Natural Language Processing and Large Language Models

Corso di Laurea Magistrale in Ingegneria Informatica

Lesson 14

Decoder-only Transformers

Nicola Capuano and Antonio Greco

DIEM – University of Salerno

Outline

- Decoder-only transformer
- GPT
- LLAMA
- Practice on text generation

Decoder-only transformer

Decoder-only transformer

- Transformers that use only the decoder part of the Transformer architecture.
- It focuses only on decoding, making it efficient for autoregressive generation tasks.
- It lacks the separate encoder layers found in sequence-to-sequence models.
- Primarily used for language generation tasks, such as text generation, summarization, and question answering.
- Popular Examples: GPT (Generative Pre-trained Transformer) series, including GPT-2, GPT-3 and GPT-4, and LLAMA.

Decoder-only transformer

- In a decoder-only transformer, text generation is achieved through an **autoregressive approach**, where each token is generated sequentially by attending only to previously generated tokens within the same sequence, rather than using encoder-decoder attention.
- The input context (prompt) and the generated text are treated as a single, continuous sequence. This continuous sequence of tokens essentially allows the decoder-only model to handle both the “encoding” (understanding the input prompt) and “decoding” (generating text) in one step, without needing a separate encoder block.
- Once a token is generated, it’s appended to the input sequence, and the model proceeds to generate the next token.

Decoder-only transformer

- **Self-Attention with Causal Masking**: The model uses self-attention within the decoder layers, but with a causal (unidirectional) mask that prevents each token from attending to future tokens. This ensures that each position only considers information from previous positions, simulating the generation process where each word is generated sequentially.

- **Implicit Context Understanding**: In a decoder-only architecture, the model builds up context as it processes tokens in sequence. As it reads through a sequence, it "remembers" previous tokens and learns relationships between them within the attention layers. This sequential buildup of context replaces the need for separate encoder-decoder attention by allowing the model to accumulate an understanding of the input as it progresses through each token.

Encoder-only vs Decoder-only

Feature	Encoder-Only Transformers (e.g., BERT)	Decoder-Only Transformers (e.g., GPT)
Architecture	Only encoder blocks (bidirectional attention)	Only decoder blocks (causal attention)
Training Objective	Masked Language Modeling (MLM)	Autoregressive Language Modeling
Context	Processes entire sequence in parallel	Processes tokens sequentially (one by one)
Main Use Cases	Text classification, NER, question answering	Text generation, story generation, code generation
Attention Type	Bidirectional self-attention	Unidirectional (masked) self-attention
Output	Contextual embeddings for downstream tasks	Sequential token generation (text or other content)

Decoder-only transformer applications

Applications of Decoder-Only Transformers are:

- **Text Generation**: News articles, stories, and creative content.
- **Conversational AI**: Chatbots and virtual assistants for real-time dialogue.
- **Programming Help:** Code generation and debugging.
- **Summarization**: Generating concise summaries of long documents.

GPT

GPT

- **GPT** is a type of decoder-only transformer developed by OpenAI.
- It generates human-like text, understanding and predicting language; being trained on vast amounts of text data, it can perform various natural language tasks without task-specific training.
- **GPT-1** (2018): Introduced decoder-only transformer architecture, with 117 million parameters (12 decoder blocks , 768-dimensional embeddings, and 12 attention heads per block).
- **GPT-2** (2019): Significantly larger with 1.5 billion parameters in its XL version (48 decoder blocks , 1600-dimensional embeddings, and 25 attention heads per block), capable of generating coherent long-form text.
- **GPT-3** (2020): 175 billion parameters (96 decoder blocks, 12,288-dimensional embeddings, and 96 attention heads per block), with advanced capabilities in language, code, and even reasoning.
- **GPT-4** (2023): Multi-modal capabilities (image and text), improved reasoning, and broader general knowledge (detailed information on the architecture are not yet available).

GPT input encoding

- GPT models, including GPT-1, GPT-2, GPT-3, and later versions, use **Byte-Pair Encoding (BPE)** as their input encoding method.

- BPE is a **subword tokenization technique** that strikes a balance between word-level and character-level representations, breaking down words into smaller, meaningful subunits (tokens) based on frequency in the training data.

- The vocabulary size varies by model version (e.g., GPT-2 uses about 50,000 tokens), allowing efficient representation of both frequent and rare words.

GPT input encoding

The main features of BPE are:

- **Subword Tokenization**: BPE splits rare or complex words into subword units while keeping common words as single tokens. For instance, a rare word like "unhappiness" might be split into "un," "happi," and "ness."
- **Fixed Vocabulary**: BPE produces a fixed-size vocabulary (e.g., around 50,000 tokens in GPT-2), containing common words, word fragments, and some single characters. This helps the model handle a wide range of text efficiently.
- **Efficiency in Language Representation**: Subword tokens allow GPT to represent a diverse range of language patterns, handling both common and rare words effectively while reducing the total number of tokens required.

GPT input encoding

The main advantages of BPE (similar to WordPiece) are:

- **Flexibility**: Handles languages with rich morphology or new words (e.g., "AI-generated") by breaking them down into reusable subword tokens.
- **Reduced Vocabulary Size**: Keeps the vocabulary smaller and training more efficient compared to a word-level tokenizer.
- **Out-of-Vocabulary Handling**: BPE is resilient to unknown words, as it can break down any new word into familiar subwords or characters.

GPT pre-training

- GPT is pre-trained to predict the next word (or token) in a sequence, given all the previous tokens. This process is also known as **autoregressive modeling**.

- In its **Next-Token Prediction** strategy, At each step GPT model learns to minimize the difference between its predicted next token and the actual next token in the training sequence, effectively learning context and word relationships.

- The prediction is sequential, namely each token is predicted based only on previous tokens, so the model learns the patterns of language in a left-to-right order.

GPT pre-training

- GPT models are trained on massive and diverse datasets sourced from a wide array of internet text.
- GPT-1 was trained on **BookCorpus**, which consists of around 985 million words (800 MB of text)
- GPT-2 was trained on **WebText** (40 GB of text from around 8 million documents with 10 billion words), a dataset curated from high-quality web pages by OpenAI.
- GPT-3 used even larger datasets (570 GB of text with hundreds of billions of words), combining sources like **Common Crawl**, **Books**, **Wikipedia**, and more.
- In all the cases, the data is selected to cover a broad range of topics and linguistic structures to make the model versatile across different domains.
- OpenAI's training FLOPS for GPT-4 is ~2.15e25, on ~25,000 A100s for 90 to 100 days.

GPT pre-training

- GPT minimizes the **cross-entropy loss** between the predicted token probabilities and the actual tokens. This loss function is well-suited to classification tasks (like predicting the next token) and provides the model with feedback on its predictions.
- GPT uses the **Adam optimizer**, an adaptive gradient descent technique, which helps accelerate convergence by adjusting learning rates based on past gradients.
- GPT applies a **learning rate scheduling** in which learning rates are gradually increased (warm-up) in the early stages and then decayed to prevent instability during training.
- **Large batch sizes** are used to stabilize training and make the model better at generalizing across diverse language patterns.

GPT fine tuning

- Fine-tuning of GPT requires a dataset labeled for the specific task, such as pairs of prompts and expected responses, or inputs and target outputs.
- Examples of tasks for which GPT has been or may be fine tuned are:
  - **Customer Support Automation**: queries, issues, information.
  - **Medical Assistance**: health guidance and support patient inquiries.
  - **Legal Document Processing**: summarize legal documents and support legal research.
  - **Coding Assistance**: Provide code snippets, explanations, and debugging help.
  - **Educational Tutoring**: Answer questions, explain concepts, and support e-learning.
  - **Content Creation**: Generate blog posts, social media content, and marketing copy.
  - **Virtual Personal Assistants**: Provide reminders, manage tasks, and answer questions.

GPT strengths

- **Language Fluency and Coherence**: GPT models generate human-like, fluent, and coherent text, often indistinguishable from human writing.
- **Broad Knowledge Base**: Trained on vast datasets, GPT has extensive general knowledge across a wide array of topics, allowing it to answer questions and generate content in diverse domains.
- **Few-Shot and Zero-Shot Learning**: GPT can perform tasks with little to no task-specific training by learning from examples in the prompt (few-shot) or adapting to a task without examples (zero-shot).
- **Creative and Contextual Writing**: GPT can generate creative content, including stories, poetry, and dialogues, which makes it useful for content creation and entertainment applications.
- **Rapid Adaptation with Fine-Tuning**: Fine-tuning on task-specific data allows GPT to perform well in specialized contexts, such as technical writing, legal assistance, and customer service.
- **Scalability with Large Models**: Larger GPT models demonstrate stronger performance and generalization, especially on complex or nuanced tasks.

GPT limitations

- **Lack of True Understanding**: GPT models generate text based on patterns rather than true comprehension.
- **Sensitivity to Prompting**: GPT's responses can vary widely based on phrasing. Small changes in prompts may yield different outcomes.
- **Ethical and Bias Concerns**: GPT can reproduce biases present in its training data, leading to biased or inappropriate outputs, especially if prompts or fine-tuning data lack diversity or sensitivity.
- **Inability to Reason or Perform Complex Calculations**: GPT is limited in logical reasoning, advanced mathematics, or tasks requiring step-by-step problem-solving without explicit instruction in the prompt.
- **High Computational Requirements**: Large GPT models require significant computational power for training, fine-tuning, and deployment, making them expensive to run and maintain.
- **Limited Memory Across Interactions**: GPT lacks persistent memory across sessions, so it cannot retain information shared by users in previous interactions without explicit prompting.
- **Vulnerability to Adversarial Prompts**: Malicious prompts can manipulate GPT into producing undesirable or unintended responses.

Popular GPT variants - Codex

- **Codex** is a GPT-3 model fine-tuned by OpenAI specifically for coding and programming tasks.
- It powers GitHub Copilot and can assist with code generation, debugging, and explanations across multiple programming languages.
- The key features are coding assistance, code generation, multi-language support for programming tasks.

Popular GPT variants – MT-NLG

- **MT-NLG** is the acronym for Megatron-Turing Natural Language Generation and was developed by NVIDIA and Microsoft
- MT-NLG is one of the largest language models, with 530 billion parameters.
- It aims to improve natural language understanding and generation in tasks like summarization, question answering, and robust few-shot learning.

Popular GPT variants – GLaM

- **GLaM** is the acronym for Generalist Language Model, developed by Google Research
- GLaM is a sparse mixture-of-experts model with 1.2 trillion parameters, but only a fraction are active per inference.
- It uses fewer resources than fully dense models like GPT-3 while achieving competitive performance across NLP tasks.

Popular GPT variants – PanGu-α

- **PanGu-α** is Huawei’s Chinese language model with 200 billion parameters, aimed at understanding and generating text in Mandarin.
- It was developed as part of Huawei’s effort to advance AI in Chinese NLP and has applications in Chinese literature, customer support, and translation.
- In general, it supports Chinese-specific language applications.

Popular GPT variants – Chinchilla

- **Chinchilla** is a DeepMind model optimized for efficiency in training data and parameters.
- It has a smaller number of parameters than GPT-3 but achieves similar or better performance, demonstrating an alternative approach to large-scale model training.
- It is thus optimized for research and practical applications.

Popular GPT variants – OPT

- **OPT** is the acronym for **Open Pretrained Transformer**, developed by Meta (Facebook AI).
- OPT models are a series of open-source language models from Meta, comparable to GPT-3 in size and capabilities.
- Meta released these models to support transparency in AI research, offering model weights and code for research and academic use.

Popular GPT variants – BLOOM

- **BLOOM** is the result of the BigScience collaborative project
- It is an open-source multilingual model with 176 billion parameters, trained by a global consortium of researchers.
- It supports 46 languages, including underrepresented languages, and is designed to make large language models accessible for diverse linguistic and cultural contexts.
- It enforces inclusivity in NLP research.

LLAMA

LLAMA

- The **LLaMA** (Large Language Model Meta AI) architecture is a family of transformer-based language models developed by Meta. LLaMA models are designed to be efficient, high-performing, and optimized for a range of NLP tasks.
- The LLaMA family includes several model sizes:
  - **LLaMA-7B**: 32 decoder blocks with 32 attention heads for each block, 4096-dimensional embeddings
  - **LLaMA-13B:** 40 decoder blocks with 40 attention heads for each block, 5120-dimensional embeddings
  - **LLaMA-30B:** 60 decoder blocks with 40 attention heads for each block, 6656-dimensional embeddings
  - **LLaMA-65B:** 80 decoder blocks with 64 attention heads for each block, 8192-dimensional embeddings
- These models are designed to offer a range of capabilities depending on the computational resources available, from smaller more efficient models to larger models.

LLAMA input encoding

- Also LLAMA models use **Byte-Pair Encoding (BPE)** as their input encoding method, obtaining a dictionary of 32768 tokens.

- However, LLaMA uses **relative positional encodings** instead of absolute positional encodings.

- This method allows the model to better handle varying sequence lengths and to generalize across different contexts, which is particularly useful for longer sequences.

- In relative positional encoding, the model learns the relationships between tokens based on their relative positions rather than their absolute positions in the sequence.

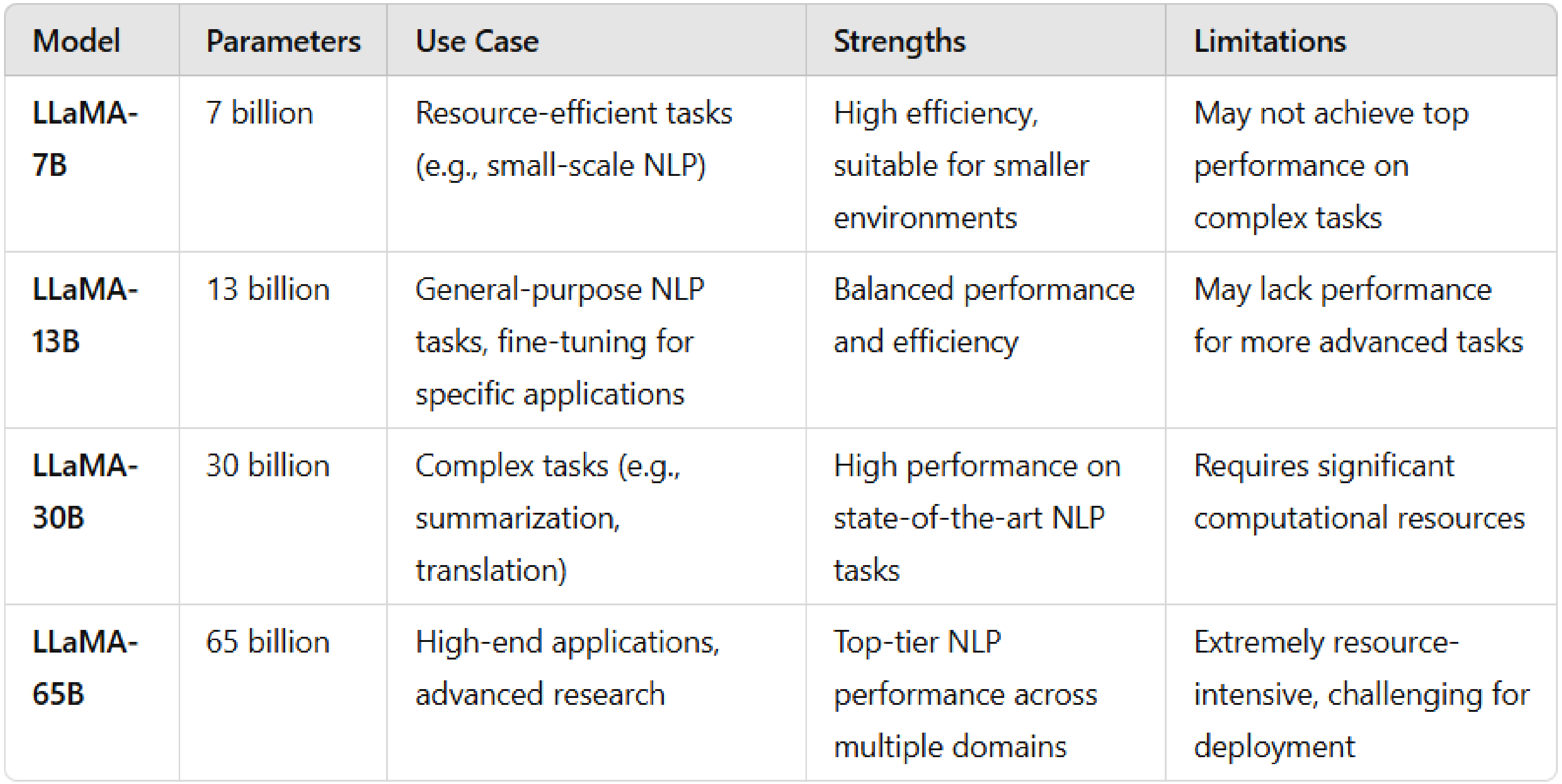
LLAMA pre-training

- Like GPT models, LLaMA is pre-trained using an autoregressive language modeling objective. This means that the model learns to predict the next token in a sequence given the previous tokens.
- LLaMA is trained on "The Pile" (825 GB, from 300 to 1000 billion tokens), a wide range of publicly available text sources, including:
  - Books (e.g., text from various domains like literature, non-fiction, etc.)
  - Web Data (e.g., content from the internet, scraped from publicly accessible websites)
  - Scientific Papers (e.g., research articles, preprints, and academic papers)
- The dataset is designed to be as diverse as possible to ensure the model learns a broad understanding of language and can generalize well to a wide range of tasks.

LLAMA pre-training

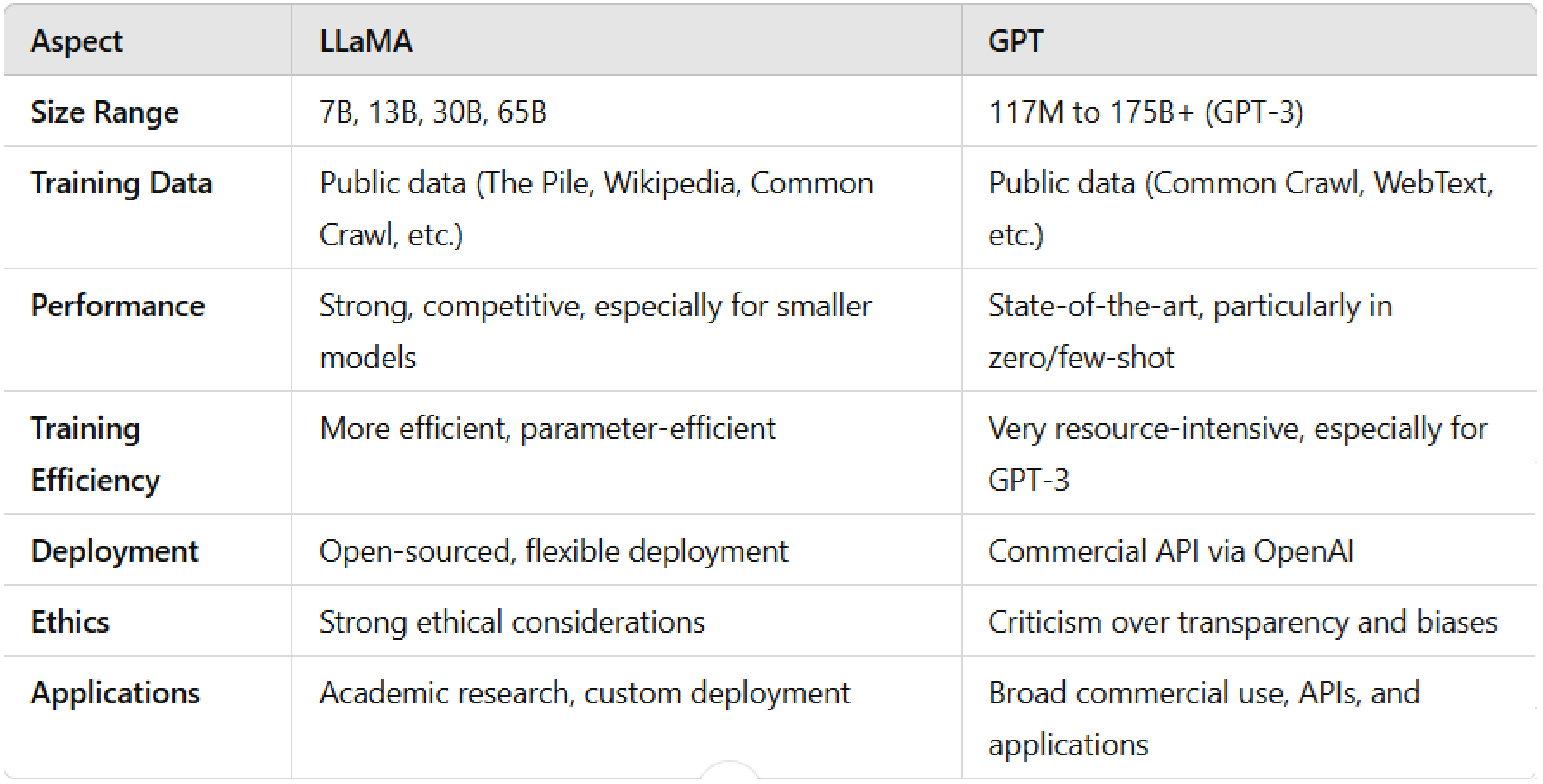
- The **loss function** used during pre-training is the **cross-entropy loss** between the predicted token probabilities and the actual next token in the sequence. This helps the model learn to predict the correct token given the context.
- The model is optimized using **Stochastic Gradient Descent (SGD)** or **Adam** optimizer with gradient clipping to prevent instability during training.
- Training also utilizes **mixed precision** to speed up computation and reduce memory requirements.
- LLaMA employs techniques like learning rate schedules (e.g., linear warm-up followed by decay), weight decay, and batch normalization or layer normalization to stabilize and improve training.

LLAMA variants



Model	Parameters	Use Case	Strengths	Limitations
LLaMA-7B	7 billion	Resource-efficient tasks (e.g., small-scale NLP)	High efficiency, suitable for smaller environments	May not achieve top performance on complex tasks
LLaMA-13B	13 billion	General-purpose NLP tasks, fine-tuning for specific applications	Balanced performance and efficiency	May lack performance for more advanced tasks
LLaMA-30B	30 billion	Complex tasks (e.g., summarization, translation)	High performance on state-of-the-art NLP tasks	Requires significant computational resources
LLaMA-65B	65 billion	High-end applications, advanced research	Top-tier NLP performance across multiple domains	Extremely resource-intensive, challenging for deployment

LLAMA vs GPT



Aspect	LLaMA	GPT
Size Range	7B, 13B, 30B, 65B	117M to 175B+ (GPT-3)
Training Data	Public data (The Pile, Wikipedia, Common Crawl, etc.)	Public data (Common Crawl, WebText, etc.)
Performance	Strong, competitive, especially for smaller models	State-of-the-art, particularly in zero/few-shot
Training Efficiency	More efficient, parameter-efficient	Very resource-intensive, especially for GPT-3
Deployment	Open-sourced, flexible deployment	Commercial API via OpenAI
Ethics	Strong ethical considerations	Criticism over transparency and biases
Applications	Academic research, custom deployment	Broad commercial use, APIs, and applications

Practice on text generation

Practice

- Looking at the Hugging Face guide on text generation https://huggingface.co/tasks/text-generation, use various models to perform text generation (code, stories, and so on)
- Search for possible models for text generation in Hugging Face https://huggingface.co/models?pipeline_tag=text-generation&sort=trending
- If you have time and computational resources, you can also fine tune one of the text generation models (https://huggingface.co/blog/ImranzamanML/fine-tuning-1b-llama-32-a-comprehensive-article)

Natural Language Processing and Large Language Models

Corso di Laurea Magistrale in Ingegneria Informatica

Lesson 14

Decoder-only Transformers

Nicola Capuano and Antonio Greco

DIEM – University of Salerno