ITCS 4011: Introduction to NLP

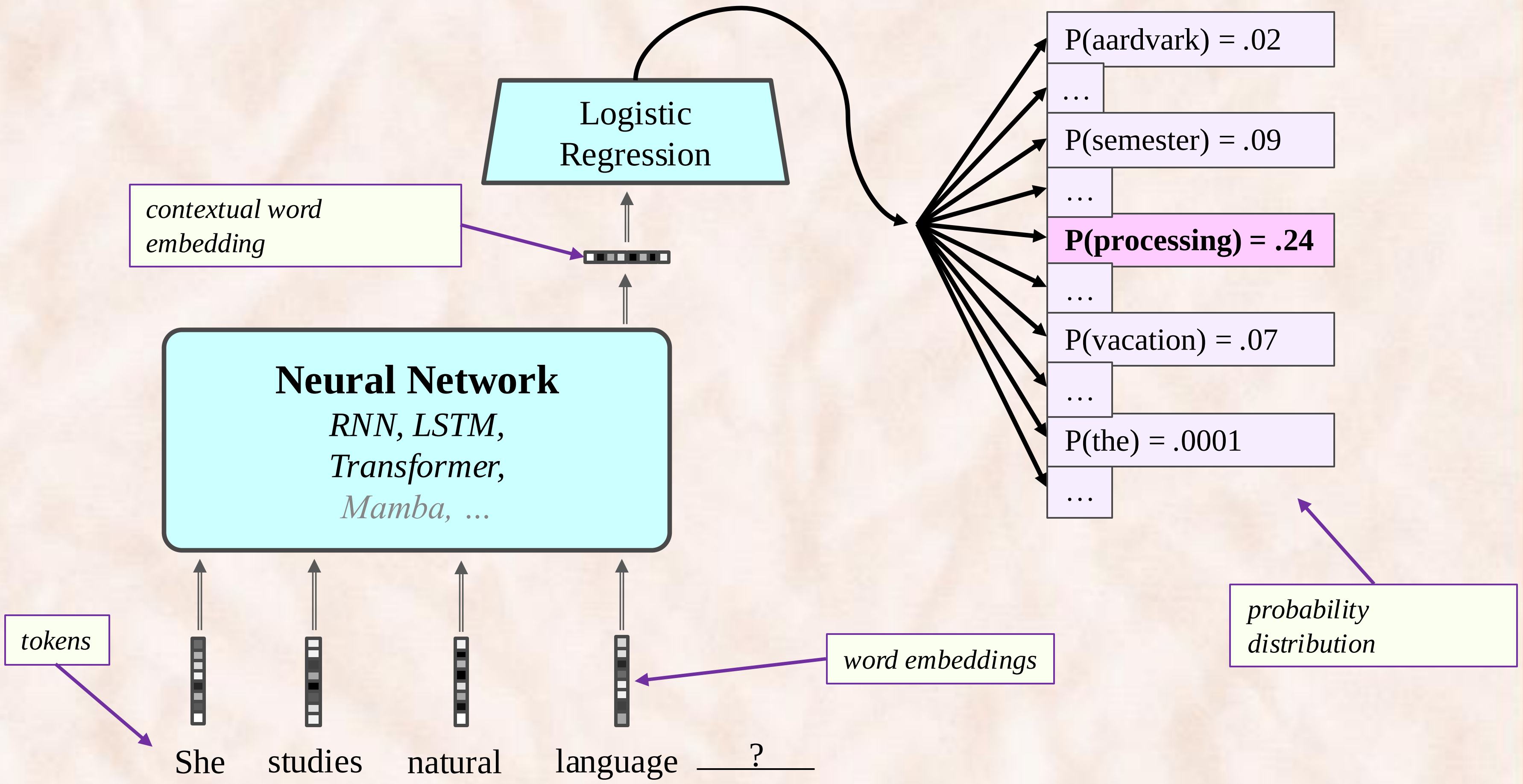
Working with Large Language Models: GPT, Llama, Gemini, …

Razvan C. Bunescu

Department of Computer Science @ CCI

rbunescu@uncc.edu

Large Language Models (LLMs)



LLMs: Training and Fine-tuning

- **Pre-trained** to “understand” natural language and code:
  - Using a language modeling (LM) objective.
- **Fine-tuned** to provide text outputs (answers) in response to their inputs (questions or **prompts**).
  - **Instruction-based fine-tuning**.
    - Collect examples of <*input*, *instruction*> → <*output*> across many tasks, fine-tune on them.
  - **Reinforcement Learning through Human Feedback** (RLHF).
    - Given a prompt, collect sample outputs from LLM and have **human labelers rank them**.
    - Train **reward model** to give higher rewards to top ranked outputs.
    - Optimize an **LLM policy** using PPO with the reward model.

LLMs: Inference and Programming

- "Programming" with GPT, Llama, Gemini, and other LMs:
  - Design a ***prompt***, usually by providing **instructions** and/or some **in-context** examples of how to successfully complete a task:
    - *zero-shot, few-shot in-context learning, CoT, …*
  - Use the prompt with:
    - A simple **chat completion API** (this lecture).
    - More complex multi-agent frameworks (next lecture).
      - **LangChain**.
      - **AutoGen**.

Three Options for Using the Chat API

1. OpenAI's GPT Models (gpt-3.5-turbo and gpt-4):
   - GPT = Generative Pre-trained Transformer
   - Pay per intput and output token, see pricing.
   - Through the ChatGPT browser app.
   - Through the Chat completions API.
     - **Directly through Python** code or through the Playground.

2. Meta's Llama 3 model:
   - Free, quantized 70B version installed by Erfan on an HPC server with A5000 GPUs.
   - API endpoint that can accommodate ~ 40 concurrent requests.
     - Also offering a web app UI.

Three Options for Using the Chat API

3. Google's Gemini model:
   - Free to use Gemini 1.5 Flash.
     - Need personal Google account, rate limits.
   - Through the Google AI Studio.
   - Through the Gemini API.

- Where to find API documentation?

GPT and Llama all use the same chat completion API.

Gemini uses a similar API.

GPT: Setting up the OpenAI API account

- Need to have an OpenAI account:
  - Same account for ChatGPT and the API.
    - Go to https://platform.openai.com, Log in / Signup.
    - **“Continue with Google”, use your UNCC email**.
      - Go to Settings.
      - Go to billing overview, *Set payment* → input credit card, or *Add to credit balance*, input $5.
        - » $5 is more than enough for the work in this class.
      - Go to billing history, *View* → *Download receipt*.

GPT: One-time Setup of API Key

- Create and store a secret API key:
  - Go to Dashboard.
  - Go to API keys and "+ *create new secret key*".
  - Copy the key and store it in a text file named **.env** as follows
    - OPENAI_API_KEY=…
  - Make sure you save the key, it will not be shown again.
- Place or copy the **.env** file in the folder you edit and run the notebook.
  - Other solutions exist, but this is what we will do in this course.
  - Do not put the secret key in your code!

Required Python Modules

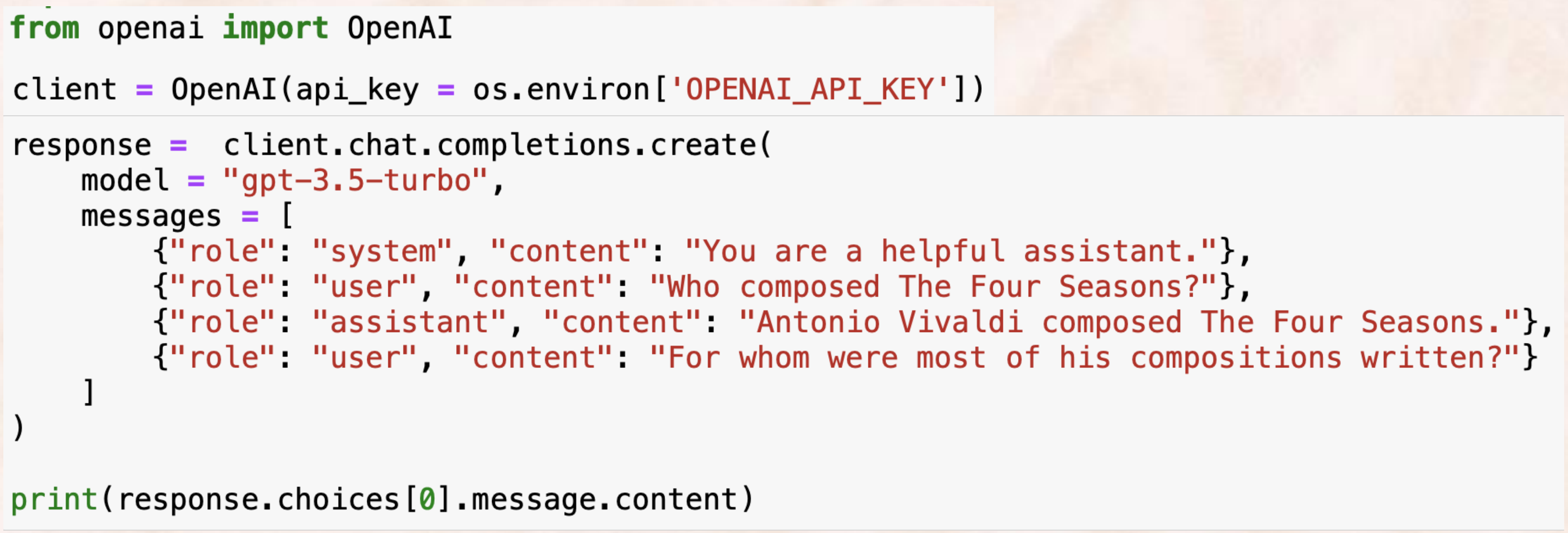
- Install the **openai** module:
  - pip install openai (use pip3)

- Install the **python-dotenv** module, using one of:
  - pip install python-dotenv
  - conda install -c auto python-dotenv

- Alternatively, use Colab instead of Jupyter:
  - Has modules already installed.
  - But ensure the *.env* file is placed in the right Drive folder.

Chat Completion API: openai.ChatCompletion.create

- Chat models take a list of messages as input and return a model-generated message as output.
  - Designed to make multi-turn conversations easy, it’s just as useful for single-turn tasks without any conversation.



```python
from openai import OpenAI

client = OpenAI(api_key = os.environ['OPENAI_API_KEY'])

response =  client.chat.completions.create(
    model = "gpt-3.5-turbo",
    messages = [
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who composed The Four Seasons?"},
        {"role": "assistant", "content": "Antonio Vivaldi composed The Four Seasons."},
        {"role": "user", "content": "For whom were most of his compositions written?"}
    ]
)

print(response.choices[0].message.content)
```

https://platform.openai.com/docs/guides/gpt
https://platform.openai.com/docs/api-reference/chat/create

Chat Completion API: openai.ChatCompletion.create

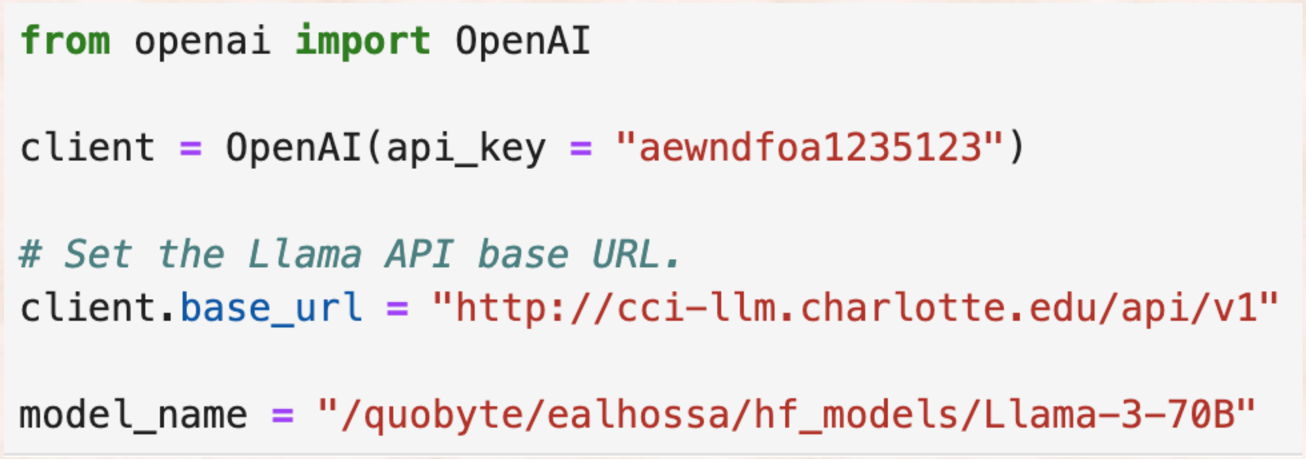
- 3 major roles in the **messages** parameter:
  - **System**: Optional first message, that indicates the LM persona.
    - Also called a *steering promp*, sets up the system behavior.
    - Default is “You are a helpful assistant”.
  - **User**: Provides questions, requests, or comments to the assistant.
  - **Assistant**: Previous responses from the LM assistant, or example of desired LM response.
    - **Need to provide the conversation so far every time** we want to continue with a new user questions.
- Typical input (RE) is **system? user (assistant user)***

Chat Completion API: openai.ChatCompletion.create

- Other useful parameters:
  - **model**: gpt-3.5-turbo or gpt-4 or gpt-4o or gpt-4o-mini.
  - **temperature**: defaults to 1, but set it to **0 for greedy decoding**.
    - We'll see how it is implemented when covering *Logistic Regression*.
  - **top_p**: defaults to 1, use 0.1 if you want the LM to sample tokens only from the top 10% of probability mass, i.e. **nucleus sampling**.
  - **n** : defaults to 1, indicates # completions (alternatives) to generate.
  - **max_tokens**: defaults to ∞, maximum # of tokens to generate.
  - presence_penalty, frequence_penalty, logit_bias: penalize or favor repetitions, or certain tokens (later in this course).

Llama 3: Chat Completion API

- **OpenAI.base_url**:
  - An attribute of the OpenAI class.
- **Model name**:
  - Specifies which version of Llama 3 is being utilized.
  - You must be on Eduroam to access the model directly. Off campus, you need connect through the educational cluster using VPN.

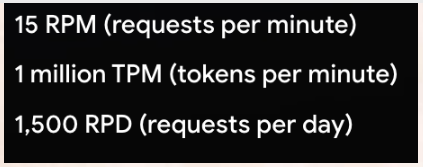


```python
from openai import OpenAI

client = OpenAI(api_key = "aewndfoa1235123")

# Set the Llama API base URL.
client.base_url = "http://cci-llm.charlotte.edu/api/v1"

model_name = "/quobyte/ealhossa/hf_models/Llama-3-70B"
```

Gemini: Setting up the API account

- Need to have a personal Google account:
  - UNCC account will not work (OIT working on it).
  - Go to https://ai.google.dev/, Sign in.
  - Continue with **personal account**.
    - See pricing for available models and pricing.

Gemini 1.5 Flash (free version)

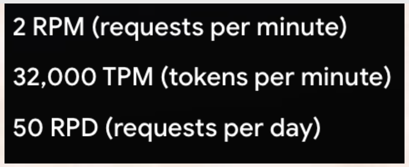


15 RPM (requests per minute)

1 million TPM (tokens per minute)

1,500 RPD (requests per day)

Gemini 1.5 Pro (free version)



2 RPM (requests per minute)

32,000 TPM (tokens per minute)

50 RPD (requests per day)

- Follow the Gemini API quickstart (next slides).

Gemini: One-time Setup of API Key

- Create and store a secret API key:
  - Get an API key from Google AI Studio.
    - Click Create API Key.
  - Copy the key and store it in a text file named **.env** as follows
    - GEMINI_API_KEY=…

- Place or copy the **.env** file in the folder you edit and run the notebook.
  - Other solutions exist, but this is what we will do in this course.
  - Do not put the secret key in your code!

Required Python Modules

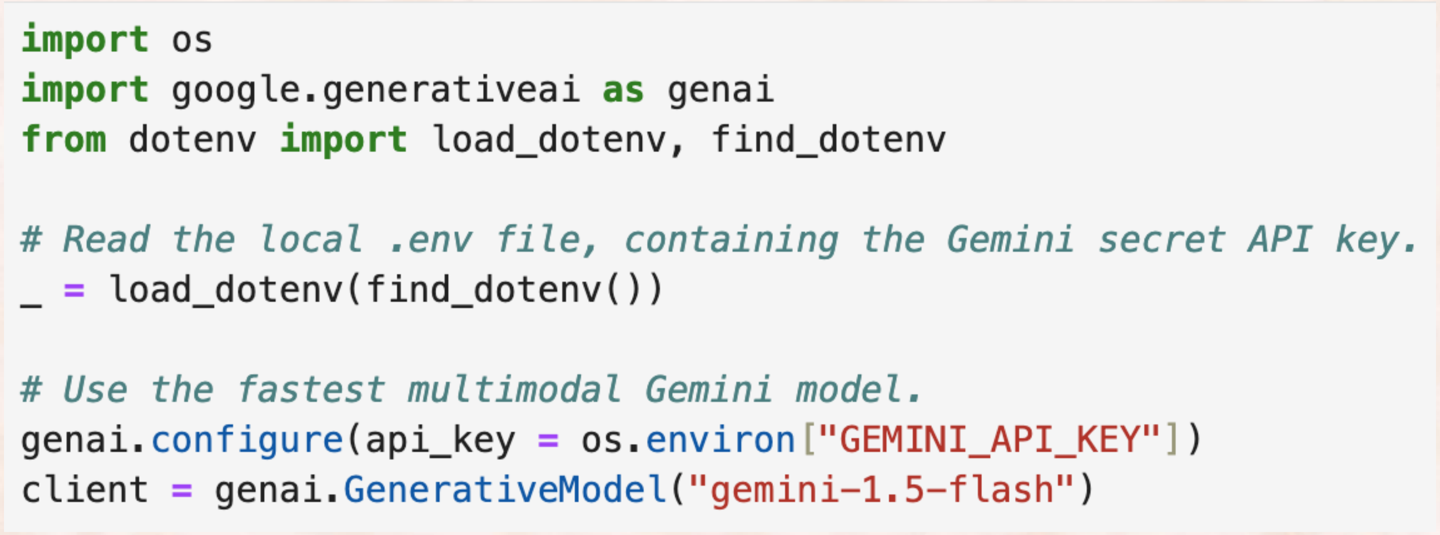
- Install the **google-generativeai** module:
  - pip install -U google-generativeai (use pip3)
    - Make sure you have latest version of pip3 and setuptools:
      - pip3 install --upgrade pip
      - python3 -m pip install --upgrade setuptools

- Install the **python-dotenv** module, using one of:
  - pip install python-dotenv
  - conda install -c auto python-dotenv

- Alternatively, use Colab instead of Jupyter:
  - Has modules already installed.
  - But ensure the *.env* file is placed in the right Drive folder.

Gemini: Google AI Studio

- First, setup to use faster version of most recent model:
  - More details here.



```python
import os
import google.generativeai as genai
from dotenv import load_dotenv, find_dotenv

# Read the local .env file, containing the Gemini secret API key.
_ = load_dotenv(find_dotenv())

# Use the fastest multimodal Gemini model.
genai.configure(api_key = os.environ["GEMINI_API_KEY"])
client = genai.GenerativeModel("gemini-1.5-flash")
```

Gemini: Google AI Studio

- There are only two roles: *user* and *model*.
  - More details at the Text Generation API documentation.

```
query = "Justin sits next to Razvan. One of them is happy and one of them is grumpy. " \
                "The person sitting next to Justin is grumpy. Who is happy?"

# Gemini supports two roles: 'user' and 'model'.
conversation = [
    {"role": "user", "parts": [query]}
]

# Send a chat completion request.
gConfig = genai.types.GenerationConfig(max_output_tokens = 300,
                                       temperature = 0)
response = client.generate_content(contents = conversation,
                                   generation_config = gConfig)

print(f"API raw response:\n{response}\n")
```

Supplementary Activities

- Take the Building Systems with the ChatGPT API short course (1 hour) from deeplearning.ai.

- Go through the examples in the Jupyter notebooks.