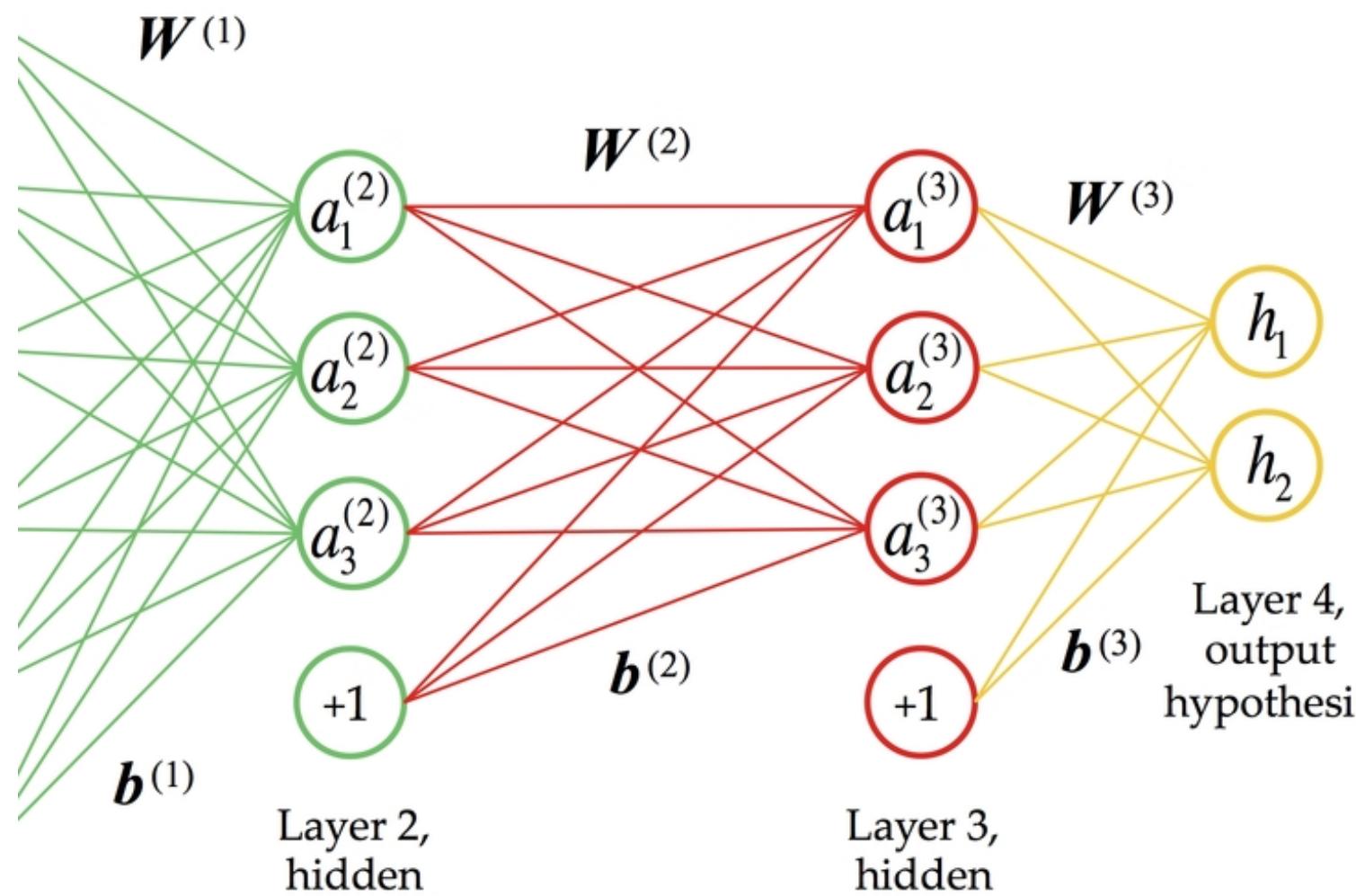
Il metodo __init__ è il costruttore della classe.

Viene chiamato quando un'istanza della classe viene creata.

In questo metodo vengono definiti i layer della rete neurale e altre impostazioni iniziali.

- **input_size:** Dimensione dell'input della rete neurale, corrispondente al numero di features in input.
- **hidden_size:** Dimensione dello strato nascosto della rete neurale, cioè il numero di neuroni nello strato nascosto.
- **num_classes:** Numero di classi di output della rete neurale.

Model



Definizione dei layer della rete neurale:

- self.l1, self.l2, self.l3: Strati lineari (fully connected) della rete neurale. nn.Linear definisce uno strato lineare che applica una trasformazione lineare agli input.
- self.relu: Funzione di attivazione ReLU (Rectified Linear Unit). ReLU è una funzione di attivazione comunemente utilizzata nelle reti neurali per introdurre non linearità.

Metodo forward:

Il metodo forward definisce il flusso di dati attraverso la rete neurale.

Output:

L'output del metodo forward è l'output finale della rete neurale, che non ha attivazione finale.

Questo significa che la rete restituisce direttamente l'output dei suoi strati lineari senza applicare alcuna funzione di attivazione o normalizzazione.

Chat.py

Questa parte di codice del progetto si occupa degli ultimi passaggi di calcolo, sulla scelta della risposta più appropriata e sull'aspetto grafico della pagina web.

1. Importazione Librerie e classi

```
● ○ ●  
1 import random  
2 import json  
3 import torch  
4 from model import NeuralNet  
5 from nltk_utils import bag_of_words, tokenize  
6 import streamlit as st  
7 from streamlit import session_state
```

2. Impostazione pagina web con Streamlit

```
● ○ ●  
20 st.set_page_config(page_title="ChatBot - AI", page_icon="robot.png",  
layout="centered", initial_sidebar_state="expanded")
```

4. Ripristino di un modello di rete neurale precedentemente addestrato

```
● ○ ●  
37 st.markdown(gradient_text_html, unsafe_allow_html=True)  
38 st.caption("L'assistente virtuale per rispondere alle tue domande su  
i problemi del tuo piano telefonico, attivo 24/7!")  
39  
40 user_question = st.chat_input("Fai la domanda:")  
41  
42 input_size = data["input_size"]  
43 hidden_size = data["hidden_size"]  
44 output_size = data["output_size"]  
45 all_words = data['all_words']  
46 tags = data['tags']  
47 model_state = data["model_state"]  
48  
49 model = NeuralNet(input_size, hidden_size, output_size).to(device)  
50 model.load_state_dict(model_state)  
51 model.eval()  
52  
53 bot_name = "Michele"
```

Chat.py

1. Recupero dei parametri e dei dati del modello dal file salvato

- I parametri importanti del modello, come le dimensioni degli strati e lo stato del modello stesso, sono caricati dal file `pth`.

2. Inizializzazione del modello con i parametri precedentemente salvati:

- Le dimensioni degli strati dell'input, nascosti e di output vengono assegnate alle variabili `input_size`, `hidden_size` e `output_size`, rispettivamente.
- Le parole utilizzate e i tag associati vengono recuperati dalle variabili `all_words` e `tags`.
- Lo stato del modello, contenente i pesi e i bias addestrati, viene caricato dalla variabile `model_state`.

3. Creazione del modello e caricamento dello stato addestrato:

- Un'istanza della classe `NeuralNet` viene creata utilizzando le dimensioni degli strati precedentemente recuperate.
- Lo stato del modello addestrato viene caricato nel modello appena creato utilizzando il metodo `load_state_dict()`. Questo metodo carica i pesi e i bias del modello.

4. Impostazione del modello in modalità di valutazione:

- Il modello viene impostato in modalità di valutazione utilizzando il metodo `eval()`.



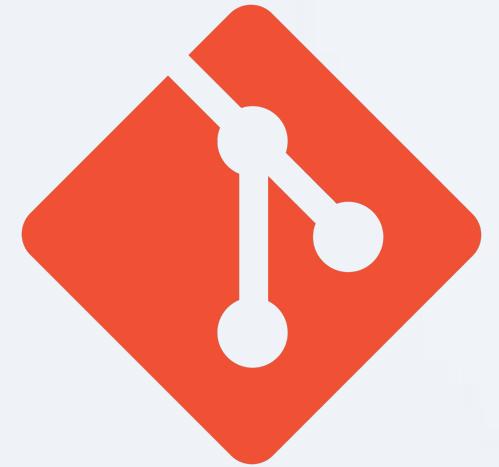
```
37 st.markdown(gradient_text_html, unsafe_allow_html=True)
38 st.caption("L'assistente virtuale per rispondere alle tue domande su
i problemi del tuo piano telefonico, attivo 24/7!")
39
40 user_question = st.chat_input("Fai la domanda:")
41
42 input_size = data["input_size"]
43 hidden_size = data["hidden_size"]
44 output_size = data["output_size"]
45 all_words = data['all_words']
46 tags = data['tags']
47 model_state = data["model_state"]
48
49 model = NeuralNet(input_size, hidden_size, output_size).to(device)
50 model.load_state_dict(model_state)
51 model.eval()
52
53 bot_name = "Michele"
```

Chat

Questa parte di codice del progetto si occupa degli ultimi passaggi di calcolo, sulla scelta della risposta più appropriata e sull'aspetto grafico della pagina web.

```
● ○ ●  
87 if user_question:  
88     response = "You: " + user_question  
89  
90     sentence = user_question  
91     session_state.listamessaggi.append(response)  
92     response_text = ""  
93     while True:  
94         sentence = tokenize(sentence)  
95         X = bag_of_words(sentence, all_words)  
96         X = X.reshape(1,X.shape[0])  
97         X = torch.from_numpy(X).to(device)  
98  
99         output = model(X)  
100        _, predicted = torch.max(output, dim=1)  
101        tag = tags[predicted.item()]  
102  
103        probs = torch.softmax(output, dim=1)  
104        prob = probs[0][predicted.item()]
```

```
● ○ ●  
106    if prob.item() > 0.75:  
107        for intent in intents['intents']:  
108            if tag == intent["tag"]:  
109                risp = random.choice(intent['responses'])  
110                print(f"{bot_name}: {risp}")  
111                response_text = f"{bot_name}: {risp}"  
112                session_state.listamessaggi.append(response_text)  
113                print(session_state.listamessaggi)  
114                for messaggio in session_state.listamessaggi:
```



GitHub & Git



Cos'è GitHub?

GitHub è una piattaforma di sviluppo di software basata su cloud che consente agli sviluppatori di collaborare, gestire e condividere progetti di codice sorgente.

Alcune caratteristiche principali di GitHub:

1. Controllo della versione:

GitHub utilizza il sistema di controllo delle versioni Git, che consente agli sviluppatori di tenere traccia delle modifiche apportate al codice nel tempo.

2. Repository:

I progetti su GitHub sono organizzati all'interno di repository.

3. Collaborazione:

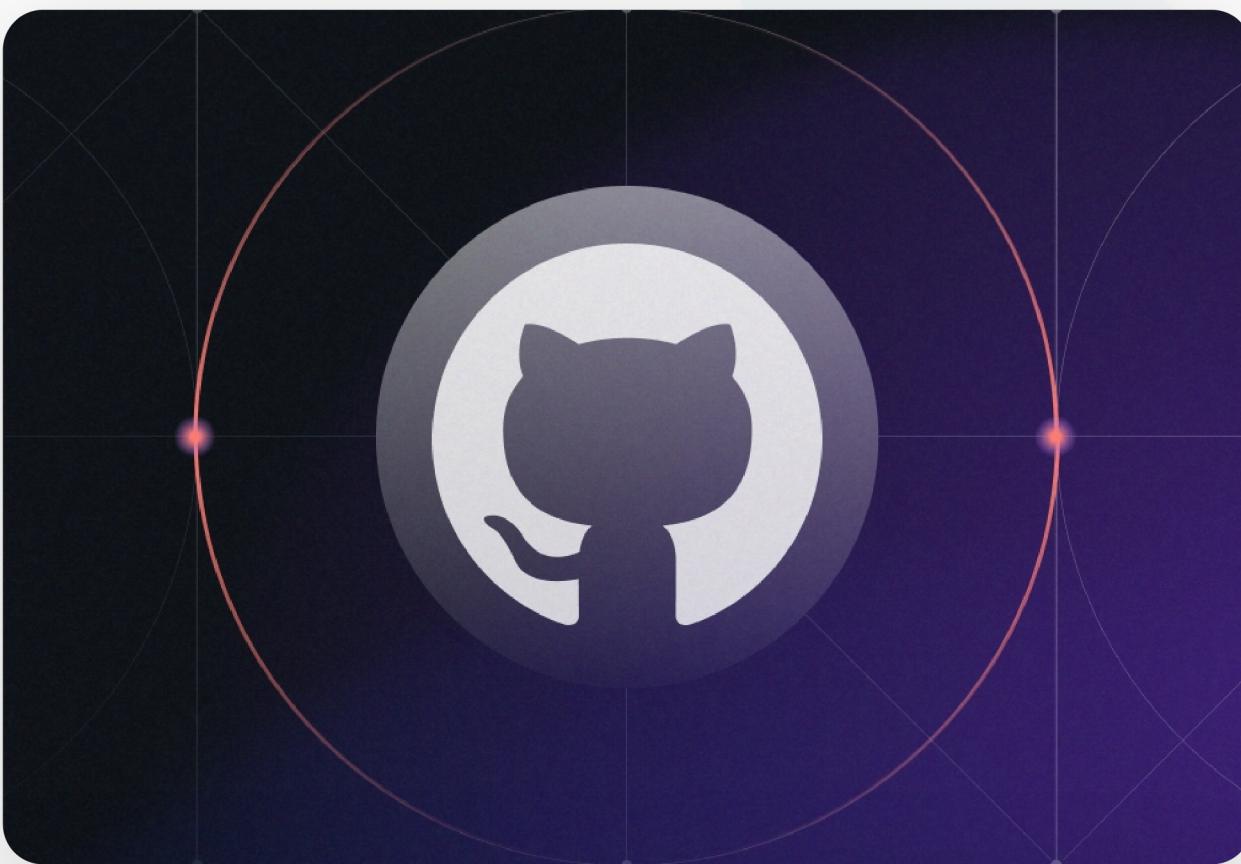
GitHub facilita la collaborazione tra gli sviluppatori consentendo loro di lavorare insieme su un progetto, apportare modifiche, commentare il codice e revisionare il lavoro degli altri.

4. Gestione dei problemi e delle attività:

GitHub offre strumenti per tracciare problemi, bug e richieste di funzionalità all'interno di un progetto. Questi strumenti consentono agli sviluppatori di tenere traccia delle attività da completare, assegnare compiti, impostare scadenze e monitorare lo stato dei problemi.

5. Documentazione:

GitHub consente agli sviluppatori di fornire documentazione dettagliata dei loro progetti utilizzando strumenti come i file README e le Wiki.



GitHub Copilot

Technical preview

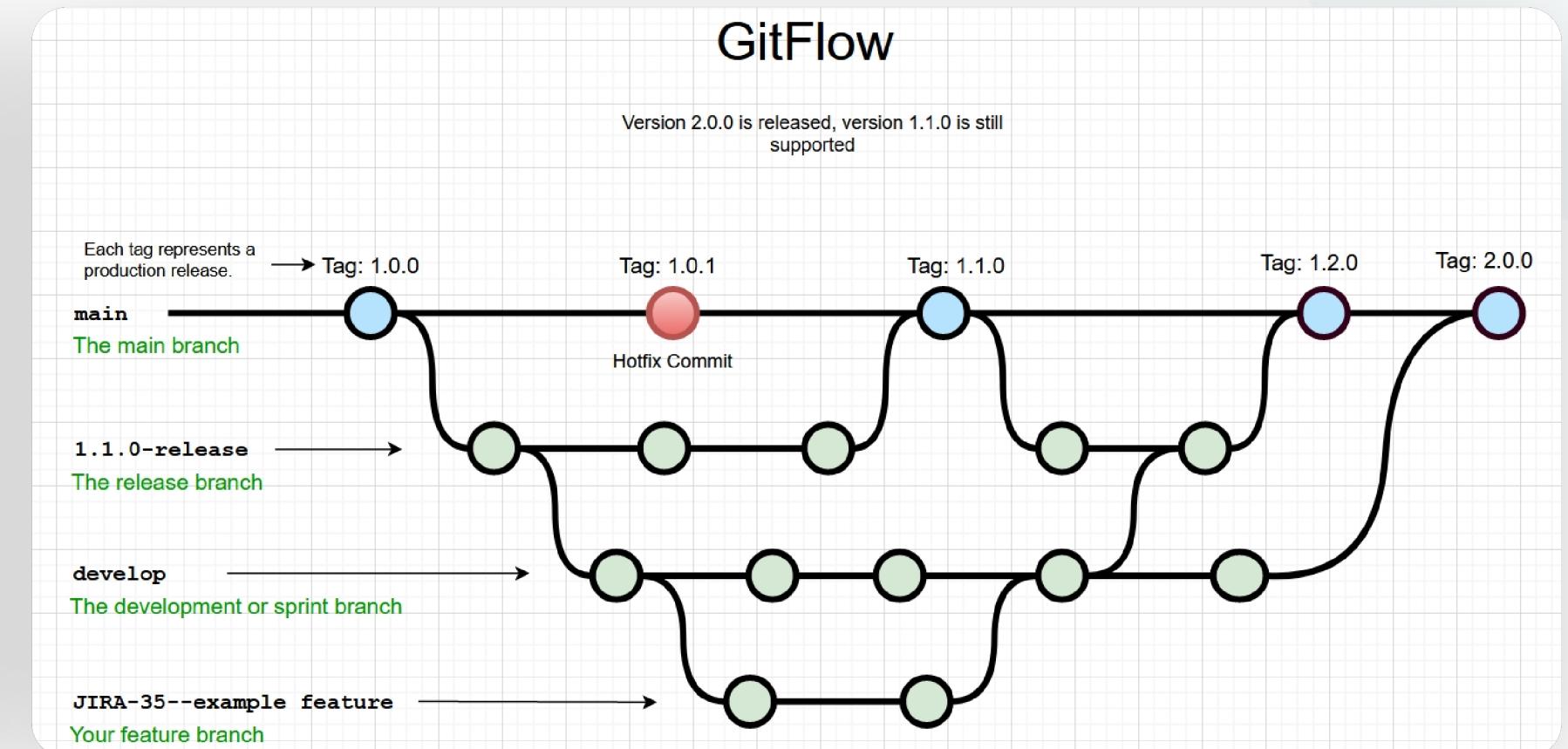
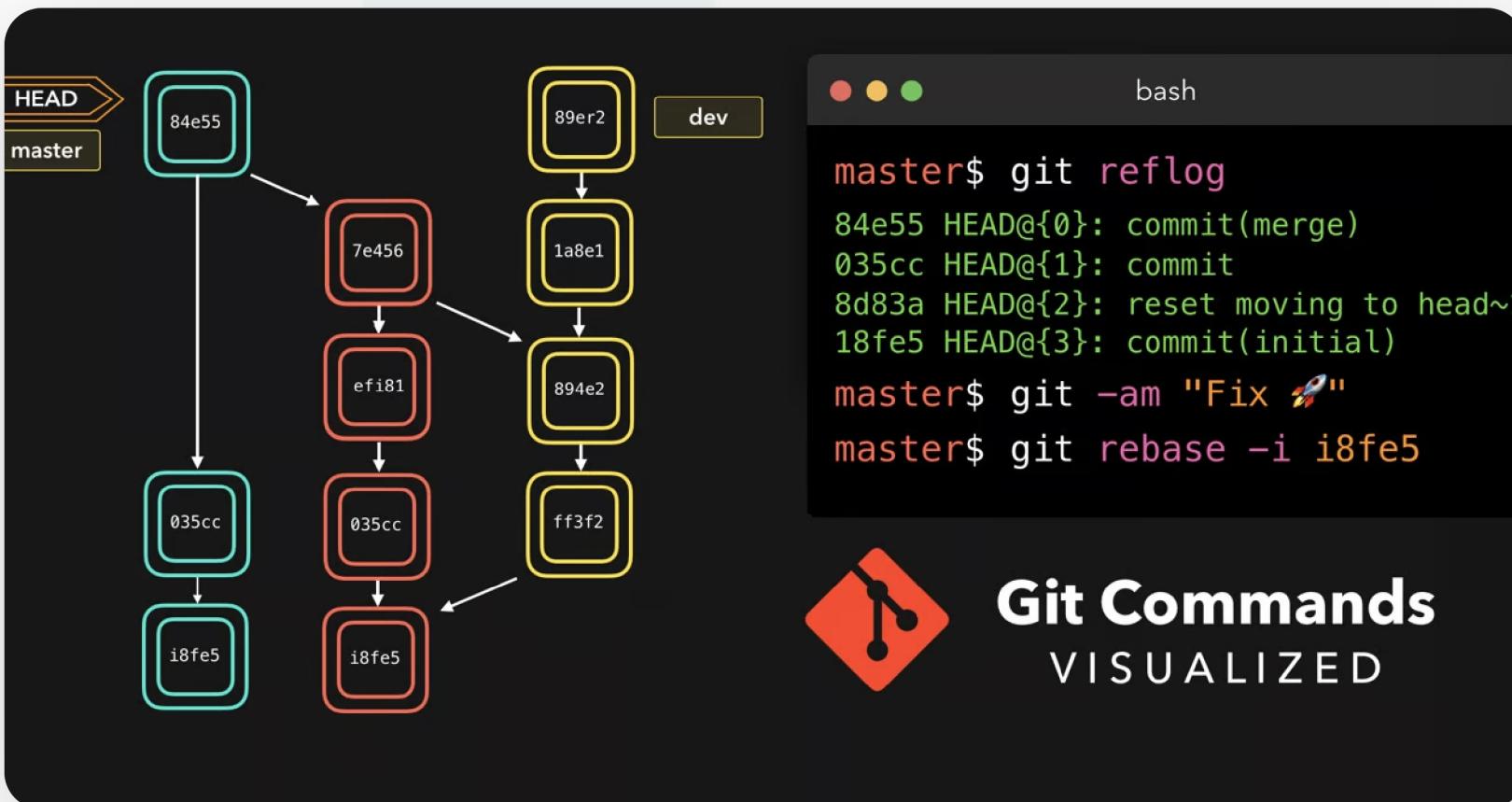
Your AI pair
programmer

fetch_pic.js

```
1 const fetchNA
2   return feto
3   method: 'GET'
4   headers: {
5     'Content-Type': 'application/json'
6   }
7 }
8 .then(res)
9 .then(jsd
10  return
11 );
12 }
```

Copilot

Struttura di GIT



La struttura di base di Git è relativamente semplice e si compone di tre principali concetti:

Repository (Repository Git):

- Un repository Git è una collezione di file di progetto e la storia delle modifiche ad essi associate.
- Un repository può essere locale (sul tuo computer) o remoto (su un server Git come GitHub, GitLab o Bitbucket).

Commit:

E' una singola revisione o snapshot di un repository in un determinato momento nel tempo.

Branches (Rami):

E' un ramo separato della storia del repository che può divergere dal ramo principale (di solito chiamato `master` o `main`).

Sono utilizzati per sviluppare nuove funzionalità in modo isolato dal ramo principale.

Dopo aver completato lo sviluppo su un ramo, le modifiche possono essere integrate nel ramo principale attraverso un processo chiamato merge o rebase.



**Grazie per
l'attenzione.**





Want to make a presentation like this one?

Start with a fully customizable template, create a beautiful deck in minutes, then easily share it with anyone.

[Create a presentation \(It's free\)](#)