Generative AI

W4 Agenda

- **On Premise LLMs**
    - **Why local LLMs are important?**
    - **How to launch LLM locally?**

On Premise LLMs

Why local LLMs are important?

Local LLMs – Pros and Cons

👍

- **Full data privacy**
- Can work offline
- Ability to use underutilzed infrastructure e.g. Priv GPUs

👎

- Usually limited to smaller models, with 3-20B params on normal GPUs
- Much slower than SaaS solutions
- Needs very high utilization and optimization to be actually cheaper than small SaaS LLMs

LLMs you can run locally on a private GPU 12-16 GB (with some quantization)

- Gemma 3 12B
- LLaMA 13B
- Falcon 7B
- Bielik-11B-v2

What is Quantization??

Definition
Converting model weights from higher-precision (e.g. FP16 or FP32) down to fewer bits (e.g. 8-bit, 4-bit) to reduce memory usage and computational overhead

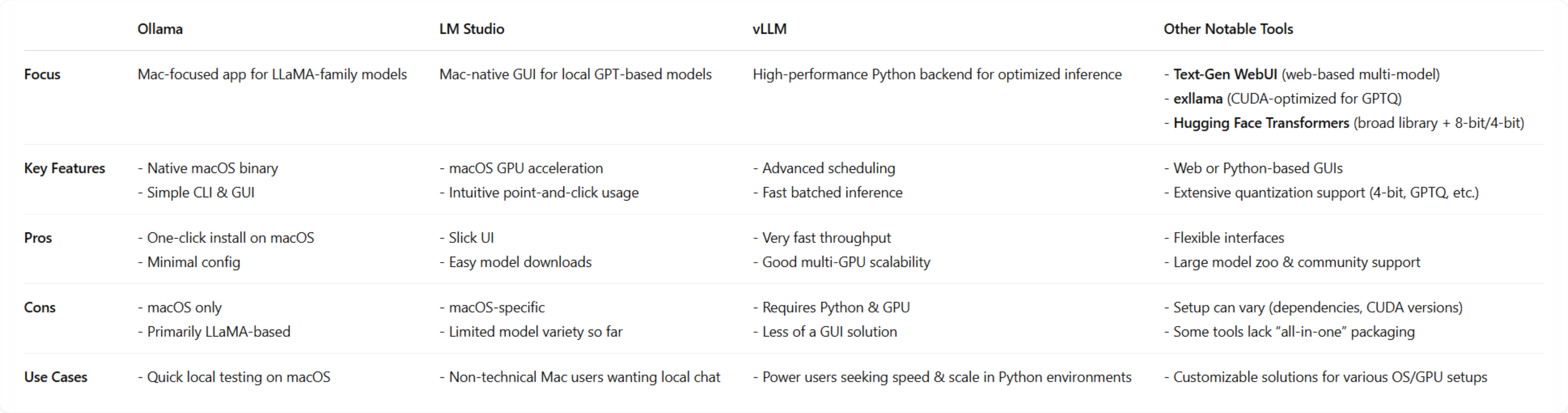
Why It Matters
- **Lower VRAM Requirements**: Fits larger models on smaller GPUs.
- **Faster Inference**: Fewer bits to process speeds up forward passes.
- **Energy Efficiency**: Reduced compute demands save power.

Approaches
- **Uniform Quantization**: Single scale factor per block of weights.
- **Advanced “K-Quant” Methods**: Multiple scales or per-channel scaling for better accuracy at the same bit depth.

Trade-Offs
- **Size vs. Accuracy**: Fewer bits can degrade model performance.
- **Implementation Complexity**: Complex quantization methods often require specialized tooling.

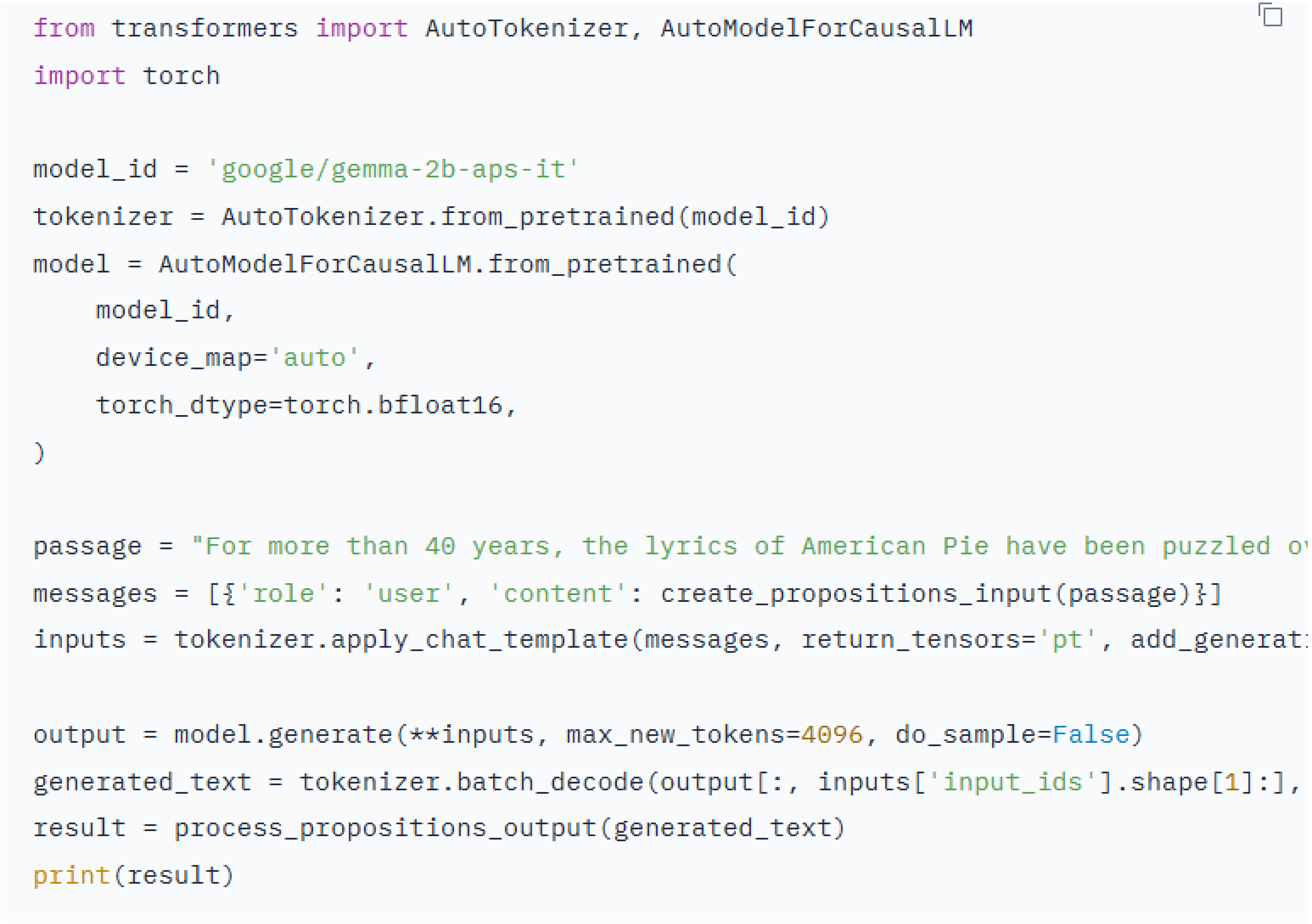
Deploying LLMs locally

How can you run LMMs locally?



	Ollama	LM Studio	vLLM	Other Notable Tools
Focus	Mac-focused app for LLaMA-family models	Mac-native GUI for local GPT-based models	High-performance Python backend for optimized inference	- **Text-Gen WebUI** (web-based multi-model) - **exllama** (CUDA-optimized for GPTQ) - **Hugging Face Transformers** (broad library + 8-bit/4-bit)
Key Features	- Native macOS binary - Simple CLI & GUI	- macOS GPU acceleration - Intuitive point-and-click usage	- Advanced scheduling - Fast batched inference	- Web or Python-based GUIs - Extensive quantization support (4-bit, GPTQ, etc.)
Pros	- One-click install on macOS - Minimal config	- Slick UI - Easy model downloads	- Very fast throughput - Good multi-GPU scalability	- Flexible interfaces - Large model zoo & community support
Cons	- macOS only - Primarily LLaMA-based	- macOS-specific - Limited model variety so far	- Requires Python & GPU - Less of a GUI solution	- Setup can vary (dependencies, CUDA versions) - Some tools lack “all-in-one” packaging
Use Cases	- Quick local testing on macOS	- Non-technical Mac users wanting local chat	- Power users seeking speed & scale in Python environments	- Customizable solutions for various OS/GPU setups

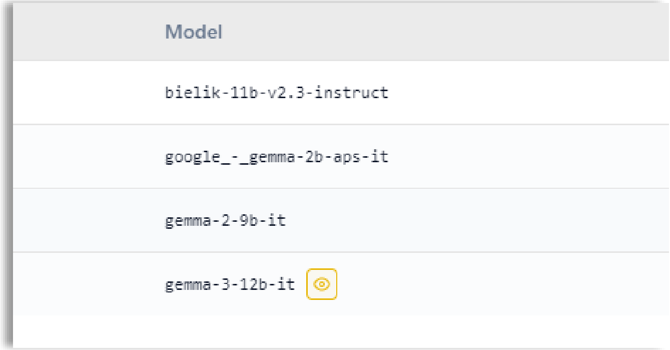
Hugging face transformers example



```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

model_id = 'google/gemma-2b-aps-it'
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    device_map='auto',
    torch_dtype=torch.bfloat16,
)

passage = "For more than 40 years, the lyrics of American Pie have been puzzled o
messages = [{'role': 'user', 'content': create_propositions_input(passage)}]
inputs = tokenizer.apply_chat_template(messages, return_tensors='pt', add_generat

output = model.generate(**inputs, max_new_tokens=4096, do_sample=False)
generated_text = tokenizer.batch_decode(output[:, inputs['input_ids'].shape[1]:],
result = process_propositions_output(generated_text)
print(result)
```

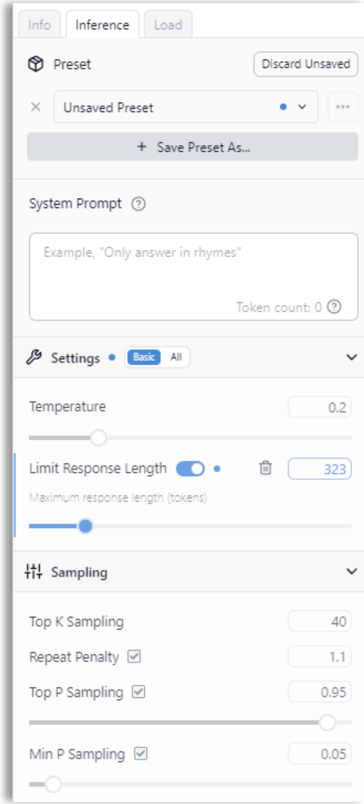
LM Studio example

Choose model



Model
bielik-11b-v2.3-instruct
google_-_gemma-2b-aps-it
gemma-2-9b-it
gemma-3-12b-it

Setup hyperparams



Unsaved Preset

System Prompt

Example, "Only answer in rhymes"

Token count: 0

Temperature 0.2

Limit Response Length 323

Maximum response length (tokens)

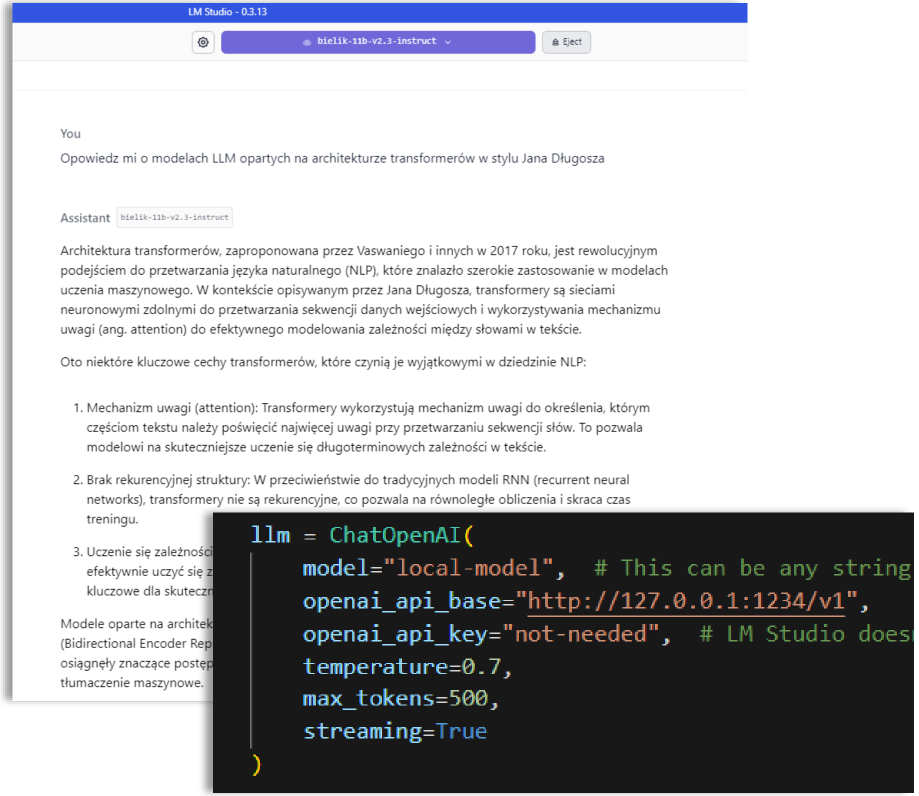
Top K Sampling 40

Repeat Penalty ☑ 1.1

Top P Sampling ☑ 0.95

Min P Sampling ☑ 0.05

Chat using API or UI



You

Opowiedz mi o modelach LLM opartych na architekturze transformerów w stylu Jana Długosza

Assistant `bielik-11b-v2.3-instruct`

Architektura transformerów, zaproponowana przez Vaswaniego i innych w 2017 roku, jest rewolucyjnym podejściem do przetwarzania języka naturalnego (NLP), które znalazło szerokie zastosowanie w modelach uczenia maszynowego. W kontekście opisywanym przez Jana Długosza, transformery są sieciami neuronowymi zdolnymi do przetwarzania sekwencji danych wejściowych i wykorzystywania mechanizmu uwagi (ang. attention) do efektywnego modelowania zależności między słowami w tekście.

Oto niektóre kluczowe cechy transformerów, które czynią je wyjątkowymi w dziedzinie NLP:

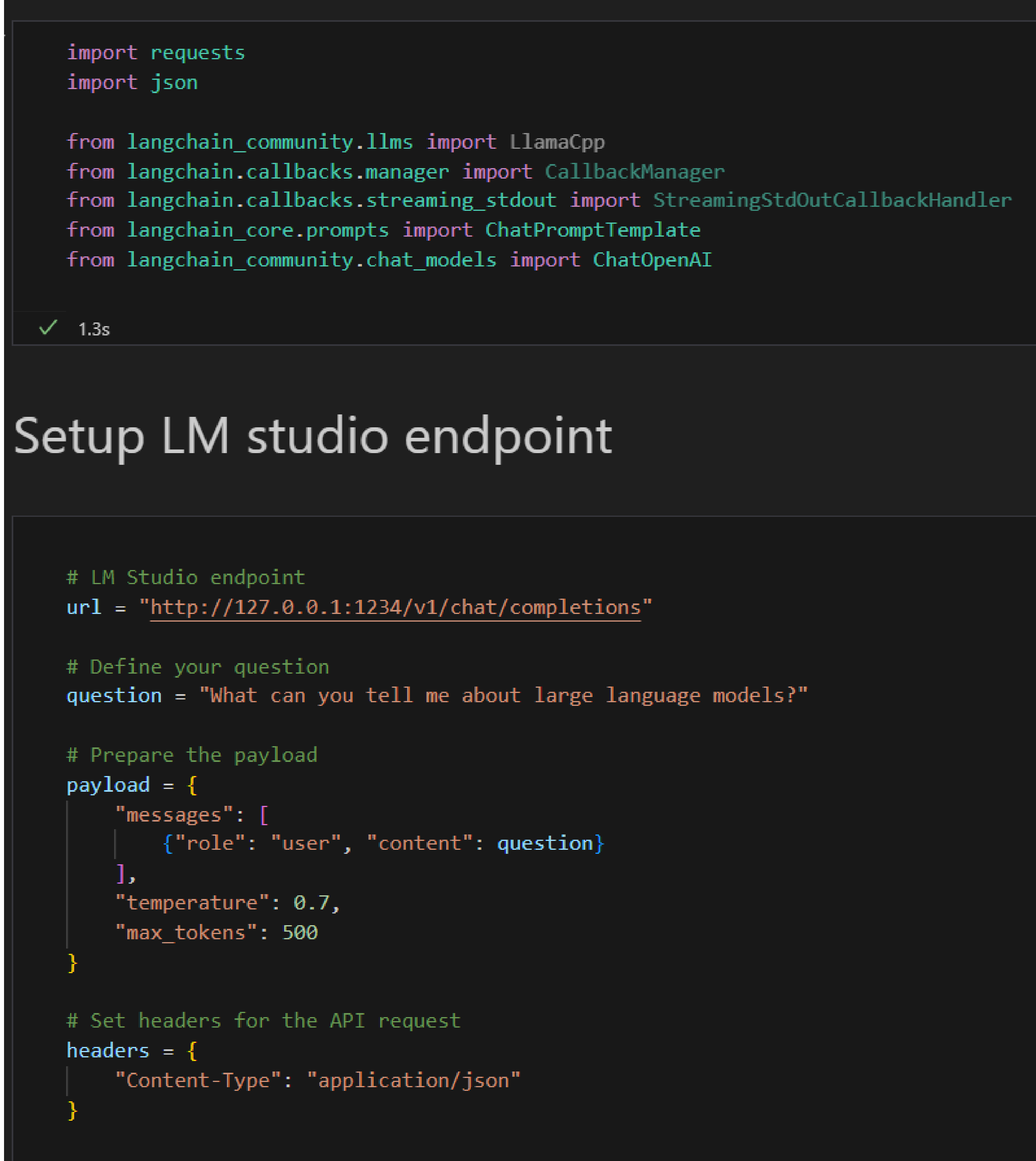
1. Mechanizm uwagi (attention): Transformery wykorzystują mechanizm uwagi do określenia, którym częściom tekstu należy poświęcić najwięcej uwagi przy przetwarzaniu sekwencji słów. To pozwala modelowi na skuteczniejsze uczenie się długoterminowych zależności w tekście.
2. Brak rekurencyjnej struktury: W przeciwieństwie do tradycyjnych modeli RNN (recurrent neural networks), transformery nie są rekurencyjne, co pozwala na równoległe obliczenia i skraca czas treningu.
3. Uczenie się zależności
efektywnie uczyć się z
kluczowe dla skuteczn

Modele oparte na architek
(Bidirectional Encoder Rep
osiągnęły znaczące postęp
tłumaczenie maszynowe.

```
llm = ChatOpenAI(
    model="local-model",  # This can be any string
    openai_api_base="http://127.0.0.1:1234/v1",
    openai_api_key="not-needed",  # LM Studio does
    temperature=0.7,
    max_tokens=500,
    streaming=True
)
```

LM Studio exercise

Start with Bonus-on-premise-LLMs.py



```
import requests
import json

from langchain_community.llms import LlamaCpp
from langchain.callbacks.manager import CallbackManager
from langchain.callbacks.streaming_stdout import StreamingStdOutCallbackHandler
from langchain_core.prompts import ChatPromptTemplate
from langchain_community.chat_models import ChatOpenAI
```

✓ 1.3s

Setup LM studio endpoint

```
# LM Studio endpoint
url = "http://127.0.0.1:1234/v1/chat/completions"

# Define your question
question = "What can you tell me about large language models?"

# Prepare the payload
payload = {
    "messages": [
        {"role": "user", "content": question}
    ],
    "temperature": 0.7,
    "max_tokens": 500
}

# Set headers for the API request
headers = {
    "Content-Type": "application/json"
}
```

GenAI Voice Solutions

What are Voice AI Solutions?

- Voice AI enables natural spoken conversations between humans and AI systems
- Combines Speech Recognition (STT), Language Models (LLM), and Speech Synthesis (TTS)
- Goal: Create human-like conversational experiences without typing

Key Components:

- Speech-to-Text (STT): Converts spoken audio into text (e.g., Whisper, Deepgram)
- Large Language Model (LLM): Processes text, generates intelligent responses
- Text-to-Speech (TTS): Converts text responses into natural-sounding audio

Voice AI Use Cases

Real-world applications of conversational voice AI

Customer Service & Support

- AI phone agents for 24/7 customer support (e.g., banks, airlines, telecom)
- Appointment scheduling and reservation systems

Healthcare

- Patient intake and symptom screening via voice
- Medication reminders and health monitoring assistants

Smart Assistants & IoT

- Home automation voice control (Alexa, Google Home, Siri)
- In-car voice assistants for navigation and entertainment

Enterprise & Productivity

- Voice-driven meeting transcription and note-taking
- Hands-free data entry and workflow automation

Voice AI Architectures

Two main approaches to building voice AI systems

3-Phase Pipeline (STT → LLM → TTS)

- Audio → Speech-to-Text → LLM processes text → Text-to-Speech → Audio
- Each component is separate and can be optimized independently
- Total latency = STT latency + LLM latency + TTS latency (typically 1-3+ seconds)

Realtime Voice-to-Voice (Speech-to-Speech)

- Single model processes audio directly without intermediate text conversion
- Examples: GPT-4o Realtime API, Gemini 2.0 Live, Moshi by Kyutai
- Native understanding of tone, emotion, and paralinguistic cues
- Latency as low as 200-500ms for near-instant responses

3-Phase vs Realtime: Pros and Cons

3-Phase Pipeline (STT-LLM-TTS)

- Flexible: swap components easily
- Mature ecosystem with many providers
- Better text accuracy for complex content
- Easier to debug and monitor each step
- Works with any LLM (GPT-4, Claude, etc.)
- Higher latency (1-3+ seconds total)
- Loses audio context (tone, emotion)
- Cannot handle overlapping speech well

Realtime Voice-to-Voice

- Ultra-low latency (~200-500ms)
- Natural interruption handling
- Preserves emotional/tonal context
- More human-like conversations
- Better for real-time interactions
- Limited model options currently
- Higher compute costs
- Less control over individual steps

Handling Streaming & Interruptions

Critical techniques for responsive voice AI

Streaming for Low Latency

- Stream STT results incrementally (partial transcripts)
- Use LLM streaming to start TTS before full response is generated
- Chunk TTS audio and play as it's generated (don't wait for full audio)
- Result: User hears response while it's still being generated

Handling Interruptions (Barge-in)

- Voice Activity Detection (VAD): Detect when user starts speaking
- Immediately stop current TTS playback when interruption detected
- Cancel pending LLM generation to save resources
- Process new user input and restart the pipeline

Best Practices

- Use WebSockets or gRPC for real-time bidirectional communication
- Implement echo cancellation to avoid AI hearing its own voice
- Buffer management: balance between latency and audio quality