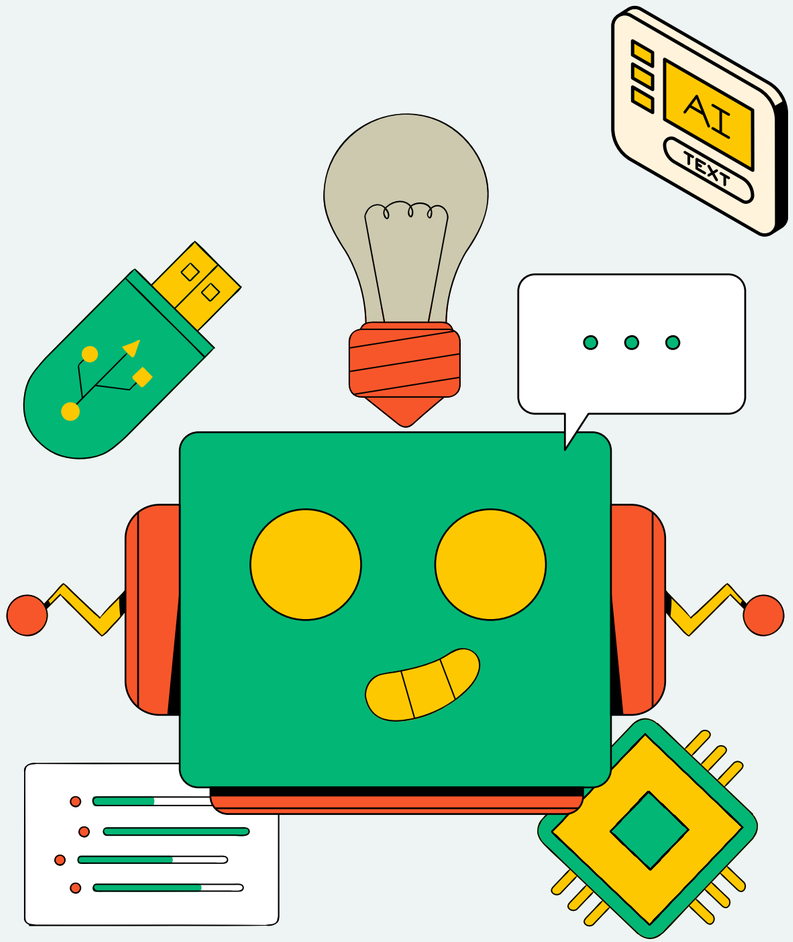
THINK UNLIMITED

WE LEARN FOR THE FUTURE



AROL GROUP AI CHATBOT

PRESENTATION

PRESENTED BY:

ALESSANDRO MANERA

PRESENTATION OUTLINE

- Data preparation
- Fine-tuning an LLM
- TF-IDF & Cosine Similarity
- Inference
- UI
- Problems I faced
- Solutions
- Questions and Answers

DATA PREPARATION

Scraping Arol webpages

Using BeautifulSoup to parse HTMLs

File cleaning

SCRAPING AROL WEBPAGES

```
import requests
import os
from bs4 import BeautifulSoup

url_home = 'https://www.arol.com/'
url_customer_care = 'https://www.arol.com/customer-care-for-capping-machines'
url_news_events = 'https://www.arol.com/news-events'
url_company = 'https://www.arol.com/arol-canelli'
url_arol_group = 'https://www.arol.com/arol-group-canelli'
url_work_with_us = 'https://www.arol.com/work-with-us'
url_contacts = 'https://www.arol.com/arol-contact'

urls = [url_home, url_customer_care, url_news_events, url_company, url_arol_group, url_work_with_us, url_contac
```

USING BEAUTIFULSOUP TO PARSE HTMLS

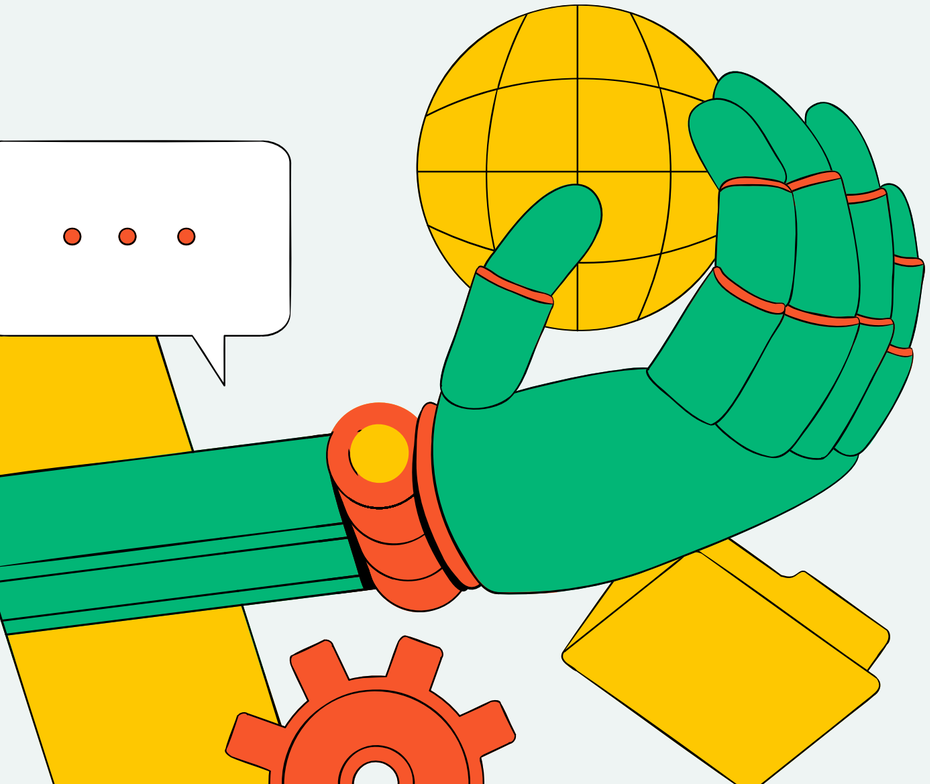
```
for url in urls:
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    #content = soup.prettify()
    text = soup.get_text(separator="\n")

    # Remove extra white spaces and blank lines
    #lines = [line.strip() for line in text.splitlines() if line.strip()]
    #cleaned_text = "\n".join(lines)

    filename = f'{url.split("//")[1].replace("/", "_")}.txt'
    if not os.path.exists(filename):
        with open(filename, 'w') as file:
            file.write(text)
```

FILE CLEANING

```python
# Directory where your text files are stored
input_files = ["www.arol.com_.txt", "www.arol.com_customer-care-for-capping-machines.txt", "www.arol.com_news
output_file = "merged_cleaned.txt"

# Function to clean text
def clean_text(text):
    # Remove excessive blank lines and spaces
    text = re.sub(r"\n\s*\n", "\n"                    le blank lines
    text = re.sub(r"\s{2,}", " ", text)  # Replace multiple spaces with a single space

    # Remove typical website navigation elements
    navigation_keywords = [
        "Home", "Sectors", "Customer care", "News & Events", "Company",
        "Arol Group", "Work with us", "Contacts", "Terms & Conditions",
        "Privacy Policy", "Cookie Policy", "Legal Notes", "Code of Ethics"
    ]

    # Remove lines that contain navigation words
    text = "\n".join(
        line for line in text.split("\n") if not any(nav in line for nav in navigation_keywords)
    )

    # Strip leading/trailing spaces from each line
    text = "\n".join(line.strip() for line in text.split("\n") if line.strip())

    return text
```

(parameter) text: str

FINE-TUNING AN LLM



It is the process of training a pre-existing model on a smaller, task-specific dataset to improve its performance for a particular use case.

This involves adjusting the model's weights using additional data while retaining its general language understanding.

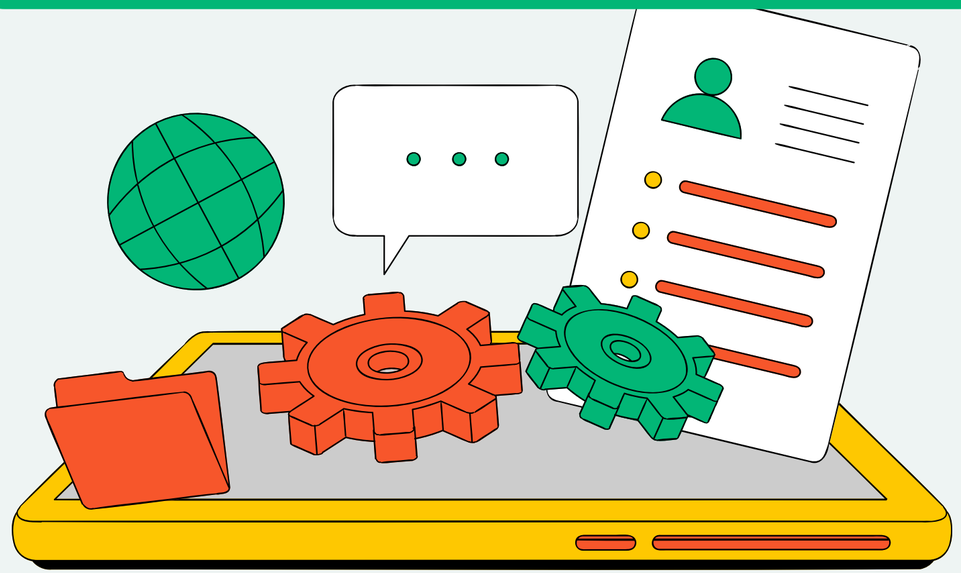
Fine-tuning helps customize the model for domain-specific knowledge, tone, or response accuracy

FINE-TUNING AN LLM

LoRA (Low Rank Adaptation)

8-bit quantization

PEFT (Parameter-Efficient Fine-Tuning)



FINE-TUNING WITH LORA 1/2

```python
# Load dataset from JSON
dataset = load_dataset("json", data_files="qa_pairs.json")  # Ensure JSON format is valid

# Load tokenizer and model
model_name = "meta-llama/Llama-2-7b-chat-hf"
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.pad_token = tokenizer.eos_token
  ⌥C Duo Quick Chat
# Function to tokenize input
def tokenize_function(examples):
    inputs = tokenizer(examples["question"], examples["answer"],
                       truncation=True, padding="max_length", max_length=512)

    # Set labels to be the same as input_ids
    inputs["labels"] = inputs["input_ids"].copy()

    return inputs

# Apply tokenization
tokenized_dataset = dataset.map(tokenize_function, batched=True)

# Load model in LoRA mode (for memory-efficient fine-tuning)
model = AutoModelForCausalLM.from_pretrained(model_name, device_map="auto", load_in_8bit=True)
```

TF-IDF & COSINE SIMILARITY

Term Frequency (TF)

Inverse Document Frequency (IDF)

Cosine similarity: treating them as vectors in a multi-dimensional space

TF-IDF & COSINE SIMILARITY 1/2

```
prompt = ChatPromptTemplate.from_template(template)
chain = prompt | model

abstracts = [
    "This is the homepage of Arol Group, which provides various industrial solutions including capping machines
    "This page discusses Arol Group's customer care services, specifically for capping machines, offering suppo
    "This page covers the latest news and events related to Arol Group, including product launches, industry up
    "Arol Canelli is part of Arol Group and is located in Canelli, Italy. The company specializes in designing
    "Arol Group is an international leader in the packaging industry, providing innovative solutions for bottli
    "This page invites potential candidates to work with Arol Group, describing the company culture, career opp
    "Arol Group provides contact information for clients, suppliers, and potential partners, facilitating commu
]

# Mapping the abstracts to corresponding file paths (representing each URL)
myDict = {
    abstracts[0]: "www.arol.com_.txt", #home
    abstracts[1]: "www.arol.com_customer-care-for-capping-machines.txt", #customer_care
    abstracts[2]: "www.arol.com_news-events.txt", #news_events
    abstracts[3]: "www.arol.com_arol-canelli.txt", #company
    abstracts[4]: "www.arol.com_arol-group-canelli.txt", #arol_group
    abstracts[5]: "www.arol.com_work-with-us.txt", #work_with_us
    abstracts[6]: "www.arol.com_arol-contact.txt" #contacts
}
```

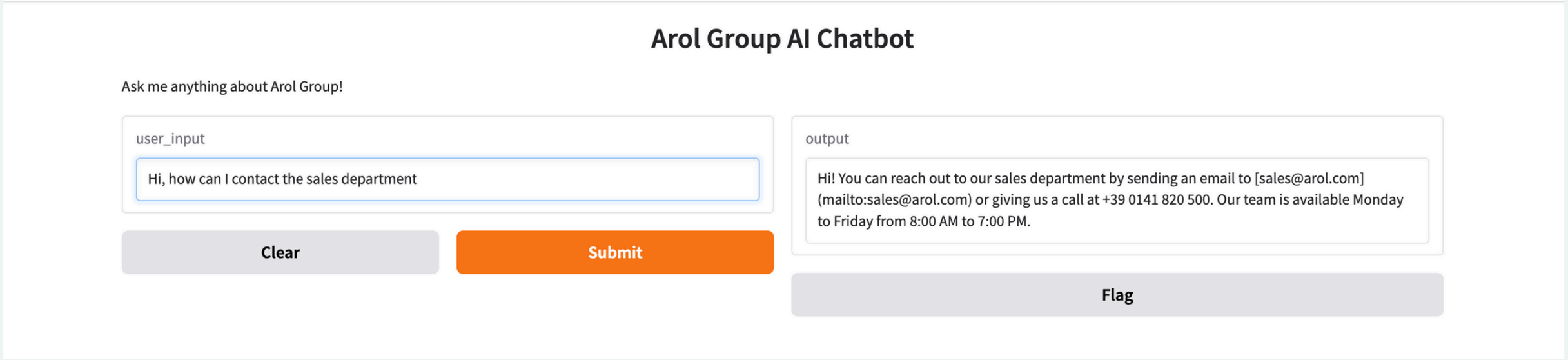
TF-IDF & COSINE SIMILARITY 2/2

```python
def prepare_context(input_text):
    # Compute similarity between input_text and abstracts
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(abstracts + [input_text])  # Include user input in the matrix
    similarities = cosine_similarity(tfidf_matrix[-1], tfidf_matrix[:-1])  # Compare input_text with abstracts
    most_similar_idx = similarities.argmax()  # Get index of the most similar abstract
    most_similar_score = similarities[0, most_similar_idx]

    print(f"Matched abstract: {abstracts[most_similar_idx]}")
    print(f"Similarity score: {most_similar_score}")

    if most_similar_score >= 0.01:  # Threshold
        matched_abstract = abstracts[most_similar_idx]
        file_path = myDict.get(matched_abstract)
        print(f"Most similar index: {most_similar_idx}")
        print(f"File path: {file_path}")
        if file_path and os.path.exists(file_path):
            with open(file_path, "r") as f:
                return f.read()
        else:
            return f"Error: The file for '{matched_abstract}' was not found."
    else:
        return "I'm sorry, I couldn't find relevant information for your query."
```

UI: GRADIO

- Quick Interface Creation
- Easy Integration
- User-Friendly
- Support for Multiple Input/Output Types

GRADIO

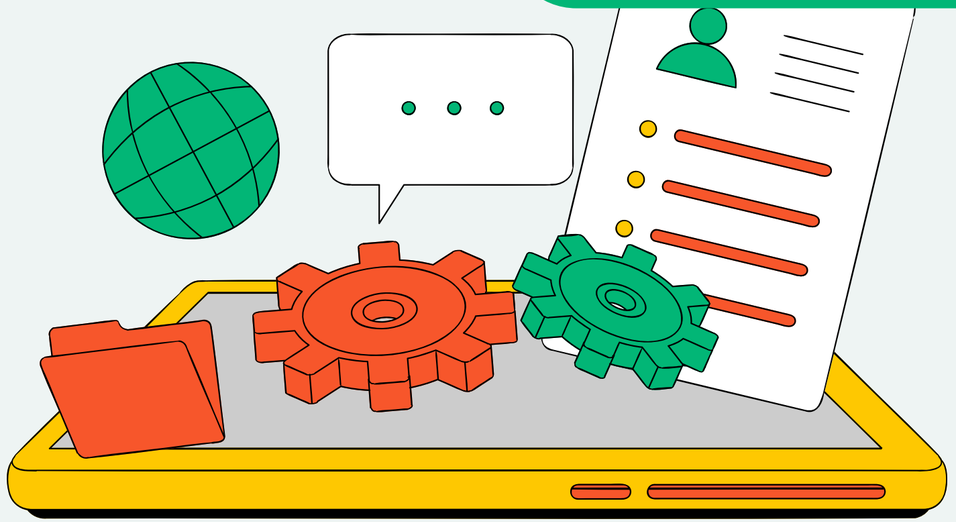

Arol Group AI Chatbot
Ask me anything about Arol Group!
user_input
Hi, how can I contact the sales department
Clear
Submit
output
Hi! You can reach out to our sales department by sending an email to [sales@arol.com] (mailto:sales@arol.com) or giving us a call at +39 0141 820 500. Our team is available Monday to Friday from 8:00 AM to 7:00 PM.
Flag

MODEL INFERENCE THROUGH HUGGING FACE

It exploits text-generation pipeline to generate responses based on user input.

load a pre-trained model

latency depends on the model size and available hardware



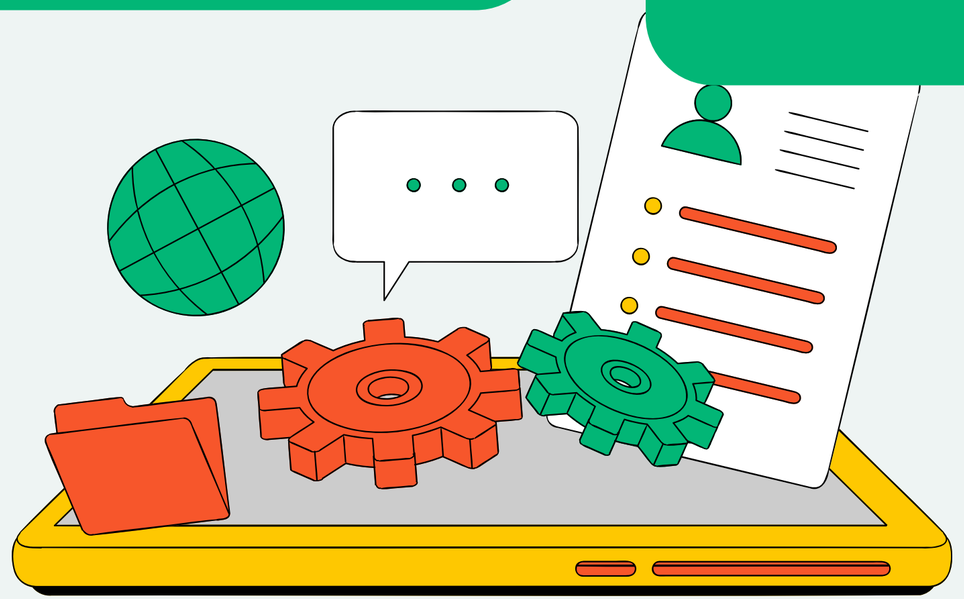
PROBLEMS

Not enough CPU

No GPU

Too generic abstracts, due to the lack of a proper database

Inference API not recognizing my model

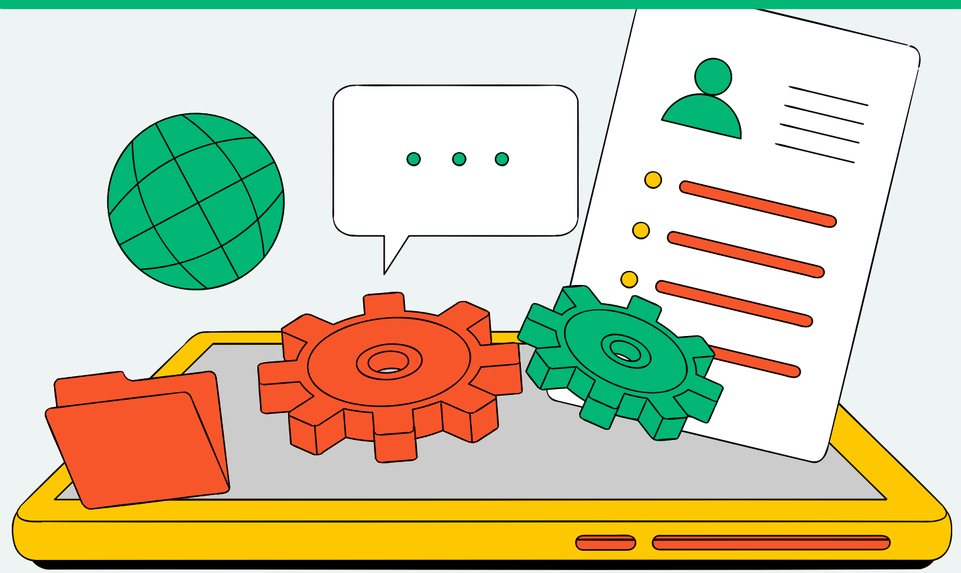


SOLUTIONS

Ollama instead of llama

Merging in one single file

Run the model locally



OLLAMA

Run your fine-tuned model locally

No rate limits or API costs

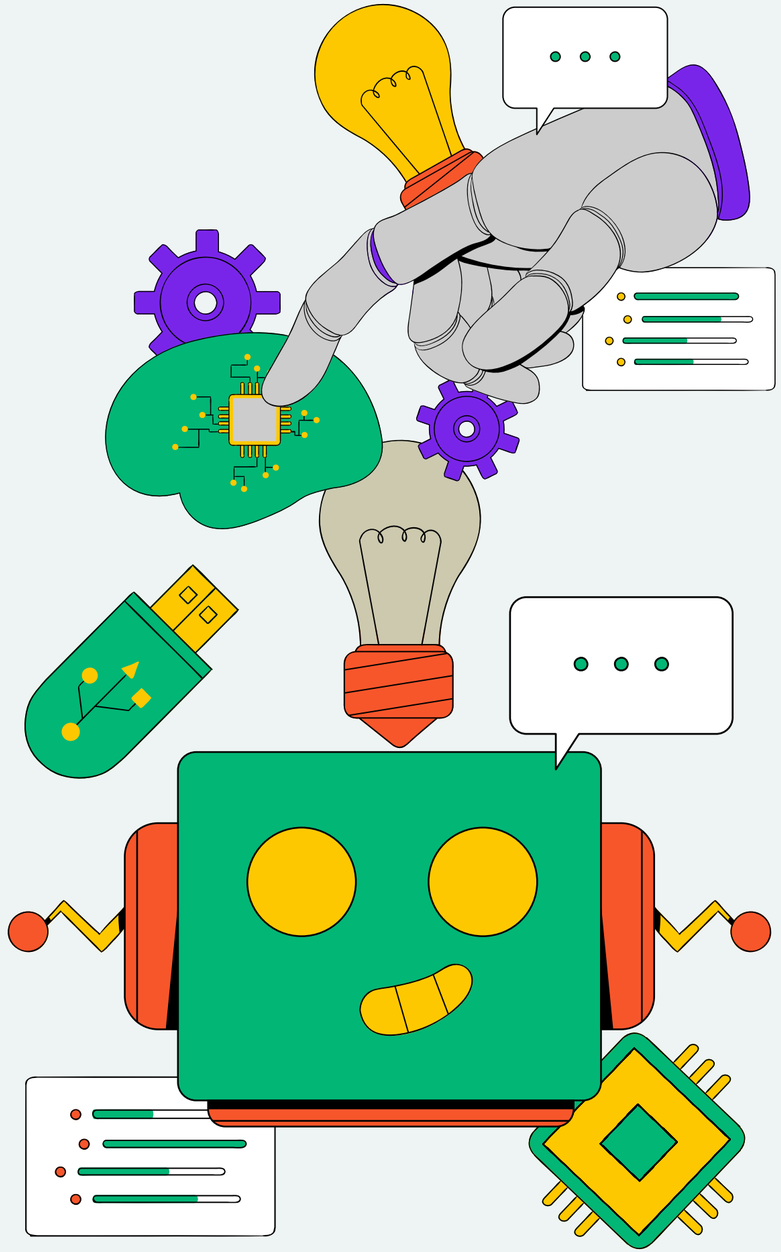
Faster response times

MERGING INTO ONE SINGLE FILE

```
# Merge all files
merged_text = ""

for file in input_files:
    with open(file, "r", encoding="utf-8") as f:
        content = f.read()
        cleaned_content = clean_text(content)
        merged_text += cleaned_content + "\n\n"  # Add spacing between files

# Save cleaned merged text
with open(output_file, "w", encoding="utf-8") as out_file:
    out_file.write(merged_text)
```

THANKS FOR YOUR ATTENTION

- Now let’s head to a short demo...



Link to GitHub repositiory:

https://github.com/AleManera/AI-ChatBot