# SISTEMAS COMPUTACIONAIS AVANÇADOS (SISTCA)

# ADVANCED COMPUTING SYSTEMS

LICENCIATURA EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
POLITÉCNICO DO PORTO

**Lab Classes Script:**

## Local Chat Agent

**Table of Contents**

## List of figures and tables

# 1.Introduction

## 1.1 Scientific/Technological Context

Currently, artificial intelligence, more specifically machine learning, has been advancing at an exponential pace in recent years and has had a significant impact on different areas, such as industry, education and our society. The advancement of different language models that are based on deep neural networks, particularly those that use the Transformer architecture, is one of the most significant advances, as the models can understand and create text with a high level of cohesion and context, which has completely changed natural language processing.

Transformers work using different types of mechanisms that recognize connections between words in different sentences, regardless of their separation. This ability is useful for different types of tasks, such as natural language production, text summarization, automatic translation and answering different questions. Furthermore, thanks to the increase in computing power, we have been able to train increasingly complex and powerful models.

Thanks to improvements in model optimization and update approaches, new artificial intelligence systems can now operate on different devices with lower energy consumption, such as smartphones or personal servers. Thanks to this new technology, various types of tools have become very useful in assisting programming, such as interactive chatbots and virtual assistants.

On a technical and ethical level, the advancement of artificial intelligence has also brought about different new questions and difficulties, which has led the scientific and technological communities to pay increasing attention to issues such as data vision, information privacy and environmental impact. Due to these new difficulties and challenges, constant efforts are being made to improve the accessibility, security and transparency of these systems, so that everyone can truly benefit from their use.

## 1.2 Motivation for this topic/lab script

We chose the Cloud AI Agents theme because of the great increase in the importance of artificial intelligence in our daily lives, especially in the area of natural language processing. Nowadays, the ability of computers to understand and create text in a cohesive way is transforming the way we interact with this technology in different areas, from education to health and even entertainment. With recent technological advances, it has become possible to study, experiment and test with different language models in an accessible way even with limited resources.

With this project we have the opportunity to deepen the knowledge about how these models work in practice, investigating different concepts, such as neural networks, model training and language interpretation. Our main motivation is, therefore, to learn in a practical and critical way how artificial intelligence is shaping the future of our communication and technology, while developing different types of technical skills.

## 1.3 Objectives (of this lab script)

The main objective of this project is to analyze the installation, configuration and practical use of differents open sources language models, with a special focus on the TinyLlama 1.1B and LLaMA 3B v2 models. With this project, we intend to explore how these models can be executed locally, even when machines have limited resources, and demonstrate how we can interact with them in different ways, such as through command lines and graphical interfaces.

In addition, the report aims to demonstrate how we can make the most of the capabilities of these models in natural language processing tasks, showing their potential in real-world contexts, such as virtual assistants, educational tools and prototypes of conversational applications.

## 1.4 Structure (of this lab script)

The report is structured in several sections. It begins with a brief introduction to the topic, explaining the importance of artificial intelligence in natural language processing. A theoretical component is then presented, covering different types of concepts, such as the Transformer architecture, LLaMA models and the tools used.

The process of installing and configuring the TinyLlama 1.1B and LLaMA 3B v2 models for local execution is then described. It then presents the implementation of different interfaces for interacting with the models, via the command line, through a Telegram bot and through a web interface. Finally, practical exercises are presented and the main conclusions of the project are summarized.

# 2 Theoretical (scientific/technological background)

## 2.1 Main theoretical concepts/terminology

To understand the project proposal we have to explain different theoretical concepts that are necessary to understand the development of language models. As a first topic, we have to know what language models are, which are systems based on neural networks with the function of understanding and generating language in a natural way. Language models will be able to perform their tasks such as translation and text generation by learning large-volume statistical patterns.

Next we have the concept of transformer, which is an architecture that is based on attention mechanisms, this architecture will allow us to have parallel processing of tokens, which will guarantee us greater performance and efficiency. The parameters of a model are adjustable units and are related to the size and complexity of a model.

In a language model, this can be either open source or proprietary depending on its license, depending on the license it defines the rights of use and modification

of the model. When the model is open source there is transparency and free customization, but when it is proprietary the license restricts access.

Finally, a very important concept is the model interaction interface, which are the ways the user can communicate with the language model.

An interaction interface is the command line that will allow the user to interact with the model through the terminal. Another interface is the Telegram bot that will allow the user to interact through the Telegram platform, allowing users to communicate in a familiar and accessible environment. Finally, the web interface is a graphical environment accessible through the browser.

## 2.2 The State of the Art

With the development of language models and the way they have revolutionized the field of natural language processing, there have been several advances in various tasks, such as text generation, automatic translation and even dialogue with users. Due to the importance of these factors, choosing the right model has become crucial.

To choose the right language model, we need to consider several factors, such as its computational capacity, its language, the complexity of the task and the type of license. Table 1 below summarizes and compares the different factors of several LLMs widely used today.

| Modelo | Número de parametros | Idiomas | Requisitos Hardware | Licença | Treinado por |
|---|---|---|---|---|---|
| LLaMA 3B V2 | 3 Biliões | Multilingue | Médio/Alto | Open | Meta AI |
| TinyLlama 1.1B | 1.1 Biliões | Multilingue | Baixo | Open | Comunidade |
| GPT-2 | 1.5 Biliões | Inglês | Médio | Open | Open AI |
| GPT-3.5 | 6.7 Biliões | Inglês/Multilingue | Alto | Proprietária | Open AI |
| Mistral 7B | 7 Biliões | Multilingue | Alto | Open | Mistral AI |
| BERT Base | 110 Milhões | Inglês | Baixo | Open | Google |
| Falcon | 40 Biliões | Multilingue | Alto | Open | Technology Innovation Institute |
| LLaMA 2 | 70 Biliões | Multilingue | Muito Alto | Open | Meta AI |

**Table 1- Comparison of various LLMs currently in use**

As for the workflow that will exist in the project, it starts with the creation of the project and then its installation. After the installation of the model, the project will present different ways of interacting with the models. We can see the steps of the workflow in the figure represented below.

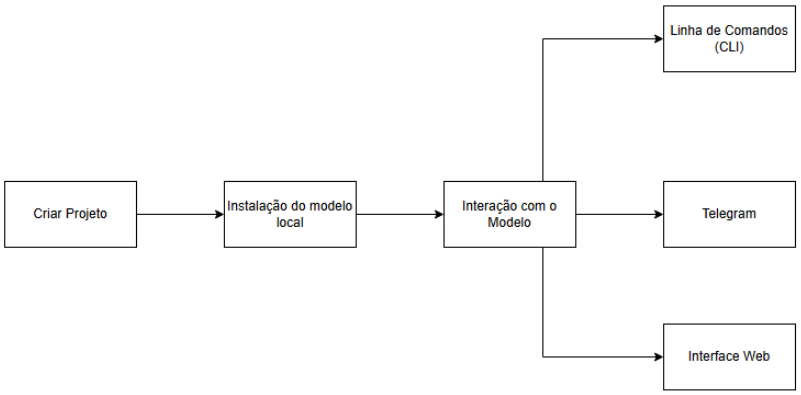



**Figure 1 - Flowchart of project stages**

# 3. Outlook of the technology

## 3.1 Brief Overview of the TinyLlama-1.1B Model

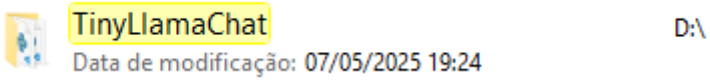
TinyLlama-1.1 B-Chat-v1.0 is a lightweight, open-source language model with approximately 1.1 billion parameters. It is optimized for chat-based tasks and balances performance and hardware efficiency well. Because of its compact size, it can run locally on personal machines. The model is available for free via the Hugging Face platform.

## 3.2 Preparing the Environment

- **Step 1: Create the working directory**

Start by creating the folder where everything related to the model will be set up.

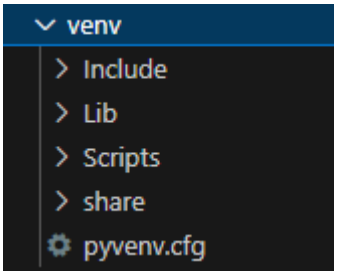
```
mkdir D:\TinyLlamaChat
cd D:\TinyLlamaChat
```



- **Step 2: Create a virtual environment**

To isolate the project dependencies, create a virtual Python environment:

```
python -m venv venv
```




- **Step 3: Activate the virtual environment**

```
.\venv\Scripts\activate
(venv) D:\TinyLlamaChat>
```

- **Step 4: Install Python dependencies**

Install the required Python packages:

```
pip install torch transformers
```

### 3.3 Hugging Face Access Token

While TinyLlama-1.1 B-Chat-v1.0 is public and does not require authentication, other large or gated models (such as Meta LLaMA or Mistral) require an access token and user permissions. Below are the steps to obtain and use such a token.

- **Step 1: If you don’t already have an account, sign up for free:**

  https://huggingface.co/join

- **Step 2: Generate an access token**

  2.1 Go to your account settings:

  https://huggingface.co/settings/tokens

  2.2 Click on **"New token"**.

  2.3 Choose a name for the token (e.g., model_access_token).

2.4 Select the required permission level:

- read: to download models (recommended).
- write: to push or edit repositories.
- fine-grained: to allow access to specific resources.

2.5 Click **"Generate a token"** and copy the generated token (hf_...).

https://huggingface.co/docs/hub/en/security-tokens

- **Step 3: Authenticate using the token**

3.1 Via Terminal:

```
huggingface-cli login
```

You'll be prompted to paste your token. Once authenticated, it will be stored locally.

3.2 Via Python

```
from huggingface_hub import login

login(token="hf_...")
```

3.3 Using an environment variable

```
set HF_TOKEN=hf_...      # On Windows
export HF_TOKEN=hf_...   # On Linux/macOS
```

- **Step 4: Request access for gated models**

  Some models, like **Meta LLaMA**, require approval:

  4.1 Visit the model page, for example:

  https://huggingface.co/meta-llama/Llama-2-7b-hf

  4.2 Click **"Request Access"** and accept the usage terms.

  4.3 Wait for approval by the model administrators.

- **Step 5: Use the token to download the model**

Once access is granted, load the model with the token:

```
from transformers import AutoModelForCausalLM, AutoTokenizer

token = "hf_..."

tokenizer = AutoTokenizer.from_pretrained("meta-llama/Llama-2-7b-hf", token=token)
model = AutoModelForCausalLM.from_pretrained("meta-llama/Llama-2-7b-hf", token=token)
```

## 3.4 Download and cache the model in drive D

Script: download_model.py

Now we will write a Python script to download and save the model in D:\TinyLlamaChat\hf_cache.

download_model.py > ...

```
# install_model.py
import os
from transformers import AutoTokenizer, AutoModelForCausalLM

# Desativa aviso de symlinks
os.environ["HF_HUB_DISABLE_SYMLINKS_WARNING"] = "1"

# ID do modelo no HuggingFace
model_id = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"
cache_dir = "D:/TinyLlamaChat/hf_cache"

print("A descarregar o modelo...")

# Download do modelo e tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_id, cache_dir=cache_dir)
model = AutoModelForCausalLM.from_pretrained(model_id, cache_dir=cache_dir)

print("Modelo instalado com sucesso em:", cache_dir)
```

```
(venv) D:\TinyLlamaChat>python download_model.py
A fazer download do modelo e tokenizer...
tokenizer.model: 100%|                                                    | 500k/500k [00:00<00:00, 3.69MB/s]
tokenizer.json: 100%|                                                   | 1.84M/1.84M [00:00<00:00, 2.76MB/s]
special_tokens_map.json: 100%|                                            | 551/551 [00:00<?, ?B/s]
config.json: 100%|                                                        | 608/608 [00:00<?, ?B/s]
model.safetensors: 100%|                                                | 2.20G/2.20G [10:37<00:00, 3.45MB/s]
generation_config.json: 100%|                                             | 124/124 [00:00<?, ?B/s]
Download concluído.
```

### 3.5 Verify the installation

Script: verify_model.py

Create a script called verify_model.py to check whether the model has been properly installed.

```python
# verify_model.py
import os
from transformers import AutoTokenizer, AutoModelForCausalLM

os.environ["HF_HUB_DISABLE_SYMLINKS_WARNING"] = "1"

model_id = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"
cache_dir = "D:/TinyLlamaChat/hf_cache"

try:
    tokenizer = AutoTokenizer.from_pretrained(model_id, cache_dir=cache_dir)
    model = AutoModelForCausalLM.from_pretrained(model_id, cache_dir=cache_dir)
    print("✅ Modelo carregado com sucesso a partir do disco D.")
except Exception as e:
    print("❌ Erro ao carregar o modelo:", str(e))
```

```
(venv) D:\TinyLlamaChat>python verify_model.py
✅ Modelo carregado com sucesso a partir do disco D.
```

Next, a script (chat.py) will be created to test the model's functionality. However, this process is part of the model's practical usage and will be further detailed in **Section 5** of this report.

## 4.Ways to Interact with the Model

After the local installation of the model (described in Section 4), there are several ways to interact with it. These methods facilitate testing, application integration, and practical use, adapting to different user profiles, from technical users to end users. Each interaction method has its own characteristics and advantages, allowing the model to be used in various contexts, according to the needs of the project. In this project, although the step-by-step guide was done using the TinyLlama 1.1B model, the interactions described in this section were carried out with the Open LLaMA 3B-V2 model, which offers higher-quality responses and allows for the generation of more complete examples.

## 4.1 Exercise 1: Interaction via command line interface

- Description

Interaction via the command line interface (CLI) allows sending questions or prompts directly to the model and receiving the corresponding responses in the terminal. This is a quick and simple way to test the functioning of the model without the need for graphical interfaces or additional integrations. The CLI is particularly useful for technical users who want to quickly validate the model, experiment with prompts, or integrate the model into automated scripts.

- Solution

Prepare the Python file with the code to interact with the model via the command line, using the script that, in this case, was named chat_CLI.py.

- Setup and Code Used

The following shows the code used:

chat_CLI.py > ...

```python
import torch
import os
from transformers import LlamaTokenizer, LlamaForCausalLM
from datetime import datetime

local_directory = "./open_llama_3b_v2_local"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"💻 Using device: {device}")

tokenizer = LlamaTokenizer.from_pretrained(local_directory, local_files_only=True, use_fast=False)
model = LlamaForCausalLM.from_pretrained(local_directory, local_files_only=True).to(device)

# Personalized initial message
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

print("\n Conversation started with Open LLaMA 3B v2 (type '/exit' to exit)")

while True:
    # Add current time in [HH:MM] format to user prompt
    time_now = datetime.now().strftime("%H:%M")
    user_input = input(f"\n[{time_now}] User: ")
    command = user_input.strip().lower()

    # Command to stop the interaction
    if command == "/exit":
        print("👋 Ending conversation. Goodbye!")
        break

    # Command to clear the terminal screen
```

```python
# Command to clear the terminal screen
if command == "/clear":
    os.system('cls' if os.name == 'nt' else 'clear')
    continue

inputs = tokenizer(user_input, return_tensors="pt", padding=True).to(device)

output_ids = model.generate(
    **inputs,
    max_new_tokens=100,
    pad_token_id=tokenizer.pad_token_id,
    eos_token_id=tokenizer.eos_token_id,
    do_sample=False,
    repetition_penalty=1.2  # Penalize repetitions in the response
)

response = tokenizer.decode(output_ids[0], skip_special_tokens=True)

# Remove the question from the beginning of the response if echoed
if user_input.lower() in response.lower():
    response = response.split(user_input, 1)[-1].strip()

# Remove simple repetitions (keep only the first line of the response)
response_lines = [line.strip() for line in response.split("\n") if line.strip()]
if response_lines:
    response = response_lines[0]

# Add current time prefix to the model's response
time_now = datetime.now().strftime("%H:%M")
print(f"[{time_now}] LLama3Bv2: {response}")
```

The model can be customized in various ways, and in this case, some customizations were implemented to improve the user experience, such as adding the time to the messages, defining personalized names for the user and the model, the command to clear the terminal, the command to stop the interaction, and filtering responses to remove repetitions.

Then, run the script chat_CLI.py in the terminal.

```
(venv) C:\modelo_open_llama_3b-v2>python chat_CLI.py
Using device: cpu
Loading checkpoint shards: 100%|████████████████████████████████████████| 3/3 [00:00<00:00,  8.53it/s]

 Conversation started with Open LLaMA 3B v2 (type '/exit' to exit)

[21:17] User:
```

- Example of Interaction

The following shows an example of interaction in the terminal, where it is possible to observe the customizations implemented in the code.

```
 Conversation started with Open LLaMA 3B v2 (type '/exit' to exit)

[21:22] User: What is the capital of france?
[21:22] LLama3Bv2: Paris, France.

[21:22] User: /clear
```

```
[21:26] User: which is the biggest desert on earth?
[21:26] LLama3Bv2: The largest deserts in terms of area are: The Sahara Desert, located mostly within Africa and covering an estimated 3.5 million squa
re miles (9.076 km2), or about one-third...

[21:26] User: /exit
👋 Ending conversation. Goodbye!

(venv) C:\modelo_open_llama_3b-v2>
```

- Final Remarks

These customizations make the interaction clearer, more organized, and easier to use, contributing to a satisfying and smooth user experience. Some of them were included simply to make the interaction more pleasant and intuitive, even though they are not essential for the functioning of the model.

## 4.2 Exercise 2: Interaction via Telegram Bot

- Description

It is possible to connect the local model to a Telegram bot, allowing interaction directly through the app's chat, whether on a mobile phone, tablet, or PC. The user sends messages to the bot on Telegram, and the model, running on the computer, processes these messages and returns the responses through the same channel. For this interaction to work, it is important that both the computer running the model and the user's device interacting with Telegram are connected to the internet. Although the model's processing occurs locally, an internet connection is essential on both sides, as Telegram is a cloud-based application and depends on this connection to send and receive messages between the user and the bot.

- Solution

Create the Python file with the code to integrate and interact with the model through the Telegram bot, using the script that, in this case, was named chat_telegram.py.

- Setup and Code Used

Open Telegram and use the search bar to look for "BotFather."

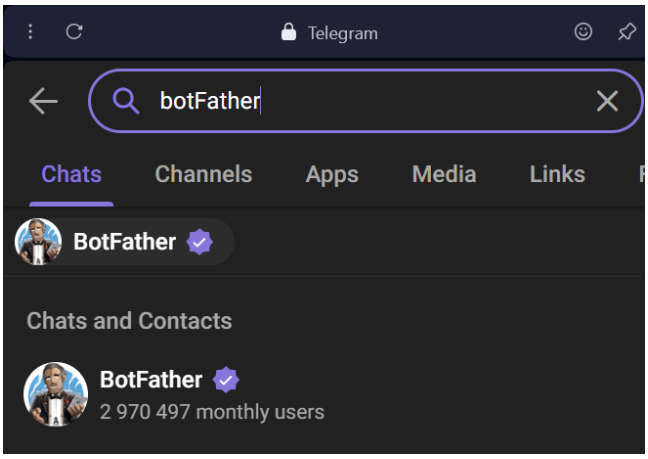



**Figure 2 - Telegram Interface**

In the chat with BotFather, the /newbot command was used to start the creation of a new bot, assigning it the name Llama_3B_V2 and the username Llama_3B_V2_bot (with the required "bot" ending). After these steps, a token was generated that will be used in the Python script to integrate the model with Telegram.

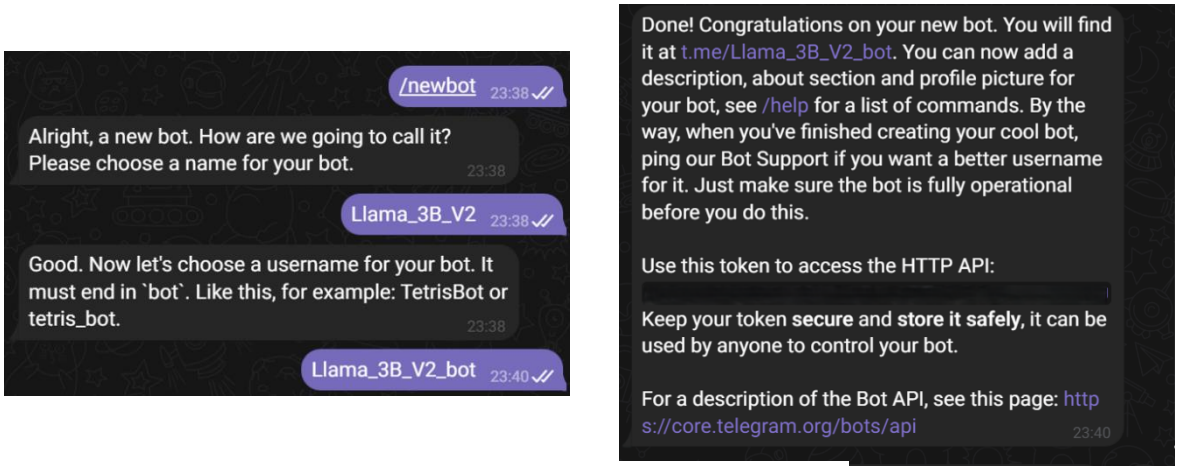



**Figure 3 - Token Acess**

It is necessary to install the required dependencies, such as the python-telegram-bot library, using the pip package manager.

```
pip install python-telegram-bot
```

Then, create the Python file with the code that enables the integration and interaction of the model with the application's bot.

chat_telegram.py > handle_message

```python
import torch
from transformers import LlamaTokenizer, LlamaForCausalLM
from telegram import Update
from telegram.ext import ApplicationBuilder, MessageHandler, CommandHandler, ContextTypes, filters
import logging

# Setup logging
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s', level=logging.INFO)

# Load model and tokenizer
local_directory = "./open_llama_3b_v2_local"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
tokenizer = LlamaTokenizer.from_pretrained(local_directory, local_files_only=True, use_fast=False)

# Corrige pad_token_id se não existir
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

model = LlamaForCausalLM.from_pretrained(local_directory, local_files_only=True).to(device)

# Define start command
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_text("🤖 Hello! I'm ready to chat. Send me a message!")

# Define message handler
```

```python
# Define message handler
async def handle_message(update: Update, context: ContextTypes.DEFAULT_TYPE):
    user_input = update.message.text
    inputs = tokenizer(user_input, return_tensors="pt", padding=True).to(device)
    output_ids = model.generate(
        **inputs,
        max_new_tokens=100,
        pad_token_id=tokenizer.pad_token_id,
        eos_token_id=tokenizer.eos_token_id,
        do_sample=False,
        repetition_penalty=1.2
    )
    response = tokenizer.decode(output_ids[0], skip_special_tokens=True)
    if user_input.lower() in response.lower():
        response = response.split(user_input, 1)[-1].strip()
    response_lines = [line.strip() for line in response.split("\n") if line.strip()]
    if response_lines:
        response = response_lines[0]

    await update.message.reply_text(response)

# Main entry point without asyncio.run()
def main():
    bot_token = '[redacted]'  # <-- Replace with your token
    app = ApplicationBuilder().token(bot_token).build()

    app.add_handler(CommandHandler("start", start))
    app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, handle_message))

    # Directly call run_polling() — it handles asyncio internally
    app.run_polling()

if __name__ == '__main__':
    main()
```

After the implementation of the presented code and once it is running, the model starts communicating directly with the bot on Telegram. This code allows the messages sent by the user to be received, processed by the model, and automatically returned as responses in the Telegram chat.

Next, run the created script in the terminal, in this case, chat_telegram.py.

```
(venv) C:\modelo_open_llama_3b-v2>python chat_telegram.py
Loading checkpoint shards: 100%| 3/3 [00:00<00:00,  6.05it/s]
2025-05-09 17:05:43,812 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot8158098996:AAG5GoicZlIVnwcI2q7YgNu6hpSmTHWoYAU/getMe "HTTP/1.1 200 OK"
2025-05-09 17:05:43,862 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot8158098996:AAG5GoicZlIVnwcI2q7YgNu6hpSmTHWoYAU/deleteWebhook "HTTP/1.1 200 OK"
2025-05-09 17:05:43,864 - telegram.ext.Application - INFO - Application started
2025-05-09 17:05:54,043 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot8158098996:AAG5GoicZlIVnwcI2q7YgNu6hpSmTHWoYAU/getUpdates "HTTP/1.1 200 OK"
```

Then, open the conversation with the bot on Telegram to start the interaction and send the first messages.

- Example of Interaction

The following shows an example of interaction on Telegram, illustrating the exchange of messages between the user and the model through the bot.

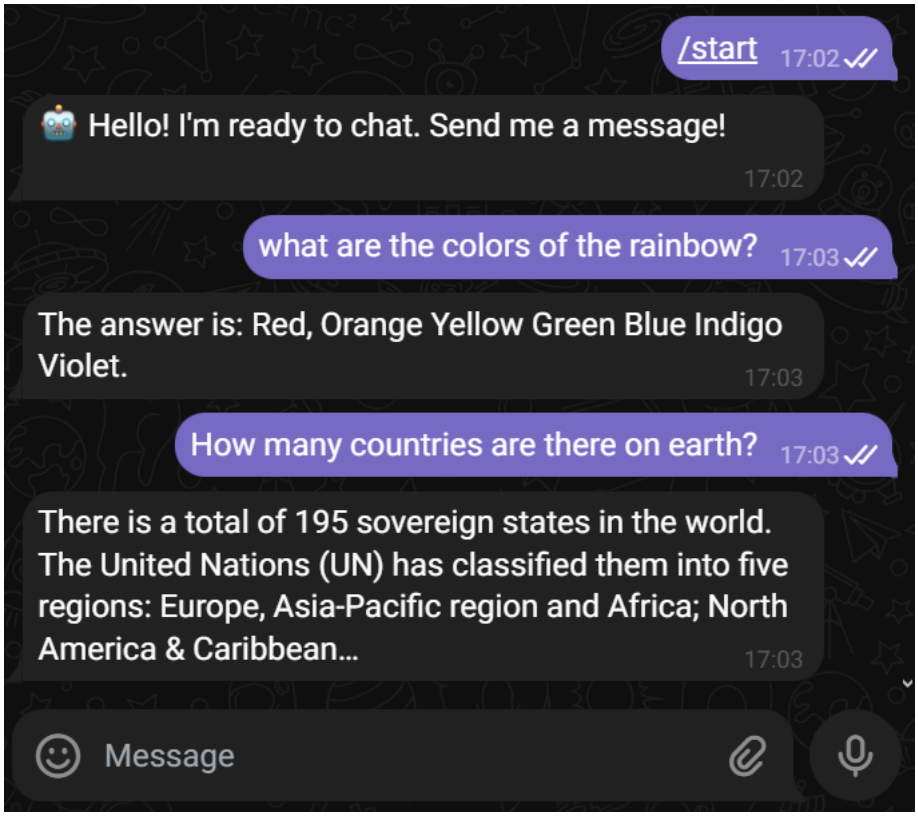



**Figure 4 - Example of Interaction**

- Final Remarks

One of the main advantages of this integration is its portability and ease of use, allowing interaction with the model from different devices with access to Telegram, as long as they have permission to communicate with the bot. A very important point to pay attention to is security, ensuring that the bot token generated by BotFather is kept confidential and not shared publicly to prevent unauthorized access.

## 4.3 Exercise 3: Interaction via Web Interface

- Description

Interaction through a web interface allows the model to be used directly in a browser, making the experience accessible from any device on the same network, such as a PC, tablet, or smartphone. This approach typically uses frameworks like Gradio, FastAPI, Flask, or Streamlit to create the backend and expose the application simply and intuitively for the end user.

- Solution

A Python file was developed using Gradio to build the web interface, using the script that, in this case, was named interface_web.py.

- Setup and Code Used

It is mandatory to install the required dependencies, such as the gradio library, using the pip package manager.

```
pip install gradio
```

To create the web interface, a Python file was prepared using Gradio. The code defines the input and output fields, the connection between the interface and the local model, and the clear, submit, and flag buttons.

web_interface.py > ...

```python
import torch
from transformers import LlamaTokenizer, LlamaForCausalLM
import gradio as gr

local_directory = "./open_llama_3b_v2_local"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
tokenizer = LlamaTokenizer.from_pretrained(local_directory, local_files_only=True, use_fast=False)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

model = LlamaForCausalLM.from_pretrained(local_directory, local_files_only=True).to(device)

def responder_mensagem(user_input):
    inputs = tokenizer(user_input, return_tensors="pt", padding=True).to(device)
    output_ids = model.generate(
        **inputs,
        max_new_tokens=100,
        pad_token_id=tokenizer.pad_token_id,
        eos_token_id=tokenizer.eos_token_id,
        do_sample=False,
        repetition_penalty=1.2
    )
    response = tokenizer.decode(output_ids[0], skip_special_tokens=True)
    if user_input.lower() in response.lower():
        response = response.split(user_input, 1)[-1].strip()
    response_lines = [line.strip() for line in response.split("\n") if line.strip()]
    if response_lines:
        response = response_lines[0]
    return response

gr.Interface(fn=responder_mensagem, inputs="text", outputs="text", title="Open LLaMA 3B-V2 Chatbot").launch()
```

(function) strip: Any

After the implementation of the code, the model becomes accessible through a browser, allowing the user to interact directly with the developed web interface. This solution makes it easy to send questions and view the responses generated by the model in a practical and intuitive way.

After that, run the command in the terminal to execute the script, in this case, interface_web.py.

```
(venv) C:\modelo_open_llama_3b-v2>python web_interface.py
Loading checkpoint shards: 100%|| 3/3 [00:00<00:00,  5.82it/s]
* Running on local URL:  http://127.0.0.1:7860
* To create a public link, set `share=True` in `launch()`.
```

Next, copy the link provided in the terminal and paste it into the browser's address bar to access the web interface.

- Example of Interaction

Here is a screenshot of the browser showing the web interface, with the question field filled in and the response generated by the model.

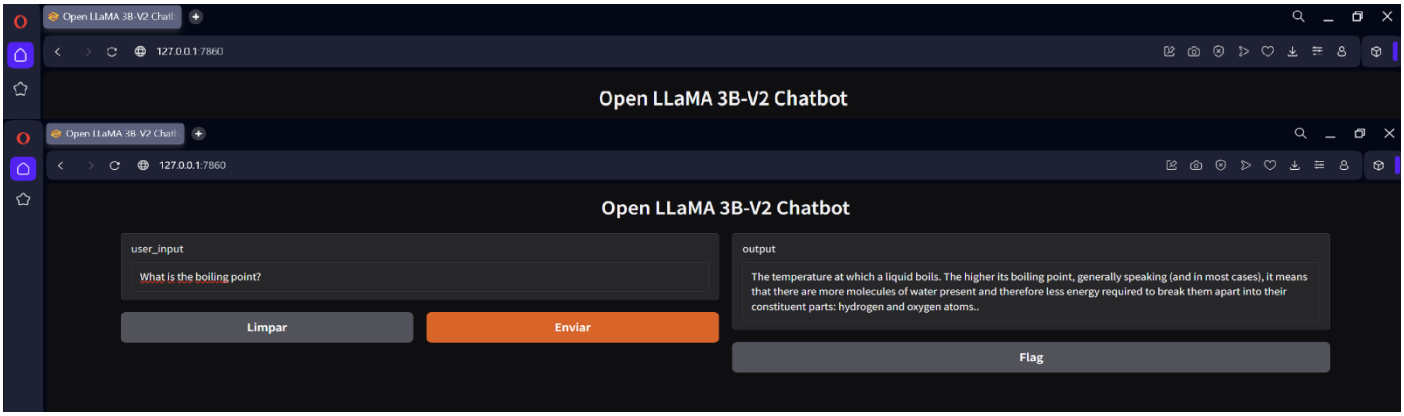



**Figure 5 - Example of Web interaction**

- Final Remarks

The web interface facilitates access to the model, eliminating the need for interaction via terminal or other technical tools. In addition, it offers the possibility of being accessed from different devices on the local network. It also serves as a foundation that can be extended for online deployment, with adaptations to the layout and additional features such as chat history, conversation export, or integration with external APIs.

## 4.4 Expansion and Integration Potential

In addition to the interaction methods addressed in this project — CLI, Telegram, and web interface — there are still other possibilities that significantly expand the potential uses of the model. For example, automation with tools such as n8n, Node-RED, or Zapier allows the model to be integrated into automated workflows, generating responses, classifying information, or performing actions without manual intervention. Extensions or plugins for tools like VS Code, Obsidian, or Jupyter offer an integrated experience directly within development or writing environments, making the use of the model even more seamless. Finally, voice interaction, through systems like Google Assistant, Alexa, or custom solutions with speech recognition, paves the way for innovative and accessible applications.

# 5. LlaMA 3B v2 Model and Tiny LlaMA 1.1B Model

Meta developed the LlaMA 3B v2 model and is part of the second generation of the LlaMA family. It features 3 billion parameters and has been trained on large volumes of text, enabling it to perform tasks such as answering questions, generating text, summarizing, and translating, all with a high level of coherence. This version is robust enough for applications in chatbots, virtual assistants, and NLP tasks with limited resources. It is often used in projects where it is necessary to maintain efficiency and quality without relying on highly powerful hardware.

The Tiny LlaMA 1.1B model is one of the versions of the LlaMA architecture. It features 1.1 billion parameters and is both compact and highly efficient. This version was developed with a focus on environments with limited resources, such as mobile devices or edge applications, while maintaining strong capabilities in natural language understanding and generating. Although its performance is lower than that of larger models, it offers faster response times and reduced memory and processing usage, making it ideal for rapid testing, prototyping, or systems with infrastructure constraints.

## 5.1 Model Comparison

| *Feature* | *LLaMA 3B v2* | *Tiny LLaMA 1.1B* |
|---|---|---|
| **Developer** | Meta | Community (based on LLaMA architecture) |
| **Architecture** | LLaMA v2 | LLaMA-based (Tiny variant) |
| **Parameter Count** | 3 billion | 1.1 billion |
| **Performance** | High coherence, good general purpose NLP | Lower than larger models, but effective for lightweight tasks |
| **Resource Requiren** | Moderate hardware | Very low – optimized for edge/mobile |
| **Ideal Use Cases** | Chatbots, assistants, general NLP | Prototyping, quick tests, mobile/embedded systems |
| **Advantages** | Balanced between quality and efficiency | Fast response, low memory/CPU usage |
| **Limitations** | Heavier than smaller models | Less accurate and powerful than larger models |

**Table 2- Comparison between LlaMA 3B v2 and Tiny LlaMA 1.1B models**

The main difference between the models, as shown in Table 1, lies in performance and efficiency. The LlaMA 3B v2 model is more suitable for tasks that require more elaborate responses due to its higher capacity for text comprehension and generation, which demands more computational resources. On the other hand, the Tiny LlaMA 1.1B model is lighter and faster, making it ideal for cases where memory and processing consumption are limited.

# 6.References


Meta AI. (2024). *LLaMA 3.2 3B – Modelo de linguagem*. Disponível em: https://huggingface.co/meta-llama/Llama-3.2-3B

Meta AI. (2024). *LLaMA 3.2 Connect: Visão para dispositivos móveis e edge*. Meta AI Blog. Disponível em: https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/

Zhang, L. et al. (2024). *LLaMA 3 Technical Report*. arXiv preprint. Disponível em: https://arxiv.org/abs/2401.02385

Asimov Academy. (2023). *Tutorial: Como acessar modelos restritos no Hugging Face*. Disponível em: https://hub.asimov.academy/tutorial/como-acessar-modelos-restritos-no-hugging-face/

Hugging Face. (2024). *Bem-vindo ao Hugging Face Hub*. Disponível em: https://huggingface.co/welcome

Hugging Face. (2024). *Verificação de segurança do utilizador*. Disponível em: https://huggingface.co/security-checkup

Hugging Face. (2024). *Guia rápido do Hugging Face Hub*. Disponível em: https://huggingface.co/docs/huggingface_hub/en/quick-start

Meta AI. (2023). *LLaMA 2 7B – Modelo de linguagem*. Disponível em: https://huggingface.co/meta-llama/Llama-2-7b-hf

Hugging Face. (2024). *Guia sobre modelos com acesso restrito (Gated Models)*. Disponível em: https://huggingface.co/docs/hub/en/models-gated

Hugging Face. (2024). *Guia para uso de modelos privados com Transformers.js*. Disponível em: https://huggingface.co/docs/transformers.js/en/guides/private