<https://huggingface.co/learn/llm-course/chapter1/1>

<https://www.datacamp.com/courses/large-language-models-llms-concepts>

**📘 Lesson Notes: Large Language Models (LLMs) & Pretrained LLMs Ecosystem**

## **Lesson Introduction**

Welcome! Today’s lesson is **Large Language Models (LLMs) & Pretrained LLMs Ecosystem**. You will understand what LLMs are, why they matter, how different architectures work, what open vs proprietary models are, and how the lifecycle from pretraining → fine-tuning → inference functions.

### **Mini Agenda**

|  |  |
| --- | --- |
| **Section** | **Topics Covered** |
| 1 | Real-World Hook |
| 2 | Introduce Topic as Solution |
| 3 | Theory: Definitions, Scaling Laws, Architectures (GPT, BERT, T5, PaLM, LLaMA), Open vs Proprietary, LLM Lifecycle |
| 4 | Practical Example |
| 5 | Business Scenario / Use-Case |
| 6 | Practice Session (with questions) |
| 7 | Case Study |
| 8 | Quiz / Assignments |
| 9 | Recap & Interview Questions |

## **Real-World Hook**

* **Question to ponder**: When you use Google Search, ChatGPT, or translation on your phone, what makes the system understand your query and respond meaningfully?
* **Problem scenario**: A company wants to build a chatbot that understands customer support queries in everyday language, but also produce correct technical answers. Off-the-shelf models may misinterpret, generate bad responses, or be too generic.

## **Introduce Topic as a Solution**

Large Language Models (LLMs) are the “brain” behind many modern natural language understanding and generation tasks. They are pretrained on large text corpora, which gives them a broad understanding of language, then can be fine-tuned or used as is to solve specific problems like customer support, summarization, translation, code generation, etc. They help solve the problem of “making machines understand and generate human-like text” in a flexible way.

## **Theory Explanation**

### **What is a Language Model?**

* **Definition (non-technical)**: A system that can predict or generate text, given some existing text. Think of it as a smart “autocomplete” or “next sentence predictor.”
* **Definition (technical)**: A probabilistic model that, given a sequence of tokens w1,w2,…,wn−1w\_1, w\_2, …, w\_{n-1}, estimates the probability distribution of the next token wnw\_n. Mathematically:  
   P(w1,w2,...,wn)=∏i=1nP(wi∣w1,…,wi−1) P(w\_1, w\_2, ..., w\_n) = \prod\_{i=1}^n P(w\_i \mid w\_1, …, w\_{i-1})
* **Types of LMs**
  + **Statistical LMs** (n-grams, Markov models)
  + **Neural LMs** (RNNs, LSTMs, GRUs)
  + **Transformers & LLMs** (BERT, GPT, T5, LLaMA)

### **Why use LLMs?**

* They generalize across many tasks after pretraining, often without task-specific training.
* They reduce effort: instead of building a model for each task, you can reuse a pretrained one.
* They often produce more fluent, contextually coherent responses.
* Allow few-shot or zero-shot learning: you can give a few examples or prompts and get good performance.

### **When to use LLMs**

* When you have language understanding or generation tasks: chatbots, summarization, translation, question answering, code generation.
* When data for specific task is limited but you want strong baseline performance.

### **When NOT to use LLMs**

* Very small devices or environments with severe compute or memory constraints.
* When you need strict guarantees (e.g., in regulated domains) and cannot afford hallucinations or unpredictable behavior.
* When cost (compute, inference, latency) is prohibitive.

### **2.Scaling Laws of LLMs**

* **What**: Empirical principles showing how performance improves when you scale up model size (parameters), data size (number of training tokens), and compute (training compute).
* **Why**: Offers guidance for how much to invest in size vs data vs compute to get returns.
* **When**: Useful when designing or choosing an LLM to train or use.
* **How**: For example, doubling the model parameters or dataset tends to improve certain metrics (e.g. loss, accuracy) in predictable ways, though with diminishing returns.

## **Scaling Laws of LLMs**

* **Scaling Laws (Kaplan et al., 2020)**:  
   The performance of LLMs improves predictably as you scale:  
  + **Model size** (number of parameters)
  + **Dataset size** (training tokens)
  + **Compute power**
* **Key Idea**:  
  + More parameters → better capacity
  + More data → less overfitting
  + More compute → faster & better convergence
* **Diminishing Returns**:  
   Past a certain point, doubling parameters or data → smaller performance gains.  
   → Led to **efficient fine-tuning (LoRA, QLoRA)** instead of always making models bigger.

### **Architectures of GPT, BERT, T5, PaLM, LLaMA**

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Type/Architecture** | **What it’s good for / Key Features** | **Example Use-Case** |
| **GPT (e.g. GPT-2, GPT-3, GPT-4)** | Decoder-only Transformer, autoregressive | Good at text generation; generating coherent, context-aware text; few-shot learning | Chatbots, content generation |
| **BERT** | Encoder-only; uses Masked Language Modeling (MLM) & Next Sentence Prediction (NSP) during pretraining | Excellent at understanding text; classification, question answering (extractive), sentiment analysis | Search relevance, classification tasks |
| **T5 (Text-to-Text Transfer Transformer)** | Encoder-Decoder; everything converted to text-to-text formulation | Flexible: translation, summarization, QA etc. One model handles many tasks | Translate English → French; summarization |
| **PaLM** | Very large decoder models from Google; designed for reasoning, chain-of-thought prompting, multitasking | State-of-art few-shot generalization; strong on reasoning, arithmetic, etc. | Complex prompts, large-scale applications |
| **LLaMA (Meta’s models)** | Decoder-only, more open; optimized for efficiency and downstream fine-tuning | Good performance with fewer resources; more accessible | Academic research, smaller companies using LLMs internally |

## **3. Architectures of Popular LLMs**

* **GPT (Generative Pretrained Transformer)**
  + Decoder-only transformer
  + Autoregressive (predicts next token)
  + Great for text generation & chatbots
* **BERT (Bidirectional Encoder Representations from Transformers)**
  + Encoder-only transformer
  + Uses *masked language modeling* (MLM)
  + Great for understanding tasks (classification, NER, sentiment)
* **T5 (Text-to-Text Transfer Transformer)**
  + Encoder-Decoder transformer
  + Converts every NLP task into a text-to-text format
  + Example: Summarization, translation, Q&A
* **PaLM (Pathways Language Model, Google)**
  + Large-scale decoder-only transformer
  + Trained with **Pathways** architecture for multi-task learning
  + Focus: Reasoning & few-shot generalization
* **LLaMA (Meta’s Large Language Model)**
  + Decoder-only model, trained efficiently
  + LLaMA 2 & 3 → open-source, optimized for fine-tuning
  + Lightweight compared to GPT-3/4 but highly competitive

## **4. Open-source vs Proprietary LLMs**

* **Open-source LLMs**
  + Examples: LLaMA, Falcon, Mistral, BLOOM
  + ✅ Advantages: Transparency, customizable, free to fine-tune
  + ❌ Limitations: Require infra, may lag behind proprietary ones
  + Pros: transparency, ability to fine-tune/customize, cost control, community support.
  + Cons: need infrastructure, risk of unmoderated outputs, sometimes performance behind state-of-the-art proprietary.
* **Proprietary LLMs**
  + Examples: GPT-4 (OpenAI), Claude (Anthropic), Gemini (Google)
  + ✅ Advantages: State-of-the-art performance, accessible via API
  + ❌ Limitations: Closed weights, expensive, restricted usage
  + Pros: often better raw performance, maintained, safer, managed by vendor, easy API usage.
  + Cons: less flexible, cost per use, privacy/data concerns, closed weights (can't inspect internals).

## **5. LLM Lifecycle**

1. **Pretraining**
   * Train on massive text corpora
   * Learn general language patterns
   * Costly & compute-heavy (billions of tokens)
2. **Fine-tuning**
   * Adapt pretrained model to a specific domain/task
   * Methods: Full fine-tuning, LoRA, QLoRA, Adapters
3. **Inference**
   * Deploy and use the model for predictions
   * Optimizations: Quantization, distillation, caching

### **LLM Lifecycle: Pretraining, Fine-tuning, Inference**

|  |  |  |  |
| --- | --- | --- | --- |
| **Stage** | **What** | **Inputs** | **Outputs / Purpose** |
| **Pretraining** | Train from scratch on massive text corpora (books, web, etc.) | Huge unlabelled text datasets; compute resources | Base model that has general language understanding & knowledge |
| **Fine-tuning** | Adapt base model to a specific task or domain | Labeled or domain-specific data (small/medium scale) | A model specialized for tasks (e.g. summarization, legal domain, medical) |
| **Inference** | Using the (fine-tuned or base) model to produce outputs in real time or offline | New inputs from users or applications | Predictions / generated text / answers etc. |

### **Syntax / Examples (non-code and code where applicable)**

**Prompt / Usage Example (non-code)**

“Given a customer support query, generate a helpful response.”

**Code Example (using Hugging Face, Python)**

from transformers import AutoTokenizer, AutoModelForCausalLM

tokenizer = AutoTokenizer.from\_pretrained("gpt2")

model = AutoModelForCausalLM.from\_pretrained("gpt2")

input\_text = "How do I reset my password?"

inputs = tokenizer(input\_text, return\_tensors="pt")

outputs = model.generate(\*\*inputs, max\_length=50)

print(tokenizer.decode(outputs[0], skip\_special\_tokens=True))

### **Advantages & Disadvantages**

* **Advantages**
  + Versatility across many text tasks
  + Rich contextual understanding
  + Can leverage transfer learning (pretrain + fine-tune)
  + Scalability and improvements as models get larger
* **Disadvantages**
  + High compute & cost (training and inference)
  + Possible generation of incorrect or biased content (“hallucinations”)
  + Large models are heavy, slow, require specialized hardware
  + Data privacy / model licensing issues

### **Limitations & How to Overcome Them**

* **Limitation**: Hallucinations, bias, lack of interpretability  
   **Overcome by**: using data filters, guardrails, human-in-the-loop, red teaming, model evaluation & bias correction.
* **Limitation**: Cost & compute limitation  
   **Overcome by**: efficient fine-tuning (LoRA, adapters), model quantization, distillation, using smaller open models.
* **Limitation**: Domain specificity (base model may not know domain jargon)  
   **Overcome by**: domain fine-tuning, prompt engineering, retrieval augmentation.

### **Alternatives**

* Instead of huge LLMs, use smaller models for specific tasks.
* Rule-based or feature-based NLP for simpler tasks.
* Retrieval-augmented approaches where you combine a smaller LLM with a knowledge base.

### **Key Characteristics**

* Large parameter count
* Trained on large corpora
* Capable of few-shot / zero-shot learning
* Contextual understanding & generation

### **Mathematical Intuition (if needed)**

* Loss functions typically cross-entropy over token predictions.
* Model architectures built on Transformer—with self-attention: queries, keys, values.
* Scaling laws approximate power law relationships: e.g. loss ~ (model size)^(-a) or data size, with constants.

## **Practical Example**

We’ll do two exercises using a small dataset.

### **Dataset (Sample)**

Here’s a simple dataset of **50 customer support queries** (non-technical), with expected responses. Each datapoint:

|  |  |  |
| --- | --- | --- |
| **ID** | **Query** | **Category** |
| 1 | “I forgot my account password, how to reset?” | Account issues |
| 2 | “Where is my order?” | Delivery inquiry |
| 3 | “Refund for damaged item” | Returns/Refund |
| … | … | … |
| 50 | “Change shipping address after placing order” | Delivery recall |

(Assume full table has 50 rows with varied queries & categories)

### **Exercise 1: Understanding Pretrained vs Fine-tuned Model**

* Use a pretrained LLM (e.g. GPT-2 base) to classify the category of each query by prompt engineering (no fine-tuning).
* Observe errors: which categories are mispredicted?

### **Exercise 2: Fine-tuning the LLM for Classification**

* Fine-tune a model with part of this data (say 40 datapoints), evaluate on remaining 10.
* Compare performance: pretrained only vs fine-tuned.

## **Business Scenario / Company Use Case**

**E-commerce Customer Support** A mid-sized online retailer receives thousands of customer support queries daily via email & chat. They want:

* Faster response times
* Consistent and accurate categorization of queries
* Automated responses for common issues

They use a pretrained LLM + fine-tune it on past support tickets to categorize and draft responses. For rare or high complexity queries, human agents intervene. This reduces response time, improves customer satisfaction, and lowers staffing cost.

## **Practice Session (5-10 Questions with Dataset)**

Use the same 50-query dataset.

1. Prompt the pretrained model to generate a response for “My package hasn’t arrived but the tracking shows delivered.” Does it answer correctly?
2. Which category is “I want to cancel my order after shipping” → Prompt vs Fine-tuned model: compare.
3. Identify three queries that the pretrained model misclassifies; propose prompt modifications to correct them.
4. Fine-tune using 30 training examples; test on 20. Compute accuracy, precision, recall by category.
5. Try limited compute: quantize the fine-tuned model; see if inference speed improves but accuracy drops. Report trade-off.
6. Use open-source vs proprietary model: pick one open (e.g. LLaMA) and one proprietary (via API), compare cost trade-offs for answering 100 queries.
7. Design a prompt to reduce hallucinations: Think: “Only respond based on provided support data; when unsure, ask for more clarity.” Test with ambiguous queries.
8. For a new domain (say medical inquiries), which steps would you take to adapt the LLM?

## **Case Study (Mini Project)**

**Mini Project: Building a Support Assistant Bot**

* **Problem**: A startup wants a chatbot to handle common customer issues (password reset, order status, refunds).
* **Data**: Use 1,000 past chat logs (sorted by issue type).
* **Tasks**:  
  1. Preprocess data.
  2. Choose base LLM (open-source).
  3. Fine-tune on labeled issue types and responses.
  4. Build inference via prompt templates + fallback to human agent.
  5. Evaluate on unseen queries: accuracy, response time, user satisfaction.
* **Optional Extension**: Deploy as web interface (could use Streamlit) so non-technical users can type queries and get responses, see when the model is unsure.

## **Assignments for Homework**

* Prepare 20 more sample support queries from a domain of your choice (travel, finance, healthcare). Label categories.
* Fine-tune an LLM on them; write a report comparing pretrained vs fine-tuned performance.
* Explore an open-source LLM vs a proprietary API: cost, ease of use, response quality.

## **Quiz (3-5 Short Questions)**

1. What is an LLM?
2. What is the difference between encoding & decoding architectures (e.g. BERT vs GPT)?
3. Explain “few-shot learning”.
4. What are two disadvantages of using proprietary LLMs?
5. What are the main stages in the LLM lifecycle?

## **Recap of Today’s Lesson**

* LLMs allow machines to understand/generate human language.
* Scaling laws guide model/data/compute trade-offs.
* Architectures differ: GPT (generation), BERT (understanding), T5 (text-to-text), PaLM, LLaMA.
* Open vs proprietary models have trade-offs.
* Lifecycle: pretraining → fine-tuning → inference.

## **Resources for Further Learning**

* Hugging Face’s LLM Course, Chapter 1 – “Introduction to LLMs” (Theory & Basics)
* Datacamp: “Large Language Models (LLMs) Concepts”
* Paper: “Attention Is All You Need” (Vaswani et al.)
* Paper: “Scaling Laws for Neural Language Models” (Kaplan et al.)

## **Interview Questions**

### **Theoretical Questions**

1. Define a Language Model.
2. What is the difference between encoder-only, decoder-only, and encoder-decoder architectures?
3. Explain scaling laws in LLMs.
4. What trade-offs exist between open-source and proprietary LLMs?
5. What is fine-tuning and why is it used?

### **Practical Questions**

1. Given a pretrained GPT-2 model, write code to classify a sentence into categories using prompt engineering.
2. Prepare dataset, fine-tune a small open-source LLM for text classification.
3. Show how to deploy model inference using minimal compute.
4. Given ambiguous query “The order arrived but product broken”, write prompt to make model clarify before answering.
5. Compare inference time & cost for two models: quantized vs full-precision.

If you want, I can generate **Streamlit UI template** for this lesson so learners can type queries and see pretrained vs fine-tuned model responses. Do you want me to build that?