## Working with Large Language Models

University of Hamburg - Spring 2025

Dr. Tobias Vlćek

## Welcome to today's workshop!

- Explore Large Language Models (LLMs).
- You've likely interacted with LLMs already:
  - ChatGPT
    - Google's Gemini
    - Claude
- By the end of this workshop, you'll understand:
  - How LLMs work at a conceptual level
  - The practical applications of these powerful tools
  - How to effectively use them in your own work

### What is an LLM?

- LLMs are sophisticated Artificial Intelligence (AI) systems.
- They represent a breakthrough in Natural Language Processing (NLP).
- These models are designed to:
  - Comprehend human language in its nuance and complexity
  - Generate coherent and contextually appropriate text
  - Engage in natural conversation and respond to queries
  - Perform language tasks with human-like understanding
- They achieve this by training on vast datasets of text from diverse sources.
- The underlying technology uses advanced statistical methods to identify patterns and predict language.

#### 🅊 Tip

Why should you care? LLMs are rapidly transforming how we:

- · Write and edit content
- Develop software
- Conduct research
- Analyze data
- Automate tasks
- Learn new concepts

Understanding these tools is becoming essential across nearly every professional field.

## A Great Overview by 3Blue1Brown

- Greg Sanderson provides an excellent, concise explanation of LLMs.
- Great starting point to understand LLMs.
- Check out his YouTube channel, 3Blue1Brown for more.

https://www.youtube.com/embed/LPZh9BOjkQs

## **Today's Key Concepts**

- Fundamental principles behind how LLMs work.
- The process of training these powerful models.
- · Limitations and ethical considerations of LLMs.
- Practical applications, including pair-programming.
- How to use LLMs with your own data (Retrieval Augmented Generation).

• Further resources and tools.

## How they work

- · High-level overview of LLM workings.
- · No deep technical details.

### **Deep Learning and Text Data**

- LLMs are deep learning models.
- · Built using artificial neural networks with many layers ("deep").
- Trained on enormous amounts of text data:
  - Internet
  - Books
  - Articles
  - Code
  - And more!
- Training allows them to learn statistical patterns of language.
- Become incredibly good at predicting word sequences.
- Foundation for tasks like:
  - Text Generation: Creating new text similar to training data.
  - **Translation:** Converting text between languages.
  - Summarization: Condensing text into shorter summaries.
  - Question Answering: Answering questions based on learned information.
  - Sentiment Analysis: Determining emotional tone of text.

## **The Transformer**

- Modern LLMs are powered by the revolutionary transformer architecture.
- Introduced in the groundbreaking paper "Attention is All You Need" (Vaswani et al., 2017).
- · This innovation fundamentally changed the landscape of NLP.

#### How Transformers Work (Simplified):

- 1. Tokenization: Text is segmented into meaningful units called *tokens*.
- 2. **Relationship Mapping:** The model creates a mathematical representation of how tokens relate to each other.
  - This allows understanding of complex connections between words and phrases in context.
- 3. Attention Mechanism: The critical innovation that makes transformers so powerful.
  - Enables the model to focus selectively on relevant parts of the input.
    - Weighs the importance of different words based on their relevance to the current task.

#### **Example: The Power of Attention**

- · Consider the sentence: "The cat sat on the mat because it was warm."
- · The attention mechanism helps the model understand that:

- "it" refers to "the mat," not "the cat."
- This is determined by analyzing the contextual relationships between all words in the sentence.

## **Tokens: The Building Blocks**

- LLMs process text as tokens, not whole words.
- Token can be:
  - Whole word (e.g., "cat")
  - Subword (e.g., "un-", "break", "-able" for "unbreakable")
  - Individual characters

#### Why use tokens?

- · Handles Unknown Words: Processes unseen words via subword units.
- Efficiency: Learning relationships between tokens is more efficient.

#### **Challenges with Tokenization:**

- · Tokenization can be tricky.
- Especially for languages without spaces (Chinese, Japanese).
- · Current methods better for languages with similar scripts (English, Spanish, French).
- Potentially disadvantages other languages.
- Area of ongoing research.

#### i Note

Think about this: How might this impact the fairness and inclusivity of LLMs?

## Context Window: LLM's "Memory"

- · Context window: LLM's short-term memory.
- Number of tokens the model can process at once.
- Larger context window:
  - "Remembers" more of conversation/document.
  - More coherent and contextually relevant responses.
  - More computational resources needed.
  - Potentially less coherence if too long.
- Shorter context window:
  - Faster processing.
  - May lose track of earlier conversation parts.

#### 🔮 Tip

There is a trade-off between the context window and the size of the model. A larger model can usually handle a larger context window, but it will also take more computational resources to run.

## **LLM Architectures**

- · LLMs are versatile tools.
- · Different tasks need different architectures.
- · Main architectures designed for specific strengths.

## **Encoder-Decoder**

- How it works:
  - Encoder: Processes input text -> condensed representation ("thought vector").
  - Decoder: Takes representation -> generates output text.
  - Two-step process: understand, then create.
- Analogy: Translation.
  - Understand English sentence (encoder).
  - Construct German sentence (decoder).
- · Key Feature: Excellent for tasks with different input and output sequences.

## **Encoder-Only**

- How it works:
  - Focuses on understanding input.
  - Processes input -> rich representation of meaning.
- · Analogy: Detective analyzing crime scene.
  - Gather clues (input) -> build complete picture.
  - No long report needed (output).
- Key Feature: Great for classification or information extraction.

## **Decoder-Only**

- How it works:
  - Focuses on generating text.
  - Takes prompt/conversation history -> predicts next word, then next, etc.
  - Builds output sequence.
- · Analogy: Creative writer or conversation.
  - Writer starts with sentence and continues story.
  - Conversation builds upon previous turns.
- Key Feature: Ideal for original text creation.

#### Mixture of Experts (MoE)

• How it works:

- Uses multiple smaller "expert" models.
- Each expert specialized in different aspect.
- "Gating network" decides which expert(s) to use.
- Analogy: Large company with departments.
  - Manager (gating network) assigns project to relevant departments.
- Key Feature: Efficient scaling to large models/datasets.
  - Only subset of experts activated per input.
    Improves performance and reduces cost.

  - Used in large chatbot models.

# **Training LLMs**

• Training an LLM: like teaching a child language, but huge scale.

## **The Training Process**

- · LLMs learn via "pre-training" on massive text/code datasets.
- 1. Input Processing:
  - · Model receives token sequence.
  - Text broken into tokens (words, subwords, characters).
- 2. Prediction Task:
  - · Given sequence, predict next token.
  - Like filling in the blank: "The cat sat on the \_\_\_\_"
- 3. Learning Loop:
  - Make prediction.
  - Compare to actual answer.
  - Adjust internal parameters.
  - Repeat billions of times.

#### i Note

This process requires enormous computational resources - training a large model can cost millions of dollars in computing power!

## **Computational Power and Parallelism**

- Training needs immense computational power.
- · Specialized hardware: GPUs (Graphics Processing Units).
- · Model Parallelism: Distributes model parts across multiple GPUs.
  - Speeds up training via parallel processing.
  - Reason for Nvidia's stock increase.

## **Training Phases**

- · Training process often divided into phases:
- Self-Supervised Learning:
  - Trained on massive dataset without explicit labels.
  - Learns to predict next token.
  - Learns basic language rules and patterns.

- Like learning grammar/vocab by reading books.

#### • Supervised Fine-tuning:

- Trained on smaller, *labeled* dataset for specific tasks.
- Specialized for tasks like question answering, summarization.
- Like specialized course after learning basics.

#### • Reinforcement Learning from Human Feedback (RLHF):

- Human feedback refines model output.
- Makes output more helpful, honest, and harmless.
- Aligns text generation with human preferences.
- Like teacher feedback to improve writing.

## **Limitations and Ethical Considerations**

- · LLMs are powerful but not perfect.
- · Understand limitations and ethical implications.

## **Bias: A Reflection of the Data**

- LLMs can exhibit biases (gender, racial, cultural, etc.).
- · Learn biases from training data.
- Biased training data -> biased output.

#### Example:

• Trained mostly on text by men -> reflect male perspectives/stereotypes.

#### Impact of Bias:

- · Unfair/discriminatory outcomes.
- · Reinforcement of harmful stereotypes.
- Erosion of trust in Al.

#### **Mitigating Bias:**

- Data Curation: Carefully select diverse training data. Include data from different demographics, languages, cultures.
- Model Fine-tuning: Train on datasets to counteract bias.
- · Auditing and Evaluation: Rigorously test models for bias.

## **Adversarial Attacks: Tricking the Model**

- · Adversarial attacks: small input changes to mislead LLM.
- Exploit model vulnerabilities.

#### Example:

• Slightly change prompt wording -> biased/harmful/nonsensical response.

#### Why is this a problem?

- · Manipulate LLMs for malicious purposes.
- · Spreading misinformation or generating harmful content.

## **A Broader View**

- LLMs raise broader ethical concerns:
- Job Displacement: Automate human tasks -> potential job losses.

- How to prepare for this shift?

- Misinformation: Generate realistic fake news easily.
  - How to combat misinformation spread?
- Malicious Use: Deepfakes, harmful content, propaganda.
  - What safeguards are needed?
- Privacy: Trained on personal data.
  - Data usage and privacy regulations needed?
- Accountability: Unclear who is responsible for LLM-generated content.
   How to hold producers/users accountable?
- Transparency: Difficult to understand LLM decision-making.
  - How to make LLMs more transparent?
- Addressing ethical concerns requires: researchers, policymakers, public.

## **Putting LLMs to Work**

· Explore how to use LLMs practically.

## Pair Programming with LLMs

- Pair programming: two programmers work together.
  - One writes code, other reviews/feedback.
- · LLMs as "AI pair programmer":
  - Code Completion: Suggest code snippets, complete lines.
  - Error Detection: Identify bugs, suggest fixes.
  - Code Generation: Generate functions/blocks from instructions.
  - Code Explanation: Explain how code works.
  - Best Practices: Suggest improvements and best practices.

#### **Tools for Pair Programming:**

- GitHub Copilot: VS Code integration, AI-powered suggestions.
- Cursor: VS Code fork, more powerful AI features.
- Zed: Code editor for pair programming (LLMs and humans). Local and API models.

## **Tips for Effective Pair Programming**

- Start with Simple Tasks: Begin with code completion and gradually move to more complex tasks.
- Be Precise in Your Instructions: The more specific your prompt, the better the LLM's response.
- Always Review Generated Code: LLMs can make mistakes or suggest inefficient solutions.
- Learn as You Collaborate: Pay attention to new techniques and patterns the LLM introduces.
- Iterate and Refine: Use the LLM's output as a starting point, then improve through conversation.

#### 💡 Tip

**Manage Context Effectively:** Remember that LLMs have a finite context window. For large codebases, focus on specific components rather than the entire project at once. Break down complex problems into manageable chunks.

## **Retrieval Augmented Generation (RAG)**

- Retrieval Augmented Generation (RAG): combines LLMs with external knowledge.
- · LLM with vast library and research assistant.

#### Why is RAG important?

- Up-to-Date Information: Access current information beyond training data.
- Reduced Hallucinations: Responses grounded in facts, less making things up.
- Domain-Specific Knowledge: Connect to your documents, databases, APIs.
- Expert in your area: Tailored to specific domain without retraining.

#### How RAG Works:

- 1. Query: Ask a question or provide prompt.
- 2. Retrieval: Search external sources for relevant info.
- 3. Augmentation: Add retrieved info to LLM's prompt (context).
- 4. Generation: LLM generates response based on:
  - Pre-trained knowledge.
  - · Retrieved information.

#### Example:

- · Question: "What are latest advancements in quantum computing?"
- · Retrieval: Search scientific publications, news, research databases.
- Augmentation: Key findings added to prompt.
- Generation: Comprehensive, up-to-date answer.

## **Running LLMs Locally with Ollama**

• Running LLMs locally: on your own computer.

#### Advantages:

- · Privacy: Data stays on your machine.
- · Cost Savings: No API fees.
- · Customization: Experiment, fine-tune for needs.
- · Offline Access: Use without internet.
- · Requires understanding hardware and technical setup.
- · Ollama: user-friendly tool for local LLMs.

### Ollama

- Ollama: free, open-source.
- · Simplifies downloading, installing, running LLMs (macOS, Linux, Windows).
- · Command-line interface (CLI), wide model support.

#### Key Features of Ollama:

- · Easy Installation: Simple download and install.
- Model Management: Download/manage models from library (Hugging Face).
- · Command-Line Interface: Interact via commands.
- API Server: Local API server, integrate in applications.

## **Hardware Requirements**

• Local LLMs need sufficient resources:

Component	Requirement	Notes
RAM	Minimum 8GB	- 16GB recommended- 32GB+ for larger models
CPU	Modern multi-core	- Can run smaller models (< 7B params)- Faster = better performance
GPU	VRAM dependent	- More VRAM = larger models- NVIDIA GPUs preferred
Storage	10GB+ free	- Models can be several GB each

#### 🛕 Warning

Performance Note: CPU-only can be much slower compared to GPU.

## **Billions of Parameters (B)**

- · LLMs described by parameters (internal learned values).
- More parameters -> more capable (generally), larger model, more resources.
- Model size not everything, quality depends on data/training/architecture.
- Newer models can be more efficient.

## **Size Overview**

- Small Models (< 7B): CPU with 16GB+ RAM. Experimentation, less demanding tasks.
- Medium Models (7B 13B): Benefit from GPU, sometimes CPU with 32GB+ RAM.
- Large Models (30B+): Powerful GPU with VRAM needed.
- Very Large Models (70B+): High-end/multiple GPUs.

#### 🔮 Tip

Model names like mistral:7b, codellama:34b - "b" = billions of parameters.

## **Quantization: Making Models Smaller**

- · Quantization: Reduce size/compute of LLMs "without significantly impacting performance".
- Represent parameters with lower precision (e.g., 8-bit, 4-bit integers).

#### **Benefits of Quantization:**

- Smaller Model Size: Less RAM/disk needed.
- Faster Inference: Faster low-precision calculations.
- · Lower Power Consumption: Less hardware demand.
- Trade-offs: Small accuracy decrease possible. More aggressive quantization = more performance impact.
- Ollama and Quantization: Supports quantized models from Hugging Face.
  - q4\_0, q8\_0 suffixes indicate quantization level.
  - q4\_0: 4-bit (balance size/quality).
  - q8\_0: 8-bit (closer to original, larger size).

#### 💡 Tip

Quantization levels offer different balance of speed, size, quality. Experiment to find best option for your hardware/needs. Default model often sufficient, quantization not always needed.

## **Getting Started with Ollama**

- 1. Download and Install Ollama: ollama.com.
- 2. Pull a Model: ollama pull mistral:7b (terminal/command prompt).

- 3. Run the Model: ollama run mistral:7b (terminal). Chat interface opens.
- 4. Ask a Question: Type prompt and press Enter.
- Experiment: Try different models from the Ollama website model library. Use Hugging Face for more models/quantization.

### Talk to the LLM in code

- To talk to the LLM in code, you can use the following code
- · Make sure that you have a local Ollama with the model you want to use running!

```
import requests
import json
def ask_llm(prompt, model="mistral:7b"):
   .....
   Send a prompt to a locally running Ollama model and get the response.
   Args:
       prompt (str): The question or prompt to send to the model
       model (str): The name of the model to use (default: "mistral:7b")
   Returns:
       str: The model's response
    .....
   url = "http://localhost:11434/api/generate"
   data = {
        "model": model,
        "prompt": prompt,
        "stream": False
   }
   try:
        response = requests.post(url, json=data)
       response.raise_for_status() # Raise an exception for bad status
       result = response.json()
       return result["response"]
    except requests.exceptions.RequestException as e:
       return f"Error: {str(e)}\nMake sure Ollama is running locally!"
def main():
   print("Local LLM Chat (type 'quit' to exit)")
   print("-" * 50)
    while True:
        user_input = input("\nYour question: ")
        if user_input.lower() in ['quit', 'exit']:
            print("\nGoodbye!")
            break
```

```
response = ask_llm(user_input)
print("\nResponse:", response)
if __name__ == "__main__":
    main()
```

To ask questions in Julia, you can use the following code:

```
using HTTP
using JSON3
function send prompt(prompt::String, model::String="mistral:7b")
   url = "http://localhost:11434/api/generate"
   headers = ["Content-Type" => "application/json"]
   body = JSON3.write(Dict(
        "model" => model,
        "prompt" => prompt,
        "stream" => false
   ))
   response = HTTP.post(url, headers, body)
   result = JSON3.read(response.body)
   return result.response
end
function chat()
   println("Chat with AI (type 'exit' to quit)")
   println("Loading model...")
   println("Note: Working in the REPL in VS Code is tricky with user input.")
   println("Thus, you first need to enter some random value and press enter.")
   println("Then, you can start chatting by asking a question.")
   println("Afterwards, you can continue by asking questions and pressing enter.")
   println("To prevent this, you have to run the code in a terminal directly.")
   while true
       print("\nYou: ")
       user_input = readline()
        if lowercase(user_input) == "exit"
           println("\nGoodbye!")
            break
        end
        try
            response = send_prompt(user_input)
           println("\nAI: ", response)
        catch e
            println("\nError: Make sure Ollama is running and the model is installed.")
            println("You can install the model with: ollama pull mistral:7b")
            break
        end
   end
```

#### end

# Start the chat
chat()

## **Recent Developments in LLM Technology**

## **Multimodality: Beyond Text**

- · Modern LLMs are evolving to work with multiple types of data:
  - Text-to-Image Models: Create detailed images from textual descriptions (DALL-E, Midjourney, Stable Diffusion)
  - Image Understanding: Analyze and describe visual content with sophisticated language
  - Audio Processing: Convert speech to text and generate natural-sounding speech
  - Video Analysis: Comprehend and narrate the content of video sequences

## **Specialized LLMs**

- · Domain-specific models are emerging to address specialized needs:
  - Medical LLMs: Trained on medical literature to assist healthcare professionals
  - Legal LLMs: Specialized in analyzing legal documents and case law
  - Scientific Models: Focused on research papers and scientific terminology
  - Code-specific Models: Optimized for software development across multiple languages

## **Frontier Research**

- · Cutting-edge research is addressing key challenges:
  - Enhanced Reasoning: Developing models with stronger logical analysis capabilities
  - Improved Factuality: Reducing hallucinations and increasing reliability of information
  - **Model Efficiency**: Creating powerful models that require fewer computational resources
  - Better Alignment: Ensuring models behave according to human values and expectations

## Wrap-up

- LLMs represent a new technology with wide-ranging applications.
  Understanding both their capabilities and limitations is essential.
  The field is evolving rapidly-staying curious and experimental will serve you well.