# Working with Large Language Models

University of Hamburg - Spring 2025

Dr. Tobias Vlćek

## Welcome to today's short workshop!

- · Explore the world of Large Language Models (LLMs).
- · You've likely interacted with LLMs already:
  - ChatGPT
  - Google's Gemini
  - Claude
- Understand:
  - How LLMs work on a high level.
  - What you can do with them.

# What is an LLM? (And Why Should You Care?)

- · LLMs are a type of Artificial Intelligence (AI).
- Specifically within Natural Language Processing (NLP).
- · Designed to:
  - Understand human language.
  - Generate human language.
  - Interact with human language.
  - In remarkably human-like ways.



**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: 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/LPZh9B0jkQs

# **Key Concepts: What You'll Learn**

- 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.

# Inside LLMs: 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: A Key Innovation

- Modern LLMs powered by transformer model architecture.
- Introduced in "Attention is All You Need" (Vaswani et al., 2017).
- · Transformers revolutionized NLP.

#### **How Transformers Work (Simplified):**

- 1. **Tokenization:** Input text broken into *tokens*.
- 2. Mathematical Relationships: Identifies relationships between tokens.
  - Understands word/phrase connections in sentences and context.
- 3. Attention Mechanism: Crucial part.
  - · Focuses on most relevant input parts when generating output.
  - · Pays "attention" to important words for meaning.

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

- · Sentence: "The cat sat on the mat because it was warm."
- · Attention mechanism understands:
  - "it" refers to "the mat," not "the cat."

- By considering context of surrounding words and relationships.

# **Tokens: The Building Blocks of Language**

- · 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.



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

# Context Window: The 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.



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: Choosing the Right Tool

- · 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: A Massive Undertaking**

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.

#### 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.

# **Ethical Implications: 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: Practical Applications

· Explore how to use LLMs practically.

# Pair Programming with LLMs: Your AI Coding Assistant

- · Pair programming: two programmers work together.
  - One writes code, other reviews/feedback.
- · LLMs as "Al 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, Al-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 with LLMs**

- Start Small: Begin with code completion, then complex tasks.
- Be Specific: Clear, concise instructions to LLM.
- Review Carefully: Always review LLM-generated code. Mistakes can happen!
- Learn from the LLM: Learn new techniques and coding styles.



**Manage Context:** Be mindful of context window. For longer codebases, LLM may "forget" earlier parts. Break down large tasks.

# Using LLMs with Your Own Data: 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.

#### **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: A Local LLM Manager**

- 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: RAM, CPU, and GPU

· 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	<ul> <li>More VRAM = larger models- NVIDIA GPUs preferred</li> </ul>
Storage	10GB+ free	- Models can be several GB each

Warning

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

## **Model Sizes: 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.
- 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.
- 5. **Experiment:** Try different models from the Ollama website model library. Use Hugging Face for more models/quantization.

# Wrap-up

- LLMs are powerful tools for many tasks.Understand their capabilities and limitations.Use them effectively in your work.