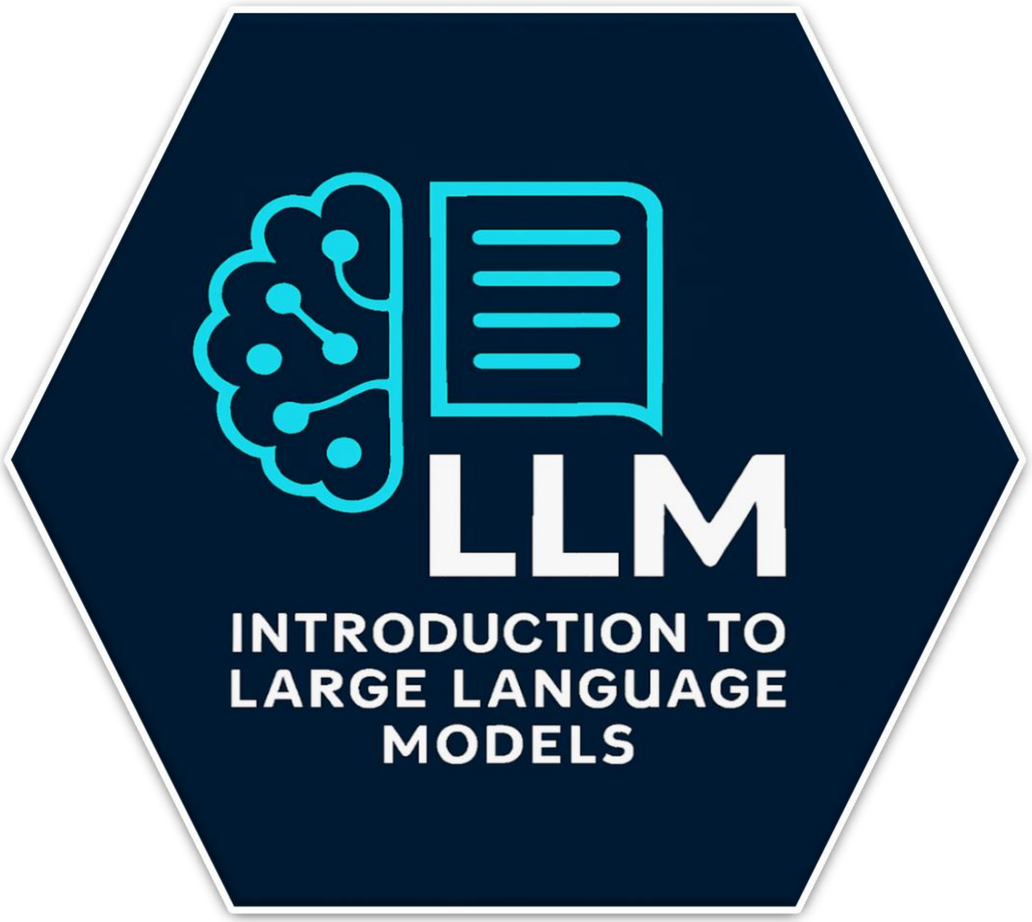


by

Ahn Nuzen

Acknowledgments

The author extends his genuine thanks to the dedicated faculty of the Grossmont College Computer Science Department for their invaluable support throughout the development of this ZBook.

Their thoughtful reviews, constructive feedback, and encouragement through many rounds of revision significantly strengthened the quality and clarity of this work.

This ZBook was made possible through funding from the Zero Textbook Cost (ZTC) Acceleration II Grant. The author is thankful to the ZTC team at Grossmont College for their support, guidance, and commitment to accessible education.

Ahn Nuzen
Grossmont College, Fall 2025

Overview

Understanding Large Language Models (LLMs) requires more than just familiarity with their textual output, it demands insight into the core principles that drive their design, training data distributions, behavior, and real-world performance. This ZBook is designed to provide that insight in an accessible, intuitive way.

Rather than delving deeply into complex mathematics or abstract theory, the goal here is to build a conceptual foundation that allows readers to reason about how LLMs work, why they behave the way they do, and how they can be applied effectively across diverse domains. Whether you're a student, developer, or technical leader, developing this intuition is key to working confidently with modern language models.

Learning Outcomes

By the end of this book, readers will be able to:

a) Conceptual Foundations of LLMs

Develop a clear and intuitive understanding of the fundamental ideas behind LLMs—including tokenization, embeddings, attention mechanisms, and transformer architecture without requiring advanced mathematical background.

b) Model Behavior & Reasoning

Analyze how LLMs process and generate text, what influences their responses, and how parameters such as model size and training data affect performance. Learn to interpret outputs critically and assess limitations such as bias, hallucination, and context sensitivity.

c) From Theory to Application

Apply foundational LLM concepts to real-world use cases such as summarization, question answering, code generation, and more. Through hands-on examples and guided exercises, connect theoretical ideas to practical tasks and system-level thinking.

Table of Contents

Module 1: What are large language models (LLMs)?

A Large Language Model (LLM) is an advanced artificial intelligence model designed to understand, generate, and interact with human language. In the context of AI, a “model” is a complex mathematical representation of something. In this case, LLM is a statistical model. The "large" in their name refers to two key aspects:

1. The number of parameters: LLMs have billions or even trillions of parameters, which are the internal variables the model learns during training. Think of them as the knobs and dials that store the model's knowledge about grammar, facts, and reasoning skills.
2. The size of the training data: They are trained on massive datasets, often encompassing a significant portion of the public internet, digital books, and other text sources.[1]

At their core, LLMs work by predicting the next most likely word (or token) in a sequence. By repeating this process, they can generate coherent sentences, paragraphs, and entire documents. Their main capabilities include text generation, summarization, translation, question-answering, and conversational chat.

1.1 A Brief History of Language Models

The development of today's powerful large language models (LLMs) is the result of several transformative stages and breakthroughs in artificial intelligence and natural language processing.

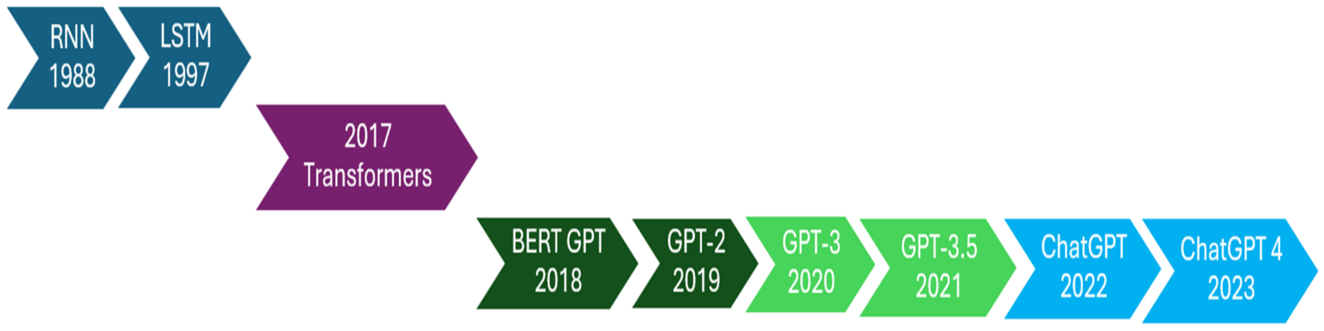


Figure-1. 1

[1] GPT-3

Here’s a detailed look at key milestones and structural advances:

a) Rule-Based & Statistical Models (Pre-2000s)

The earliest language models were statistical. **N-gram models**, for example, calculated the probability of a word appearing given the previous 'n' words.[2] These models were effective for simple tasks but lacked a true understanding of context or meaning.

b) Early Neural Networks (2000s-2010s)

Word Embeddings (e.g., Word2Vec): Around 2013, a breakthrough occurred where words could be represented as numerical vectors. Words with similar meanings were located closer to each other in this vector space, giving models a rudimentary sense of semantics for the first time.[3]

c) Recurrent Neural Networks (RNNs) Long Short-Term Memory networks (LSTMs)

These architectures were designed to process sequential data like text. They had a form of "memory" that allowed them to consider previous words when interpreting the current one, which was a significant improvement over statistical model.

d) The Transformer Revolution (2017-Present)

The modern era of LLMs was ignited by the 2017 Google research paper, "Attention Is All You Need."[4] This paper introduced the **Transformer architecture**, which abandoned the sequential nature of RNNs. Its key innovation was the **attention mechanism**, which allows the model to weigh the importance of all words in the input text simultaneously. This parallel processing capability enabled models to be trained on much larger datasets, leading to a deeper understanding of long-range context and nuanced relationships in language.

The transformer architecture gave rise to the two foundational types of modern LLMs: encoder-based models like **BERT** (Bidirectional Encoder Representations from Transformers), which excel at understanding tasks, and decoder-based models like the **GPT** (Generative Pre-trained Transformer) series, which are experts at text generation.

[2] https://en.wikipedia.org/wiki/Word_n-gram_language_model?utm_source=chatgpt.com

[3] https://en.wikipedia.org/wiki/Word2vec
[4] https://en.wikipedia.org/wiki/Attention_Is_All_You_Need

As shown in Figure 1.2, the transform architecture integrates with multi-head attention to process input sequences effectively.[5]

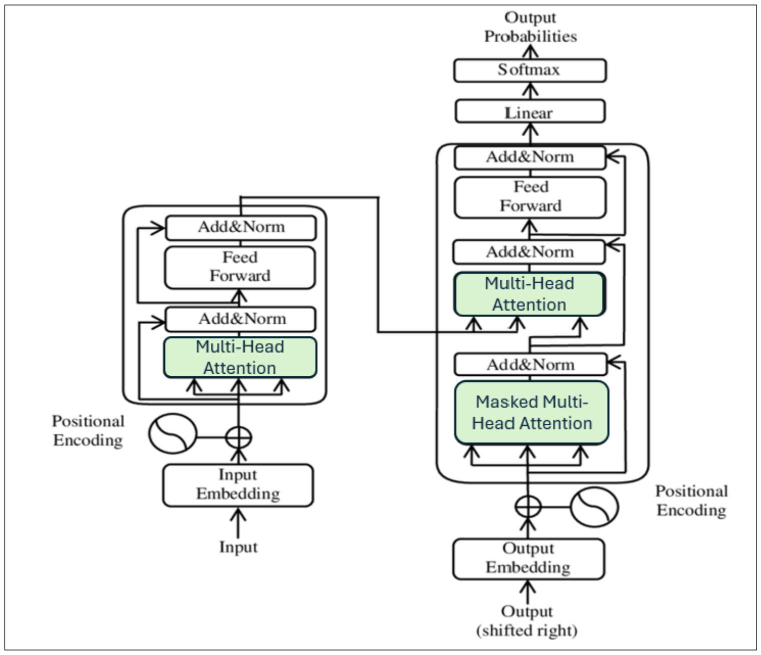


Figure-1. 2

Figure 1.3 provides an illustration of simplified AI architectures and use cases.

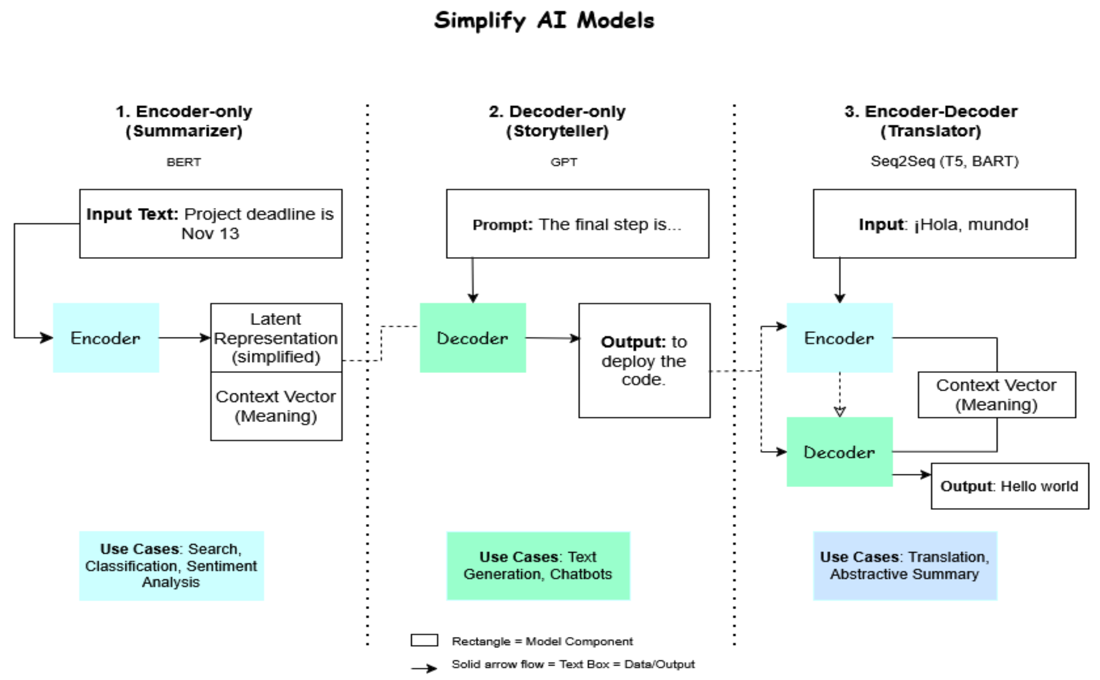

Simplify AI Models
1. Encoder-only (Summarizer)
BERT
Input Text: Project deadline is Nov 13
Encoder
Latent Representation (simplified)
Context Vector (Meaning)
Use Cases: Search, Classification, Sentiment Analysis
2. Decoder-only (Storyteller)
GPT
Prompt: The final step is...
Decoder
Output: to deploy the code.
Use Cases: Text Generation, Chatbots
3. Encoder-Decoder (Translator)
Seq2Seq (T5, BART)
Input: ¡Hola, mundo!
Encoder
Context Vector (Meaning)
Decoder
Output: Hello world
Use Cases: Translation, Abstractive Summary
Rectangle = Model Component
Solid arrow flow = Text Box = Data/Output

Figure-1. 3

[5] Transformer Model Architecture

1.2 What makes a Language Model "Large"?

A "Large" Language Model (LLM) is defined by three key factors:

- **Parameters:** LLMs possess billions or even trillions of parameters, which are the internal weights the model learns during training to process information. For example, GPT-2 2019 has 1.5 billion parameters, 2020 GPT-3 has 175 billion parameters, and 2023 GPT-4 has an estimated of 1.4 trillion parameters.[6]
- **Training Data:** They are trained on immense and diverse datasets, including books, websites, and code, which allows them to generalize their knowledge across many different subjects. GPT-3 training data set is estimated at 1.2 Terabytes.[7]
- **Compute Resources:** The training process is incredibly resource-intensive, requiring powerful GPU or TPU clusters running for weeks or months. Using the model for tasks (inference) also demands significant computational power.
- **Sample of ChatGPT-3 Training data:** As shown in Figure-1.4, GTP-3 was trained using large volumes of data collected from multiple sources.

GPT-3 training data [8]

Dataset	**# tokens**	**Proportion within training**
Common Crawl	410 billion	60%
WebText2	19 billion	22%
Books1	12 billion	8%
Books2	55 billion	8%
Wikipedia	3 billion	3%

Figure-1. 4

a) Common Crawl is a nonprofit 501(c)(3) organization that crawls the web and freely provides its archives and datasets to the public.[9] Common Crawl's web archive consists of petabytes of data collected since 2008.
b) WebText2 is a dataset of web pages that have been linked to on Reddit and received at least three upvotes**.** WebText2 was developed by OpenAI and used in the training of GPT-3. There is a publicly available, open-source version of WebText2 called OpenWebText2.
c) Books1 & Books2 are two internet-based books corpora, containing a random sample of a small subset of all the public domain books that are available online.

[6] Large Language Model (LLM)
[7] LLM training datasets
[8] GPT-3
[9] https://commoncrawl.org/

1.3 What are Tokens, embeddings, and context windows.

In large language models (LLMs), tokens are the basic units of text the model processes, and embeddings are numerical vectors that represent these tokens. Context windows refer to the maximum number of tokens (input plus output) the model can consider at once.

1.3a Tokens: They are the basic building blocks of text that a language model processes. Instead of seeing words or sentences, the model sees a sequence of tokens. A token can be a word, a part of a word, a character, or a punctuation mark.

For example, the sentence "AI models are powerful" might be broken down into five tokens:

["AI", "models", "are", "power", "ful"]

The process of breaking text down into these pieces is called **tokenization**. This allows the model to handle a vast vocabulary and understand grammatical structures by working with a fixed set of these smaller units.

1.3b Embeddings are numerical representations of tokens. Since models work with numbers, not words, each token is converted into a list of numbers called a **vector**.

The key idea is that these numbers capture the **meaning and context** of the token. Tokens with similar meanings will have similar numerical vectors. For instance, the embeddings for "cat" and "kitten" would be mathematically closer to each other than the embeddings for "cat" and "car." This allows the model to understand relationships and analogies in language.

Let's see an example of embedding vectors for cat, kitten and car.

Word	**Embedding (Vector)**
cat	[0.9, 0.2, 0.1]
kitten	[0.85, 0.25, 0.15]
dog	[0.8, 0.1, 0.2]
car	[0.1, 0.9, 0.8]

What Do These Numbers Mean?

Each “embedding” is a set of numbers—a vector—which represents one word’s meaning as a point in a multi-dimensional numerical space. This is called an embedding space.

Each number in the vector is a coordinate along one dimension of this space. Words with similar meanings get similar or nearby vectors. Figure 1.5 presents a 3D visualization of the embedding vectors for cat, kitten, dog, and car. The figure clearly demonstrates that car is distant from cat, kitten, and dog in semantic space.

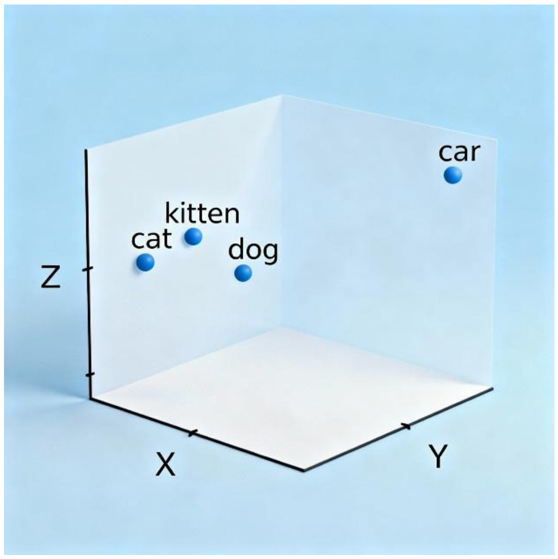


Figure-1. 5

Intuitively:

“cat” and **“kitten”** are very close to each other — they both represent small furry animals.

“cat” and **“dog”** are somewhat close — both are pets or animals.

“cat” and **“car”** are far apart — unrelated meanings.

But we can measure how “close” two embeddings are using cosine similarity.

1.3c Cosine similarity

Cosine similarity is a mathematical technique used to measure how similar two vectors are by calculating the cosine of the angle between them. In machine learning and natural language processing, data like text or images is often converted into vectors called **embeddings**, and cosine similarity helps us compare how “close” or similar those embeddings are.

- If the cosine similarity is **1**, the vectors are pointing in exactly the same direction (maximum similarity).
- If it is **0**, they are at 90°, or orthogonal meaning no similarity.
- If it is **-1**, they are pointing in exactly opposite directions (maximum dissimilarity).

The formula for cosine similarity between two vectors $\mathbf{A}$and $\mathbf{B}$ is:

$$Cosine\ \text{similarity} = \frac{\boldsymbol{A} \cdot \boldsymbol{B}}{\parallel \boldsymbol{A} \parallel \parallel \boldsymbol{B} \parallel}$$

where $\mathbf{A} \cdot \mathbf{B}$ is the dot product,
and $\parallel \mathbf{A} \parallel$ and $\parallel \mathbf{B} \parallel$ are the magnitudes of the vectors
Example:

- similarity(cat, kitten) ≈ **0.99**
- similarity(cat, dog) ≈ **0.85**
- similarity(cat, car) ≈ **0.10**

Conclusion:

Think of embeddings as coordinates on a **semantic map**:

- Words with similar meanings (like *cat*, *kitten*, *puppy*) appear **close together**.
- Unrelated words (*car*, *banana*, *justice*) are **far apart**.

This “map of meaning” is what allows models like ChatGPT to recognize patterns, analogies, and relationships in language.

1.3d Context Windows

is the maximum number of tokens a model can look at and process at one time. It's essentially the model's **short-term memory**.

If a conversation or a document is longer than the context window, the model can't "see" or remember the tokens that came before it.

- **Small Context Window:** Can lead to the model losing track of earlier parts of a long conversation or document.

- **Large Context Window:** Allows the model to handle longer texts, maintain coherence over extended dialogues, and understand complex prompts that rely on information provided much earlier.

Model capabilities are often defined by the size of their context window, which can range from a few thousand tokens to over a million in the latest models.

1.4 Core capabilities: Text generation, comprehension, and interaction.

The three core capabilities—Text Generation, Comprehension, and Interaction—work together to make Large Language Models (LLMs) powerful and versatile tools.

1.4a Text Generation

This is the ability of an LLM to create new, original text. It's the most visible "generative" aspect of Generative AI. The model takes a prompt (an input instruction) and predicts the most plausible sequence of words to follow, effectively writing on its own.

- How it works: At its core, the model is a highly advanced prediction engine. Based on the patterns learned from its vast training data, it calculates the probability of the next best word (or token) to add to the sequence and repeats this process until the response is complete.
- Examples:
    - Writing an email or a report.
    - Composing a poem or a song.
    - Generating computer code in languages like Python or SQL.
    - Creating marketing copy or product descriptions.

1.4b Comprehension

Comprehension is the model's ability to understand the meaning, context, and nuance of the text it receives. Before an LLM can generate a useful response, it must first understand the request. This is the Summarization: Reading foundational capability that makes its output relevant.

- How it works: The model uses its embeddings to convert text into numerical representations that capture semantic meaning. It analyzes these representations to identify key topics, entities, relationships, and sentiment.
- Examples:

a long document and condensing it to its main points.

- Question Answering: Extracting a specific answer from a provided text.
- Sentiment Analysis: Determining if a customer review is positive, negative, or neutral.
- Translation: Understanding a sentence in one language and re-creating its meaning in another.

1.4c Interaction

Interaction is the ability to engage in a dynamic, multi-turn conversation. This capability combines generation and comprehension in a continuous loop, allowing the model to act as a conversational partner.

- How it works: The model uses its context window (its short-term memory) to remember the previous parts of the dialogue. For each new turn, it *comprehends* the user's latest input in the context of history and then *generates* a response that is coherent and relevant to the ongoing conversation.
- Examples:
    - Chatbots and Virtual Assistants: Answering follow-up questions and maintaining a consistent dialogue flow.
    - Brainstorming Partner: Building ideas and suggestions over several exchanges.
    - Interactive Tutors: Guiding a user through a problem step-by-step, adapting to their responses.

In essence, an LLM comprehends a user's request, generates a relevant text-based answer, and uses its interaction abilities to continue the conversation coherently.

1.5 LLM is all about Autoregressive Task

Large language models (LLMs) such as GPT and similar transformers perform language generation fundamentally as an autoregressive task. In this framework, the model predicts the next token in a sequence based on all previously generated tokens. Thus, in the context of LLMs, "inner workings" at inference can largely be described as an autoregressive generation process.[10]

However, this description summarizes only the generation/inference stage. The overall training procedure includes additional nuances—such as self-supervised learning and sometimes masked prediction—but the **core mechanism for generating text is autoregression**: each output token is produced as a conditional probability given the sequence so far. We can represent Autoregressive mathematically as**:**

$$P(\mathrm{x_t} \mid x_1, x_2, \dots, x_{t-1})$$

The equation expresses that the joint probability of a sequence of tokens is decomposed into the product of conditional probabilities of each token, given all previous ones — the foundation of autoregressive text generation.

[10] Autoregressive Model

Which means, the model predict the probability of the next word/token x_t given all the previous words/tokens, and that is what GTP, LaMDA, Llama and similar models do when generating text one token at a time.

Example:

If you've seen:
`“The cat sat on the”`
Then the model computes: $P(\text{Mat}|"The\ cat\ sat\ on\ the")$
And chooses the most likely next token (e.g., "mat").

Why It’s Called “Autoregressive”?
Because the model regresses on itself — it uses its own past outputs as inputs to predict the next one.

In summary: LLMs do not "understand," "know," or "feel."

They generate text by predicting what comes next, based on training data—not by engaging in conscious thought, genuine reasoning, or moral judgment. Avoid attributing human-like abilities to LLMs, as their apparent intelligence is an illusion created by statistical prediction.

Module 1: Learning Materials & References

a) Introduction to Large Language Models
b) A brief overview of large language models
c) Large language model
d) Large Language Model (LLM)
e) Attention and Self-Attention for NLP
f) BERT vs GPT Video
g) A Practical Introduction to Large Language Models (LLMs) Video
h) What are Autoregressive Models?
i) Autoregressive (AR) Model for Time Series Forecasting
j) Understanding Autoregressive Time-Series Modeling
k) Encoder-Decoder Models

Assessment: Module 1: What are large language models (LLMs)?

Part A: Quiz

Multiple choice

1.1a In the context of a Large Language Model, what is the primary purpose of 'embeddings'?

A. To convert text tokens into numerical vectors that capture their meaning and relationships.
B. To generate new, original text based on a user's prompt.
C. To limit the amount of text the model can consider at one time, acting as its short-term memory.
D. To break down input text into smaller, manageable units like words or parts of words.

1.2a. Which of the following best describes an LLM's 'context window'?

A. The total size of the data set the model was trained on.
B. The user interface or chat application used to interact with the model.
C. The maximum amount of text (tokens) the model can process and 'remember' in a single instance.
D. The number of parameters the model has, which defines its complexity.

1.3a What are the three key components that make a language model 'Large'?

A. The algorithm used for tokenization, the type of embeddings, and the output format.
B. Billions of parameters, massive training datasets, and extensive computing resources.
C. The number of developers, the age of the model, and the popularity of its name.
D. Speed of inference, user interface design, and number of languages supported.
1.4a The 'black box' nature of LLMs refers to which of the following challenges?

A. The models are often proprietary, and their source code is not publicly available.
B. The models require a dark-themed user interface to reduce eye strain.
C. The physical servers used for training are kept in secure, windowless facilities.
D. It is difficult to understand the internal reasoning or logic the model uses to arrive at a specific answer.

1.5a When an LLM confidently generates incorrect or fabricated information as if it were a fact, this phenomenon is called:
A. A feature
B. An opinion
C. A hallucination
D. A bug

1.6a In Transformer architecture, what is the primary function of the self-attention mechanism?
A. To allow the model to weigh the importance of different words in the input sequence when processing a specific word.
B. To add information about the position of each token in the sequence.
C. To reduce the number of parameters in the model for faster inference.
D. To combine the outputs of multiple independent models into a single prediction.

1.7a What problem is 'positional encoding' intended to solve in a Transformer model?
A. The model's inability to understand the meaning of rare or specialized words.
B. The self-attention mechanism, by itself does not have an inherent sense of word order or position.
C. The high computational cost associated with processing very long text sequences.
D. The tendency of the model to generate repetitive or generic text.

1.8a What does Chain-of-Thought (CoT) prompt encourage an LLM to do?
A. Engage in a more natural, multi-turn conversation with the user.
B. Generate a response by retrieving information from an external knowledge base.
C. Summarize a long and complex piece of text into a few key bullet points.
D. Write out the intermediate reasoning steps before giving a final answer to a problem.

1.9a. How does a Retrieval-Augmented Generation (RAG) model architecture primarily work?
A. It generates multiple possible answers and uses a second model to rank them for quality.
B. It breaks down a complex problem into smaller, simpler steps for the model to solve sequentially.
C. It fetches relevant documents from an external knowledge base and provides them to the LLM as context to generate an answer.
D. It fine-tunes the entire model on a new dataset before answering a question.

1.10a In language model evaluation, what does a lower 'perplexity' score generally indicate?
A. The model is more creative and generates more diverse text.
B. The model has been trained on a larger number of different languages.
C. The model is better at predicting a sample of text; it is less 'surprised' by the correct sequence of words.
D. The model is smaller and faster to run.

1.11a What is meant by the term 'alignment' in the context of LLM safety?
A. Matching the model's parameter count to the available computing hardware.
B. The process of making the model's objectives and behaviors consistent with human values and intentions.
C. Ensuring the model's output grammatically aligns with the user's prompt.

D. Aligning the model's knowledge base with the most recent world events.

1.12a Which of the following is the best example of 'bias amplification' in an LLM?
A. An LLM refuses to answer a question that violates its safety guidelines.
B. An LLM's performance on a specific task improves after being fine-tuned on new data.
C. An LLM provides a factually incorrect answer to a question about a recent event.
D. An LLM that learned from historical texts associates the word 'doctor' more with men than women and generates examples reflecting this skew.

Part B: Activities

Activity#1: Exploring Inner Workings of LLM

This activity is broken into two parts. First, we'll test the model's core capabilities. Second, we'll probe its limitations to understand what "black box" means.

You can perform this activity using any public LLM interface, such as Google Gemini, OpenAI's ChatGPT, or Anthropic's Claude to demonstrate basic capabilities and discuss the "black box" nature of the model.

Part 1: Demonstrating Basic Capabilities

Let's test the three core functions we discussed: Generation, Comprehension, and Interaction.

1. Test: Text Generation

Objective: To see the model create original, creative text from a simple instruction.

Your Prompt:

Copy and paste the following into the LLM interface:

```
Write a short, four-line poem about the sunset over the ocean,
as seen from a beach in San Diego.
```

What to Observe:

- **Originality:** The model won't just copy a poem from the internet. It will generate a new one based on the patterns it has learned.
- **Constraint Following:** Notice if it adheres to your constraints: is it four lines long? Is the theme "sunset over the ocean in San Diego"?

2. Test: Comprehension

Objective: To see the model read, understand, and extract key information from a piece of text.

Your Prompt:

Copy and paste the following text and instruction into the LLM:

```
Article: "This afternoon, city officials in San Diego announced a new
initiative to enhance the gardens in Balboa Park. The project, scheduled to
begin in October, will focus on planting drought-tolerant native species to
conserve water while preserving the park's beauty. A spokesperson noted that
the plan aims to make the park a model for sustainable urban horticulture."
```

```
Your instruction: Based on the article above, summarize the
main point in a single sentence and identify the primary goal
of the project.
```

What to Observe:

- Summarization**:** Does the model correctly identify the main idea (the new garden initiative)?
- Information Extraction: Can it pull out the specific detail of the project's goal (to be a model for sustainable horticulture)? This proves it's not just repeating words but understanding concepts.

3. Test: Interaction

Objective: To see the model maintain context over a multi-turn conversation.

Your Prompt (First Turn):

```
Let's brainstorm ideas for a fun Friday afternoon in San Diego.
I'm starting from downtown. Give me three different options,
each with a one-sentence description.
```

Your Prompt (Second Turn, after it responds):

```
Okay, tell me more about the second option. How would I get
there from the Gaslamp Quarter and what's one thing I must do
there?
```

What to Observe:

- Memory (Context): In the second response, does the model remember what "the second option" was without you having to repeat it?
- Relevance: Does it provide a relevant answer to your follow-up questions (directions, activity)? This demonstrates its ability to have a coherent, stateful conversation.

Activity#1: Deliverable:

Students will prepare a short presentation (5-10 mins) summarizing their findings, including: LLM core capabilities, limitations, understanding inner workings of LLM.

Part 2: Discussing the "Black Box" Nature

The "black box" nature of LLMs refers to the fact that **even the creators cannot fully explain *why* the model makes a specific decision or generates a particular sequence of words.** We can see the input (our prompt) and the output (the model's response), but the internal process through billions of parameters is too complex for direct human inspection.

Why are LLMs a Black Box?

- **Immense Scale:** With billions to trillions of parameters, tracing a single path of "reasoning" is mathematically and practically impossible.
- **Emergent Properties:** Capabilities like reasoning or translation were not explicitly programmed. They *emerged* from the training process in ways that are not fully understood.[11]
- **No Verifiable Logic:** The model is a master of pattern matching, not a logical engine. It knows *that* two concepts are related, but it can't explain *why* in a provable, causal way. This can sometimes lead it to "hallucinate"—to state falsehoods with complete confidence.

Activity#2: Poking the Black Box

Objective: To see how the model can fail or "hallucinate" when its pattern-matching abilities are not enough, revealing its non-human way of "thinking."

Your Prompt:

Copy and paste this logical puzzle into the LLM:

```
I have five books on my shelf. If I take the third book, where
is the book that was originally fourth?
```

[11] https://en.wikipedia.org/wiki/Large_language_model#Emergent_abilities

What to Observe:

- **The Correct Logic:** The human answer is that: there is a gap after taking the third book, and the original fourth book stays right where it was.
- **The LLM's Response:** The model might get this right. However, it might also get it wrong or give a confusing, overly complex answer. Ask a follow-up: `How did you arrive at that conclusion?`
- The "Why": Its explanation is key. It's not retracing a logical deduction. It's ***generating a plausible-sounding** explanation* based on text patterns of how explanations are usually structured. This is the black box in action: ***the output seems intelligent, but the underlying process is opaque and not based on genuine understanding.***

Implications of the Black Box:

- Trust & Safety: It's difficult to fully trust LLMs for high-stakes applications (like medical diagnosis or financial advice) if we can't verify their reasoning.
- Bias: Biases learned from the training data can influence outputs, and it's hard to find and remove them if we can't see the internal logic.[1]
- Unpredictability: A model can work perfectly 99 times and then produce a strange or harmful output on the 100th try for reasons that are not apparent.

Activity#2: Deliverable:

Students will prepare a short presentation (5-10 mins) summarizing their findings, including: The nature of LLM Blackbox, implications of LLM being a Blackbox, and built in biases.

Part C: Python Exercises

1.1c Cosine Similarity

Write a small python program using the given embedded vector, and compare similarity between cat vs kitten, cat vs car.

Word	Embedding (Vector)
cat	[0.9, 0.2, 0.1]
kitten	[0.85, 0.25, 0.15]
dog	[0.8, 0.1, 0.2]
car	[0.1, 0.9, 0.8]

Sample output:

```
cat vs kitten: 0.99
cat vs car: 0.10
```

1.2c Autoregressive Text Generation with a Pretrained GPT-2

Write a Python script that uses Hugging Face Transformers and GPT-2 to predict the next token and auto complete the prompt by generating one token at a time, where each token depends on all previous ones by applying the autoregressive math model mentioned in section 1.5.

```
# stater code for text generation using GPT-2
from transformers import GPT2LMHeadModel, GPT2Tokenizer
import torch

# Load pre-trained GPT-2 model and tokenizer
model_name = "gpt2"
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
model = GPT2LMHeadModel.from_pretrained(model_name)
model.eval()

# The goal is to have GPT-2 auto complete the sentence with
# Initial prompt:
prompt = "The cat sat on the"
input_ids = tokenizer.encode(prompt, return_tensors='pt')

# Generate text one token at a time
output_ids = input_ids.clone()

max_tokens = 20  # how many new tokens to generate

for _ in range(max_tokens):
    # Get model output
    # loop thru output_ids to get the next token
    # append the next token to output_ids

# Decode and print the full output
print(generated_text)
```

Explanations:

autoregressive generation builds text one token at a time. In every iteration:

- Feed in the current sequence
- Get logits for the next token
- Choose the next token (via argmax or sampling)
- Append it to the sequence

What is logits? In the context of machine learning, logits are the raw, unnormalized scores produced by a model's output layer before any activation function is applied. They represent the model's initial, unscaled assessment of the likelihood of different classes or outcomes. Think of them as the "raw scores" before they are converted into probabilities

What is argmax? It is a mathematical function that finds the input that gives the maximum output of a function. It's often used in machine learning.

In other words, argmax tells you **where** the maximum value of a function occurs, not what the maximum value is. Think of it as answering the "where" question.

For example, if you have a function that gives you the temperature for each day of the week, argmax would tell you which day of the week is the hottest, not the temperature itself.

Example 1: A simple list of numbers

Let's say you have the following list of numbers: [10, 12, 14, 11, 5]

The maximum value in this list is **14**. argmax will return the **index** of that value, which is **2** (since lists are zero-indexed, the first element is at index 0, the second at index 1, and so on).

Example 2: A function

Imagine a function $f(x) = -(x-2)^2 + 3$. This function has a peak (maximum value) at $x = 2$.

- **max(f(x))** would be **3** (the maximum value of the function)
- **argmax(f(x))** would be **2** (the input x that gives the maximum value)

In machine learning, argmax is commonly used to find the most likely prediction. For example, in an image classification model, the output might be a list of probabilities for each possible class (e.g., "cat", "dog", "bird").

[0.1, 0.8, 0.1] (probabilities for "cat", "dog", "bird")

argmax would return **1**, which corresponds to "dog", as it has the highest probability.

Module 2: Introduction to Natural Language Processing (NLP) Using LLMs and Use Cases

Natural Language Processing (NLP) is a field of Artificial Intelligence (AI) that focuses on enabling computers to understand, interpret, and generate human language – essentially, to bridge the gap between how humans communicate and how computers process information. It's a complex and rapidly evolving area, drawing on concepts from computer science, linguistics, statistics, and machine learning.

2.1a Goal of NLP

For decades, computers were designed to operate with structured data – numbers, logic, and clear instructions. Human language, on the other hand, is incredibly messy, ambiguous, and full of nuance. It's filled with:

- **Ambiguity:** A single word can have multiple meanings (e.g., "bank" - a financial institution or the side of a river).
- **Context:** The meaning of a sentence often depends on the surrounding text or situation.(e.g., "The **bat** flew over the field." – baseball bat flew over the baseball field, or an animal bat flew over an open field.)
- **Variability:** Language changes constantly, with new words and phrases emerging, and existing ones taking on new meanings (e.g., "tablet" – a slat of stone, or A portable, flat computer with a touchscreen interface)
- **Grammatical Complexity:** Sentences can be structured in countless ways, following complex rules of grammar and syntax.

The goal of NLP is to develop algorithms and models that can

1. **Understand**: Determine the meaning of text or speech. This involves tasks like:

   Part-of-Speech (POS) Tagging: Identifying the grammatical role of each word (noun, verb, adjective, etc.).
   Named Entity Recognition (NER): Identifying and classifying entities like people, organizations, locations, dates, and quantities.
   Sentiment Analysis: Determining the emotional tone or attitude expressed in the text (positive, negative, neutral).

2. **Process**: Manipulate and transform language data. This can include:

   Machine Translation: Automatically converting text from one language to another
   Text Summarization: Creating concise summaries of longer texts.

Text Generation: Creating new text, such as writing articles or responding to prompts.

3. **Interact:** Communicate with humans in natural language. This is the basis of chatbots and virtual assistants.

2.1b Key Techniques and Approaches in NLP:

- **Rule-Based NLP:** Historically relied on hand-crafted rules to analyze language. While useful for specific tasks, it's often brittle and doesn't scale well.
- **Statistical NLP:** Uses statistical models to learn patterns from large amounts of text data.
- **Machine Learning (ML) & Deep Learning (DL):** Modern NLP heavily relies on ML and DL techniques, particularly using neural networks (like Recurrent Neural Networks - RNNs, Transformers) to learn complex language representations. Transformers, in particular, have revolutionized the field with models like BERT, GPT, and others.

2.2 LLMs as the New NLP Toolkit

Before Large Language Models (LLMs) like GPT, the classic Natural Language Processing (NLP) workflow relied heavily on a step-by-step pipeline that included tasks such as:

- **Tokenization:** The text was broken down into individual units – usually words but sometimes phrases or sub-word units. For example, “The cat sat on the mat.” would become ["The", "cat", "sat", "on", "the", "mat"].

- **Stemming**: This process aims to reduce words to their root form by chopping off suffixes. For example, “running”, “runs”, and “ran” would all be reduced to “run”. It’s often a crude process, sometimes resulting in non-words.

- **Lemmatization:** Similar to stemming, but more sophisticated. It reduces words to their dictionary form (lemma) while considering the context. For example, “better” would be lemmatized to “good”.

- **Part-of-speech (POS) tagging:** Each word was assigned a grammatical tag (e.g., noun, verb, adjective, adverb). This helped us understand the role of each word in the sentence. For example, “cat” would be tagged as a noun.

2.1.1 Why This Was Challenging:

Each of these steps was a discrete module, often requiring hand-coded rules, heuristics, or statistical models. The pipeline was not end-to-end: errors from one step could propagate to the next, making the overall process brittle and language specific. The challenges include:

- **Domain-Specific Knowledge:** Each step often required experts to define rules and parameters – rules that were often specific to a particular domain (e.g., medical text, legal documents).
- **Brittle & Error-Prone:** The system was fragile. It would break down if it encountered unfamiliar words, variations in spelling, or unconventional sentence structures.
- **Lack of Contextual Understanding:** The core issue was that these systems treated words in isolation, without truly *understanding* the meaning of the sentence as a whole.

2.3 LLM Revolution:

LLMs, such as GPT-3 and later models, have drastically transformed how traditional NLP tasks are performed. Mainly, LLMs don't perform these steps explicitly. Instead, they learn these relationships *implicitly* through their massive training data. They develop a deep "contextual understanding" – like a child learning language intuitively, rather than through explicit rules.

2.3a Unified, End-to-End Approach

LLMs process raw text directly and "understand" language holistically. Tokenization is embedded in their architecture, but at a more abstract, learned level.

Tasks like stemming, lemmatization, and POS tagging can be performed within the same, unified model, often without explicit pre- or post-processing.

2.3b Contextual Understanding

Unlike rule-based systems, LLMs draw on billions of parameters and immense training data, enabling them to capture deep context and subtle nuances.

For POS tagging or lemmatization, an LLM considers the entire sentence (or even surrounding sentences), leading to far higher accuracy—especially with ambiguous words.

2.3c No Hand-Crafted Features Needed

Classical pipelines require hand-engineered features or linguistic rules for each language and task.   LLMs learn these features automatically from data, generalizing across languages and domains.

2.3d Transfer Learning and Few-Shot Abilities

With LLMs, a single model can tackle multiple tasks (summarization, tagging, translation, etc.). You only need to prompt or fine-tune the model instead of building discrete pipelines for each task. Few-shot or zero-shot learning enables LLMs to perform tasks with little to no task-specific training data.

2.3e Improved Robustness and Flexibility

LLMs handle noisy, ungrammatical, or domain-specific text better due to exposure to diverse internet-scale data. Traditional pipelines often break with misspellings, idioms, or novel usages.

2.3f Traditional vs. LLM-Powered NLP: Side-by-Side

Task	Traditional NLP	LLM Approach
Tokenization	Rule-based, language-dependent	Subword tokenization, learned from data
Stemming/Lemmatization	Heuristics, lookup tables	Contextual understanding, implicit in modeling
POS Tagging	CRFs/HMMs, feature engineering	End-to-end contextual predictions
Errors & Robustness	Error cascade, brittle	Joint, holistic interpretations
Task Flexibility	Separate models/pipelines per task	Unified multitask model
Language Transfer	Manual or not supported	Cross-lingual with minor adaptation

Figure-2 1

In Summary

LLMs have virtually replaced many individual modules of the traditional NLP pipeline with a flexible, end-to-end, contextual, and robust solution. This shift enables more accurate, efficient, and adaptive NLP across languages and domains, reducing development time and expanding the reach of language technologies.

2.3 Available LLMs

Objective: Students will explore the diverse applications of LLMs across various industries and domains.

Overview of LLM Categories: Briefly touch upon different types of LLMs (e.g., GPT, PaLM, Llama).

Industry Applications: Dive into specific use cases:

2.3.1 Business & Industry:

a) **Chatbots:** Customer service, sales support.

b) **Code Generation:** Assisting developers.

c) **Market Sentiment Analysis:** Tracking brand perception.

2.3.2 Science & Academia:

a) **Research Assistance:** Summarizing papers, generating hypotheses.

b) **Data Analysis:** Extracting insights from text data.

c) **Literature Review Summarization:** Quickly understanding research trends.

2.3.3 Creative & Personal:

a) **Writing Aids:** Generating ideas, improving writing style.

b) **Content Creation:** Drafting articles, blog posts.

c) **Brainstorming Tools:** Generating new concepts.

Module 2: Learning Materials & References

a) What is NLP (Natural Language Processing)? Video
b) Large Language Models explained briefly Video
c) NLP vs LLM: Understanding Key Differences
d) LLM vs NLP
e) Natural Language Processing - Tokenization (NLP Zero to Hero - Part 1) Video
f) spaCy Python NLP Library
g) Natural Language Toolkit

Assessment: Module 2: Introduction to Natural Language Processing (NLP) Using LLMs and Use Cases

Part A: Quiz

Multiple choice

2.1a. What is the primary reason that understanding context is crucial in Natural Language Processing (NLP)?

a) It allows NLP systems to automatically correct grammatical errors.
b) It enables NLP systems to accurately determine the intended meaning of words and sentences.
c) It simplifies the process of translating languages.
d) It reduces the amount of data needed to train NLP models.

2.2a The sentence, "Visiting relatives can be annoying," has a different meaning depending on the situation. Which of the following best illustrates this concept?

a) The sentence is grammatically incorrect.
b) The sentence is a common idiom with a fixed meaning.
c) The meaning of the sentence is entirely dependent on the surrounding text or situation.
d) The sentence contains a double negative, making it confusing.

2.3a What is "Word Sense Disambiguation" (WSD) in the context of NLP?

a) The process of simplifying complex sentences.
b) The process of generating new words.
c) The process of translating languages automatically.
d) The process of determining the correct meaning of a word based on its context.

2.4a Why do Recurrent Neural Networks (RNNs) struggle to capture long-range dependencies in text?

a) They have limited memory and cannot retain information about previous words effectively.
b) They are unable to process sequential data.
c) They are designed to process all words in a sentence simultaneously.
d) They are primarily used for tasks other than language processing.

2.5a Which of the following best describes the function of the "attention mechanism" in Transformers?

a) It automatically corrects grammatical errors.
b) It allows the model to focus on the most relevant parts of the input sequence.
c) It simplifies the process of generating new text.
d) It reduces the amount of data needed to train NLP models.

Part B: Activities

Activity#1: Named Entity Recognition & Sentiment Analysis with an LLM

Raw Text: Provide a short, relevant text (e.g., a news article, a customer review).

No injuries reported after emergency landing at San Diego airport following bird strike. Flight AS 1414 from San Diego to Cabo San Lucas, Mexico, experienced a bird strike during takeoff sometime after noon, prompting an Alert 2 at the airport.
Alaska Airlines said in a statement to NBC 7 that the alert was issued "for priority handling. The captain and first officer are trained for these situations and landed the aircraft safely without any issue," the statement said.

Prompt Examples:

- "Extract all named entities (people, organizations, locations) from the following text: [Paste Text Here]"
- "Analyze the sentiment of the following text. Is it positive, negative, or neutral? [Paste Text Here]"

Activity#1 Derivable:

Students will craft effective prompts to get the desired results.
prepare a short presentation (5-10 mins) summarizing their findings, including: LLM effectiveness in NLP traditional tasks.

Activity#2: Research & Presentation on a Novel LLM Use Case

Students will research a specific industry or domain and identify a *novel or impactful* use case of an LLM.

Activity#2 Deliverable:

Students will prepare a short presentation (5-10 mins) summarizing their findings, including:

1. The industry/domain.
2. The specific LLM being used.

3. How the LLM is being applied.
4. The potential impact or benefits.

Part C: Python Exercises

2.3c textblob and spacy

Research the internet including using your favorite AI engines to understand how Python can take advantage of the following NLP packages **textblob** and **spacy** the two most popular NLP Python Packages.

Provide an overview of how both of these packages determine the components of an English sentence, the main ideas behind tokenization, part-of-speech tagging, and Lemmatization.

2.4c sentiment of the text

Write a Python Scripts that can determine if a customer review is negative positive or neutral using the python package **textblob**

2.5c Subject of the text with spacy

Write a Python Script that can determine the subject of a given English text as well as the main topic of the text using python package **spacy**.

2.6c Extra credit, other than English

For doing 2.5c in language other than English, such as Spanish, French, or Arabic.

Module 3: LLM Architecture

In this module, we will exam the innovative architecture that has redefined Natural Language Processing: the **Transformer**. This innovative design forms the backbone of nearly all contemporary Large Language Models, enabling machines to understand and generate human language with remarkable accuracy and fluency.

Throughout this module, you'll gain insights into the principles, components, and evolution of Transformer-based models, setting the foundation for understanding how today's most advanced language systems operate.

3.2 The Transformer Revolution

The modern era of LLMs was ignited by the 2017 Google research paper, "Attention Is All You Need." [12] This paper introduced the Transformer architecture, which abandoned the sequential nature of RNNs. Its key innovation was the **attention mechanism**, which allows the model to weigh the importance of all words in the input text simultaneously. This parallel processing capability enabled models to be trained on much larger datasets, leading to a deeper understanding of long-range context and nuanced relationships in language.

Before the Transformer, the state-of-the-art models for sequence data (like text) were Recurrent Neural Networks (RNNs) [13] and their more advanced variant, Long Short-Term Memory (LSTM) networks.

3.2a Sequential Nature

These models process text word by word, in order. The network maintains a "hidden state" or "memory" that gets updated after each word, carrying information from the past. For the sentence "The quick brown fox," the model would first process "The," then update its state and process "quick," and so on.

3.2b The Vanishing/Exploding Gradient Problem

The core limitation is maintaining context over long distances. Information from early words must travel through every single step to reach later words. During training, this can lead to gradients (the signals used for learning) becoming infinitesimally small (vanishing) or astronomically large (exploding). This makes it very difficult for the model to learn

[12] https://en.wikipedia.org/wiki/Attention_Is_All_You_Need
[13] https://en.wikipedia.org/wiki/Recurrent_neural_network

relationships between words that are far apart, like the connection between "she" and "Alice" in a long paragraph.

3.2c Lack of Parallelization

Because the computation for word **t** depends on the result from word **t-1**, the process is inherently sequential. You cannot process all words in a sentence at the same time. This creates a significant bottleneck for training on the massive data sets required for LLMs.

In short, while RNNs/LSTMs were powerful, they struggled with long-range dependencies and were slow to train. The world needed a new architecture that could handle long sequences and be highly parallelizable.

3.3 Transformer architecture framework

Transformer architecture maintains a high-level structure common in sequence-to-sequence tasks like machine translation: the **Encoder-Decoder framework.**[14]

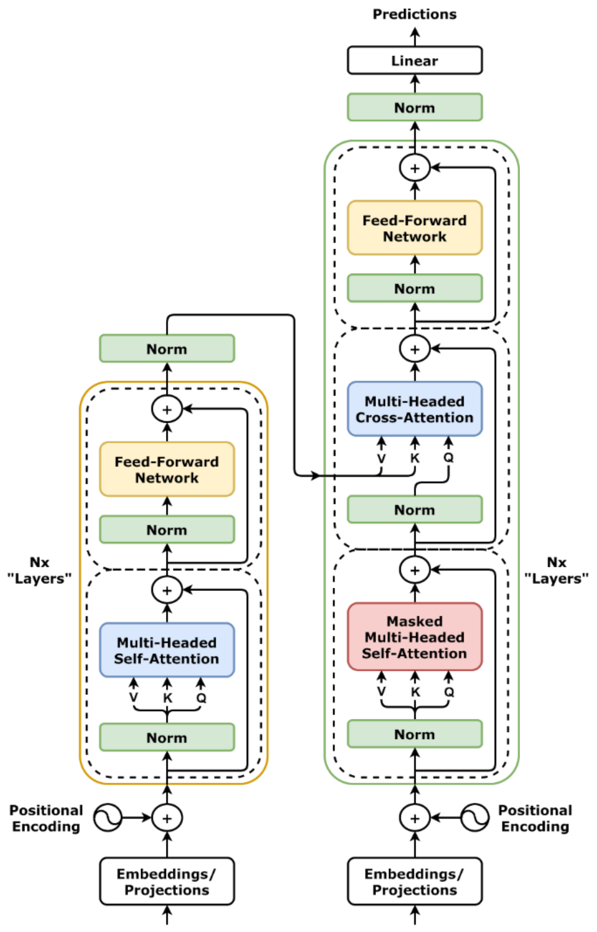


Figure-3 1

[14] A standard Transformer Architecture

3.3a The Encoder

Its job is to read and "understand" the input sentence. It takes the entire input sequence (e.g., "I love cats") and processes it to create a rich numerical representation, called a **context or Embedding vector**. This representation captures the meaning and nuances of each word in the context of the entire sentence. An encoder is typically a stack of identical encoder layers.

3.3b The Decoder

Its job is to take the embedding vector from the encoder and generate the output sequence (e.g., "J'aime les chats"). It generates the output one word at a time, using both the encoder's context and the words it has already generated to predict the next word. A decoder is also a stack of identical decoder layers.

While this framework existed before, the **internal workings** of the Transformer's encoder and decoder are what make it revolutionary. They do not use recurrence; instead, they rely entirely on the attention mechanism.

3.3c Positional Encodings

If you get rid of recurrence and process all words at once, how does the model know the order of the words? Is "The cat chased the dog" the same as "The dog chased the cat"?

To solve this, the Transformer injects information about word position into the initial input embeddings. This is done using **Positional Encodings**. It is a vector of the same dimension as the word embedding. This vector is generated using a specific, predetermined formula and is unique for each position in the sequence. This position vector is then *added* to the word embedding vector.

The original paper[15] used a combination of sine and cosine functions for Positional Encodings (PE) .The sine function is used for even indices of the embedding while the cosine function is used for odd indices.

The PE formula is expressed as follows:

$$PE_{(pos,2i)} = \sin(\frac{pos}{10000\,2i/d_{model}})$$

[15] https://en.wikipedia.org/wiki/Attention_Is_All_You_Need#Methods_discussed_and_introduced

$$PE_{(pos,2i+1)} = \cos(\frac{pos}{10000\ 2i/d_{model}})$$

Where:

- pos is the position of the word in the sequence (0, 1, 2, ...).
- i the dimension within the embedding vector (0, 1, 2, ...).
- d_model is the total dimension of the embedding vector (e.g., 512)

The key takeaway is that this formula creates a unique signature for each position that the model can learn to interpret, allowing it to understand relative and absolute positions without the sequential chain!

3.3e Feed-forward Networks and Residual Connections

Within each encoder (and decoder) layer, there are two key sub-layers:

Multi-Head Self-Attention:

This is where the model weighs the importance of other words in the sequence.

Position-wise Feed-Forward Network (FFN):

This is a simple, fully connected neural network applied independently to each position's representation after attention has been calculated. It consists of two linear transformations with a ReLU activation in between. Its purpose is to process the attention output further, transforming it into a more refined representation.

Residual Connections (or "Skip Connections"):

A residual connection is a type of shortcut in a neural network where the input to a layer (or set of layers) is added directly to the output of that layer. This forms what is called a "skip connection" or "shortcut path" that bypasses one or more layers.

This is a crucial trick for training very deep networks. The input to a sub-layer (like attention or the FFN) is ***added*** to the output of that sub-layer.

$$\text{Output} = \text{LayerNorm}(\text{x} + \text{Sublayer}(\text{x}))$$

This allows gradients to flow directly through the network, bypassing the transformation layers. It helps combat the vanishing gradient problem and makes it much easier to train deep stacks of encoder/decoder layers (e.g., 12, 24, or even 96 layers in modern LLMs). The LayerNorm (Layer Normalization) step is a normalization technique that stabilizes the training process.

In other words, a residual connection is a direct link that lets the network combine an input with the output of a transformation, allowing the network to "reuse" the input.

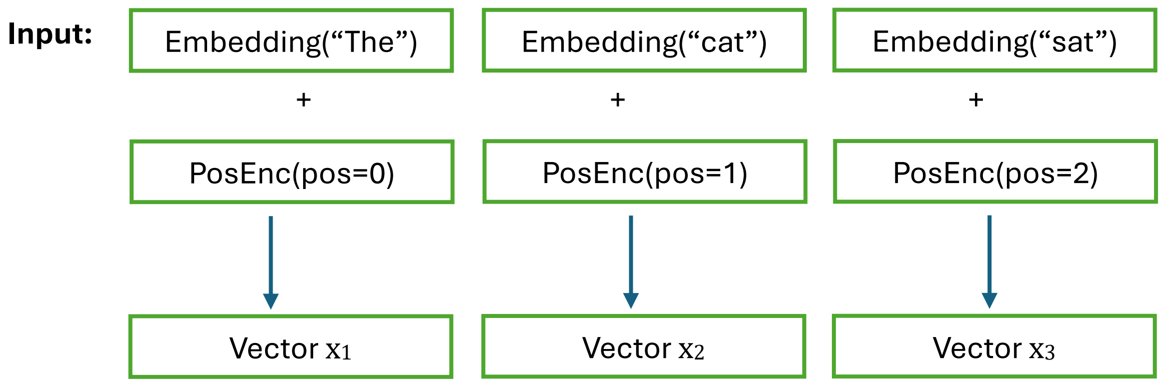
3.3f Activity: Diagram the Flow of a Short Sentence Through a Simplified Transformer Encoder Block.

Let's trace the sentence **"The cat sat"** thru a single encoder block. Assume the word embeddings and positional encodings have a dimension of 4.

Sentence: "The", "cat", "sat" **Positions:** 0, 1, 2

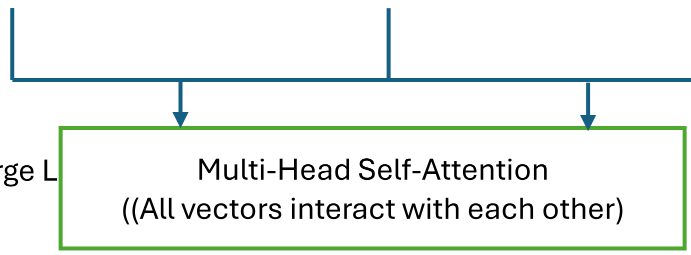
<u>**Step #1**</u>: Input Embeddings + Positional Encodings

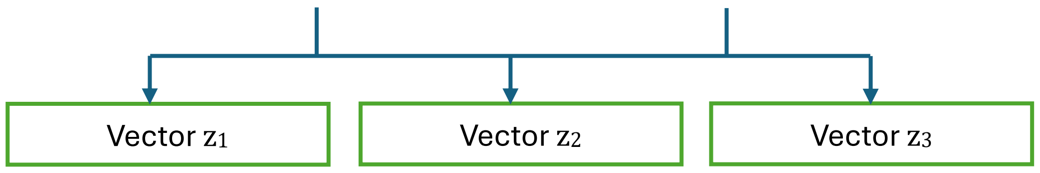
- Each word is converted into a vector (Word Embedding).
- A unique vector for each position is generated (Positional Encoding)
- These two vectors are added together to create the final input representation of the block



<u>**Step #2**</u>: Multi-Head Self-Attention

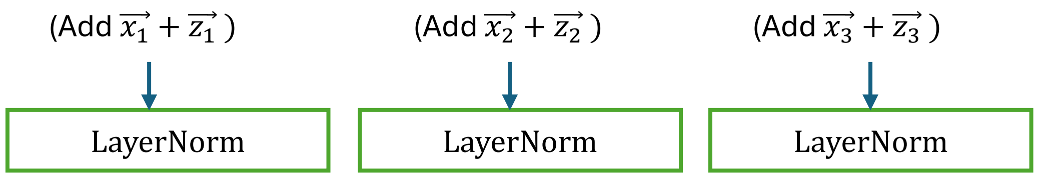
- All three vectors $\vec{x_1}$, $\vec{x_2}$, $\vec{x_3}$ are fed into the self-attention mechanism simultaneously.
- The mechanism calculates how much "attention" each world should pay to every other word (including self). For example, to process "sat," it might pay attention to "cat." unique vector for each position is generated (Positional Encoding)
- The output is a new set of vectors ($\vec{z_1}$, $\vec{z_2}$, $\vec{z_3}$), where each vector is a weighted sum of all the input vectors, enriched with contextual information.


Multi-Head Self-Attention
((All vectors interact with each other)



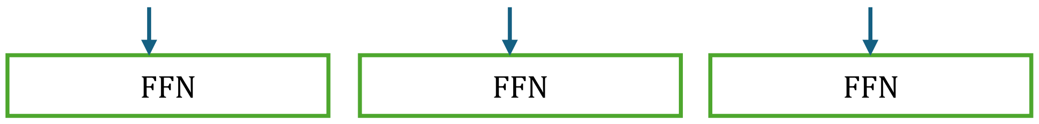
Step #3: Add & Norm (Residual Connection)

- The original three vectors $\vec{x_1}$, $\vec{x_2}$, $\vec{x_3}$ are added to the ($\vec{z_1}$, $\vec{z_2}$, $\vec{z_3}$). This creates a "Skip connection." The result is then normalized.


(Add $\vec{x_1} + \vec{z_1}$)
(Add $\vec{x_2} + \vec{z_2}$)
(Add $\vec{x_3} + \vec{z_3}$)
LayerNorm
LayerNorm
LayerNorm

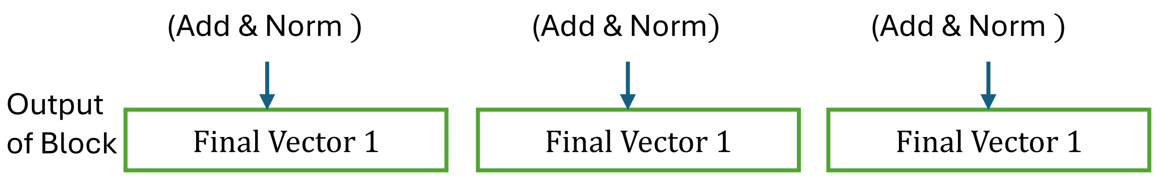
Step #4: Feed-forward Network (FFN)

- Each of the resulting vectors is passed independently thru the same Feed-Forward Network. This provides further processing and transformation.



Step #5: Add & Normal (Final output of the Block)

- Another residual connection: the input to the FFN is added to its output. The result is normalized again.
- This final set of vectors is output of the encoder block. It can be passed to the next encoder block or, if it's the final block, sent to the decoder.



The entire process happens if parallel for all words, making it incredibly efficient on modern hardware like NVIDIA GPUs.

Example:

Simple Python code using PyTorch to illustrate how to implement Add & Norm (Add and Layer Normalization) blocks from the Transformer architecture. We could easily use Tensorflow and Keras to perform the same example, but in the biased opinion of the author PyTorch is a bit more friendlier than Tensorflow.

```python
import torch
import torch.nn as nn

class AddNorm(nn.Module):
    """Residual connection followed by layer normalization."""
    def __init__(self, normalized_shape, dropout=0.1):
        super().__init__()
        self.dropout = nn.Dropout(dropout)
        self.ln = nn.LayerNorm(normalized_shape)
    def forward(self, X, Y):
        # Y: output from previous layer (sublayer), X: input to that layer
        return self.ln(self.dropout(Y) + X)

# Example usage
batch_size, seq_len, d_model = 2, 4, 8
X = torch.randn(batch_size, seq_len, d_model)      # Input
Y = torch.randn(batch_size, seq_len, d_model)      # Output from sub-layer

add_norm = AddNorm(normalized_shape=d_model, dropout=0.1)
output = add_norm(X, Y)
print(output.shape)   # Should be (2, 4, 8)
```

In this example:

- X represents the input to the sub-layer (like the input to an attention or feed-forward block).
- Y is the output from that sub-layer.
- The AddNorm block performs dropout on Y, adds X and Y (residual connection), and then normalizes the result.

You can now plug this block into any Transformer or similar architecture where Add & Norm is needed.

3.4 The Attention Mechanism

The transformer attention mechanism is a core innovation that enables the model to dynamically focus on different parts of the input sequence when processing each token. It works by computing "attention scores" using three vectors for each token—queries, keys, and values—which capture relationships between tokens.

These scores determine how much influence each token should have on another, allowing the model to weigh important contextual information effectively. Multi-head attention extends this by running several attention processes in parallel, enabling the model to attend to diverse aspects of the input simultaneously.

This mechanism is key to the transformer's ability to understand complex language patterns, capture long-range dependencies, and achieve high performance in natural language tasks.

3.4.1 self-attention

It is a mechanism that allows a model to weigh the importance of different words in an input sequence when processing a specific word.

Think about the pronoun "it" in the sentence: "The bottle is empty, so you can throw **it** away."

When a human reads this, we instantly know "it" refers to "the bottle," not "empty." Self-attention gives the model a similar capability. When the model processes the word "it," self-attention helps it calculate an "attention score" for every other word in the sentence. It will learn to assign a high score to "bottle" and low scores to the other words. The final representation of "it" will then be heavily influenced by the representation of "bottle."

This is why it's called "self"-attention: the sentence is attending to *itself*.

3.4.2 Components of: Queries (Q), Keys (K), and Values (V)

To implement the attention mechanism, we use three vectors for each input word, which are generated by multiplying the word's embedding by three distinct, learned weight matrices ($\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$):

Query (Q): Think of the Query as the current word's "question." It's a representation of the current word that says, "I am looking for other words that are relevant to me." For the word "it," the query vector would essentially ask, "What in this sentence could be thrown away?"

Key (K): Think of the Key as a word's "label" or "advertisement." It's a representation that says, "This is what I am." The key vector for "bottle" would advertise its properties as a physical, throwable object.

Value (V): Think of the Value as the word's actual "substance" or content. If the Key is the label on a filing cabinet drawer, the Value is the content inside that drawer. It's the representation we actually want to use once we've figured out how important this word is.

The attention mechanism works in two main steps:

3.4.2a Scoring:

For a given word, its **Query** vector is compared with the **Key** vector of every other word in the sentence (including itself). This comparison is done using a dot product, which produces a raw score. A high score means the key is very relevant to the query.

3.4.2b Summing:

The scores are then normalized (using SoftMax) to represent raw similar scores into normalized attention weights that sum to **1**. Each word's **Value** vector is multiplied by its corresponding weight, and all the weighted Value vectors are summed up. This produces the final output for the given word—a new representation that is a blend of all other words, biased towards the most relevant ones.

3.5 Scaled Dot-Product Attention Formula

This entire process is captured in a single matrix formula, known as Scaled Dot-Product Attention

$$Attention = (\mathrm{Q}, \mathrm{K}, \mathrm{V}) = \mathrm{softmax}\,(\frac{QK^T}{\sqrt{(d_k}})\mathrm{V}$$

Where:

$\boldsymbol{QK^T}$:The core scoring step. We multiply the Query Matrix $\mathbf{Q}$ by the transpose of the key matrix $\mathbf{K}$. This performs a dot product between every query vector and every key vector, resulting in a square matrix of attention scores.

$\sqrt{(d_k)}$: This is the scaled part. We divide all the scores by the square root of the dimension of the key vectors, $\mathrm{d_k}$. Why? For large values of $\mathrm{d_k}$, the dot products can grow very large in magnitude, pushing the softmax function into regions where its gradients are tiny. This scaling factor keeps the gradients stable and helps the model learn more effectively.

V:

This is the final step where we multiply the attention weights by the Value matrix **V**. This computes the weighted sum, creating the final output vectors where each vector is a rich, context-aware representation of the corresponding input word.

softmax:

The softmax function is applied to each row of the score matrix. This converts the raw scores into a probability distribution (i.e., positive numbers that sum to 1). These are the final attention weights.

Example:

$$\begin{array}{ccccc} \textbf{Input} & & \textbf{softmax} & & \textbf{Probabilities} \\ \begin{bmatrix} 1.6876 \\ -0.6640 \\ -0.5032 \\ -0.4353 \end{bmatrix} & \rightarrow & \begin{bmatrix} e^{x_1} / \sum_{n=0}^{N-1} e^{x_n} \\ e^{x2} / \sum_{n=0}^{N-1} e^{x_n} \\ e^{x3} / \sum_{n=0}^{N-1} e^{x_n} \\ e^{x4} / \sum_{n=0}^{N-1} e^{x_n} \end{bmatrix} & = & \begin{bmatrix} 0.8285 \\ 0.0789 \\ 0.0926 \\ 0.3486 \end{bmatrix} \end{array}$$

Python Example:

PyTorch provides a very convenient way to compute softmax in Python

```
import torch
import torch.nn as nn

sm = nn.Softmax(dim=1)    // each row is processed independently.
input = torch.randn(2, 3) // 2x3 input random matrix
output = sm(input)        // normalize input 0-1 with Softmax().

"""
input tensor([[ 1.6876, -0.6640, -0.5032],
        [-0.4353, -0.0189, -1.4793]])

output softmax tensor([[0.8285, 0.0789, 0.0926],
        [0.3486, 0.5287, 0.1227]])
"""
```

3.6 Multi-Head Attention

A single self-attention mechanism, or "head," learns one specific type of relationship. For example, one head might learn to connect pronouns to their nouns, while another might learn to connect verbs to their subjects.

To get a richer representation, the Transformer uses **Multi-Head Attention**. Instead of having one set of Q, K, V weight matrices, it has multiple sets (e.g., 8 or 12 "heads").

The process is as follows:

1. The initial input embedding is projected into multiple, lower-dimensional Q, K, and V sets—one for each head.
2. Scaled Dot-Product Attention is performed in parallel for each head. Each head produces its own output vector, focusing on a different aspect of the sentence's structure or meaning.
3. The output vectors from all heads are concatenated together.
4. This concatenated vector is passed through a final linear layer to project it back to the original model dimension and combine the learnings from all heads.

This is like having multiple experts look at the same sentence simultaneously. One might be a grammar expert, another a causality expert, and another a semantics expert. By combining their insights, the model develops a much more robust and nuanced understanding of the language.

3.7 Activity: Visualize an Attention Map for a Sample Sentence

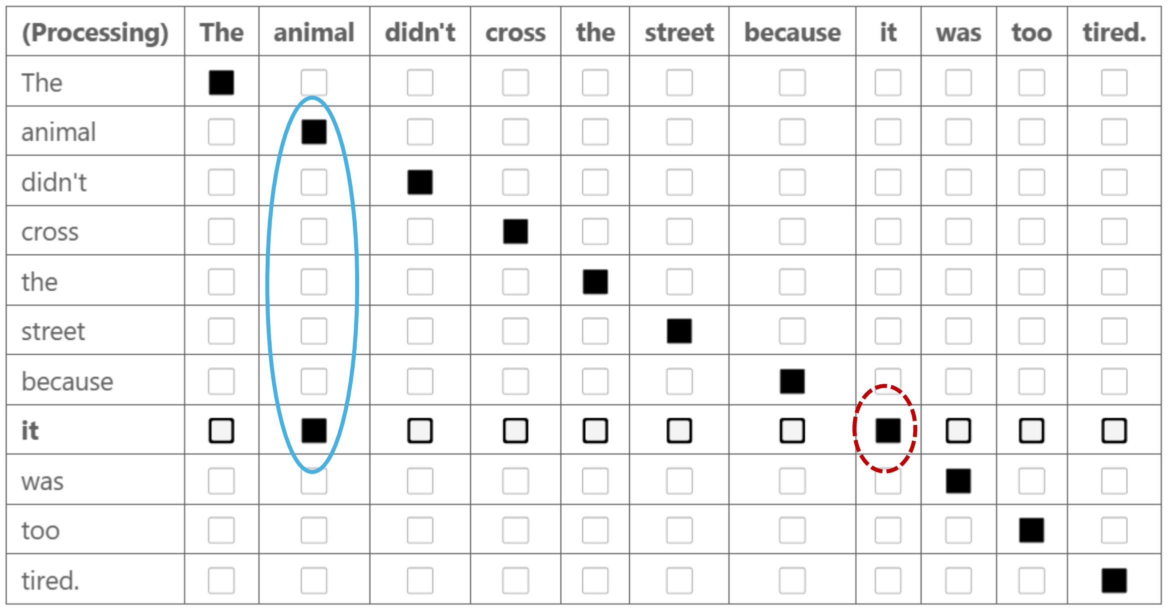
Let's visualize what the attention weights (the output of the softmax) might look like for a specific head.

Sentence: “The animal didn't cross the street because **it** was too tired."

We want to see what the model "pays attention to" when it processes the word **it**. The row corresponding to **it** in the attention matrix shows the weights it assigns to all other words.

Conceptual Attention Map:

The solid fill color of the square represents the strength of the attention weight.



(Processing)	The	animal	didn't	cross	the	street	because	it	was	too	tired.
The	■	□	□	□	□	□	□	□	□	□	□
animal	□	■	□	□	□	□	□	□	□	□	□
didn't	□	□	■	□	□	□	□	□	□	□	□
cross	□	□	□	■	□	□	□	□	□	□	□
the	□	□	□	□	■	□	□	□	□	□	□
street	□	□	□	□	□	■	□	□	□	□	□
because	□	□	□	□	□	□	■	□	□	□	□
it	□	■	□	□	□	□	□	■	□	□	□
was	□	□	□	□	□	□	□	□	■	□	□
too	□	□	□	□	□	□	□	□	□	■	□
tired.	□	□	□	□	□	□	□	□	□	□	■

Figure-3 2

Interpretation of the row for "it":

- The solid black square is at the intersection of the row "it" and the column "animal". This indicates that when the model is processing the word "it", it is paying the most attention to the word "animal".
- There are also strong self-attention scores along the diagonal (e.g., "it" attending to "it"), which is common.
- The other squares in the row are light (□), showing very low attention scores. The model has correctly learned that "it" does not refer to "street," "cross," or "because."

This ability to dynamically compute relationships between any two words in a sequence, regardless of their distance, is the central innovation of the Transformer and the key to the power of modern LLMs.

Module 3: Learning Materials & References

a) https://en.wikipedia.org/wiki/Attention_Is_All_You_Need
b) https://en.wikipedia.org/wiki/Recurrent_neural_network
c) Transformer (deep learning architecture)
d) Calculate LayerNorm with PyTorch

e) Exploring Architectures and Configurations for Large Language Models (LLMs)

Assessment: Module 3: LLM architecture

Part A: Quiz

Multiple choice

3.1a What is the primary purpose of the Encoder in Transformer architecture?

A. To add positional information to the word embeddings.
B. To generate the output sequence word by word.
C. To create a contextualized numerical representation of the input sequence.
D. To calculate the final loss and update model weights.

3.2a How does a Transformer model understand the order of words in a sentence without using recurrence (like in RNNs/LSTMs)?

A. The order is determined by the Decoder part of the model only.
B. Through the use of Positional Encodings added to the word embeddings.
C. By using a fixed-size context window for every word.
D. By processing the sentence from right to left.

3.3.a In the context of the attention mechanism, what are the roles of Queries (Q), Keys (K), and Values (V)?

A. Q, K, and V are all identical copies of the input embeddings.
B. Q is the output, K is the input, and V is the context.
C. Q calculates the position, K calculates the meaning, and V calculates the attention score.
D. Q is the current word's question, K is every other word's label, and V is every other word's content.

3.4a What is the main advantage of Multi-Head Attention over a single attention mechanism?

A. It allows the model to attend to information from different representation subspaces in parallel.
B. It exclusively handles long-range dependencies, while single-head handles short-range ones.
C. It is computationally less expensive than single-head attention.

D. It eliminates the need for a Feed-Forward Network in the encoder layer.

3.5a What is the purpose of the 'residual connections' (or skip connections) in the Transformer architecture?
A. To help gradients flow through the network more easily, aiding in the training of deep models.
B. To reset the attention scores at each layer.
C. To reduce the dimensionality of the word embeddings.
D. To combine the outputs of the encoder and decoder.

Part C: Python Exercises

3.1c Implementing Positional Encodings

Before the input embeddings are passed to the first encoder layer, a Positional Encoding vector is added to them. This function injects information about the word's position in the sequence. Your task is to implement this using the original sinusoidal formula.

Write a Python function get_positional_encoding(seq_len, d_model) that computes the positional encoding matrix for a given sequence length and model dimension.

Use the formulas 3.3c

Instructions:

a) Use the NumPy library for numerical operations.
b) The function should return a numpy array of shape (seq_len, d_model).
c) The pos ranges from 0 to seq_len - 1.
d) The i ranges from 0 to d_model / 2 - 1. The term 2i covers even indices (0, 2, 4, ...) and 2i+1 covers odd indices (1, 3, 5, ...).

Sample output: **Shape of the resulting PE matrix: (50, 512)**

3.2c Visualize Positional Encodings

Plot the resulting PE Matrix in previous exercise 3.1c Implementing Positional Encodings.

Sample output:

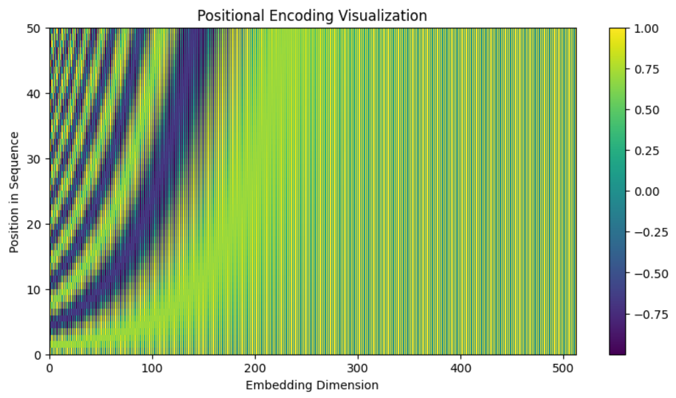


Figure-3 3

3.3c Scaled Dot-Product Attention

The core of the Transformer is the attention mechanism. It calculates a weighted sum of the Value vectors based on the similarity between Query and Key vectors.

Write a Python function scaled_dot_product_attention(Q, K, V, mask=None) that implements the attention formula in section 3.5.

Instructions:

a) Use the torch library for this implementation.
b) The inputs Q, K, and V will be torch.Tensor objects.
c) The shape of the inputs will be (batch_size, seq_len, d_k). For simplicity in this exercise, d_k is the dimension of the keys/queries.
d) The function should return the final attention output tensor and the attention weights tensor.
e) Optionally, handle a mask that can be used to prevent the model from attending to certain positions (e.g., padding tokens or future tokens in a decoder). When a mask value is 1, the corresponding attention score should be set to a very small number (like -1e9) before applying softmax.

Sample output:

```
--- Attention without Mask ---

Output shape: torch.Size([1, 5, 64])
```

```
Attention weights shape: torch.Size([1, 5, 5])
Example attention weights for the first word:
 [0.16641134 0.05426218 0.47122872 0.03142337 0.2766744 ]
Sum of weights for the first word: tensor(1.)

--- Attention with Causal Mask ---
Masked output shape: torch.Size([1, 5, 64])
Masked attention weights shape: torch.Size([1, 5, 5])
Masked weights for the 3rd word (should not attend to 4th or 5th):
 [0.5689898  0.13827316 0.29273704 0.         0.        ]
Notice how weights for positions 3 and 4 (indices) are ~0.
```

Module 4: Training and Fine-Tuning of LLMs

The goal of this module is to provide a comprehensive understanding of how Large Language Models are created, from the massive undertaking of pre-training to the practical techniques used to adapt them for specific tasks.

Learning Objectives for this module:

- Describe the three primary types of data used for pre-training and their respective roles.
- Compare and contrast the Masked Language Modeling (MLM) and Next-Token Prediction (NTP) training objectives.
- Explain the role of specialized hardware (GPUs/TPUs) in training large models and the scale of computational cost.
- Analyze the ethical challenges associated with training on vast, unfiltered internet data.

4.1 Pre-training from Scratch

A foundational model is a large-scale artificial intelligence system trained on extensive and diverse datasets. Instead of being built for one specific task from the start, it learns general patterns that can later be adapted or fine-tuned for a wide range of applications—such as answering questions, writing code, analyzing images, or translating languages.

Some leading foundational models include GPT-4, Llama 3, Gemini, and Claude 3, which serve as the "base intelligence" for many specialized AI applications.

4.1.1 How Foundational Models are Built:

- **Analogy:** Start by comparing a foundational model to a human brain at birth. It has the structure and capacity to learn, but it knows nothing about the world, language, or logic. Pre-training is the process of exposing this "brain" to a vast amount of information so it can form a general understanding of human language and knowledge.
- **Define Foundational Model:** A large-scale model pre-trained on a massive, broad dataset that can be adapted to a wide range of downstream tasks.

4.2 The Ingredient - Data at an Unfathomable Scale

The quality and quantity of the training data are arguably the most critical factors in a model's capabilities.

4.2.1 The Web (The Ocean of Data):

- **Source:** Common Crawl. It's a publicly available, petabyte-scale crawl of the web. It contains billions of web pages, including forums, articles, blogs, and more.
- **Role:** Provides sheer volume and diversity of language, covering nearly every topic imaginable. This is where the model learns the breadth of human expression.
- **Challenge:** It's incredibly noisy. It contains low-quality text, hate speech, personal information, and misinformation. Significant filtering and de-duplication are required.

4.2.2 Books (The Library of Curated Knowledge):

- **Source:** Datasets like Google Books, Project Gutenberg, and private collections.
- **Role:** Teaches the model long-range coherence, narrative structure, high-quality grammar, and deep knowledge on specific subjects. Books are professionally edited and structured, providing a high-signal learning source.
- **Example:** A model might learn about historical events from a textbook or complex character relationships from a novel.
- **Code (The School of Logic):**
  - **Source:** Publicly available code repositories, most notably a large snapshot of GitHub.
  - **Role:** Teaches reasoning, logic, and structured thinking. Code has strict syntax and logical flow (if/then, loops, functions). This helps the model "think" more methodically and is crucial for its reasoning and programming abilities.

4.3 The Recipe - Core Training Objectives

Once you have the data, you need a "task" or "game" for the model to play to learn from it. This is the training objective.

4.3a Masked Language Modeling (MLM) - "The Detective"

- **Associated Model:** BERT (Bidirectional Encoder Representations from Transformers) and its variants.
- **How it Works:** Take a sentence and randomly hide (mask) about 15% of the words.
  - Input: The quick brown [MASK] jumped over the [MASK] dog.
- **The Goal:** The model's only job is to predict the original words that were in the [MASK] positions.
- **Key Insight (Bidirectionality):** To guess the masked words correctly, the model must understand the context from *both* the left and the right side of the mask. This makes it extremely powerful for tasks that require deep understanding of the full sentence, like sentiment analysis, text classification, and question answering. It learns what a word *means* in context.

Masked Position	Model Prediction	Explanation
1st [MASK]	**fox**	Fits grammatically after "The quick brown"
2nd [MASK]	**lazy**	Fits naturally before "dog"

So, the reconstructed sentence would be:
The quick brown **fox** jumped **over** the lazy dog.

4.3b Next-Token Prediction (NTP) - "The Storyteller"

- **Also Known As:** Causal Language Modeling (CLM).
- **Associated Model:** The GPT (Generative Pre-trained Transformer) series.
- **How it Works:** Give the model a sequence of words and ask it to predict the very next word.
    - Input: The quick brown fox jumped...
- **The Goal:** The model must predict the next most probable word (e.g., over). Then, that word is added to the input, and it predicts the next one, and so on.
- **Key Insight (Autoregressive):** This process is inherently generative. By repeatedly predicting the next word, the model can write entire sentences, paragraphs, and documents. This is why these models are so good at creative writing, summarization, and dialogue.

The objective is to predict the next token given all the previous tokens (left-to-right only)

Step	Input so far	Model predicts next token
1	The	quick
2	The quick	brown
3	The quick brown	fox
4	The quick brown fox	jumped
5	The quick brown fox jumped	over
6	The quick brown fox jumped over	the
7	The quick brown fox jumped over the	lazy
8	The quick brown fox jumped over the lazy	dog

Figure4- 1

4.3c Seq2Seq Models (Sequence-to-Sequence Learning or Translator)

- **Also Known As:** Encoder–Decoder Architecture or Text-to-Text Modeling
- **Associated Models:** T5 (Text-To-Text Transfer Transformer), BART, MarianMT, and other translation or summarization models.
- **How it Works:** The model consists of two main parts — an **encoder** that reads and understands the input text, and a **decoder** that generates the corresponding output text.
    - **Input:** “Translate English to French: The cat is on the mat.”
- **The Goal:** Produce a target sequence that corresponds to the input meaning (e.g., “Le chat est sur le tapis.”).
- **Key Insight (Transformative):** Seq2Seq models are **bidirectional during encoding** (to understand full input context) and **autoregressive during decoding** (to generate coherent output step by step). This makes them ideal for tasks that transform one form of text into another, such as **translation, summarization, and paraphrasing**.

4.4c MLM, NTP and Seq2Seq Side-by-Side Summary

As shown in Figure4-2, the features and attributes of each model are highlighted for comparison.

Feature	BERT (MLM)	GPT (NTP)	Seq2Seq (Translator e.g., T5, BART)
Training Objective	Masked Language Modeling	Next Token Prediction	Text-to-Text (e.g., denoising, translation)
Context Direction	Bidirectional (both left & right)	Unidirectional (left → right)	Encoder–Decoder (input encoded, output decoded)
Uses [MASK]?	Yes	No	Yes (for denoising pretraining)
Prediction Goal	Fill in missing tokens	Predict next token	Generate output sequence based on input
Example Use	Understanding text (e.g., classification, QA)	Generating text (e.g., writing, dialogue)	Transforming text (e.g., summarization, translation, paraphrasing)
Architecture	Encoder-only	Decoder-only	Encoder–Decoder

Figure4- 2

Figure 4-3 provides an illustration of simplified AI architectures.

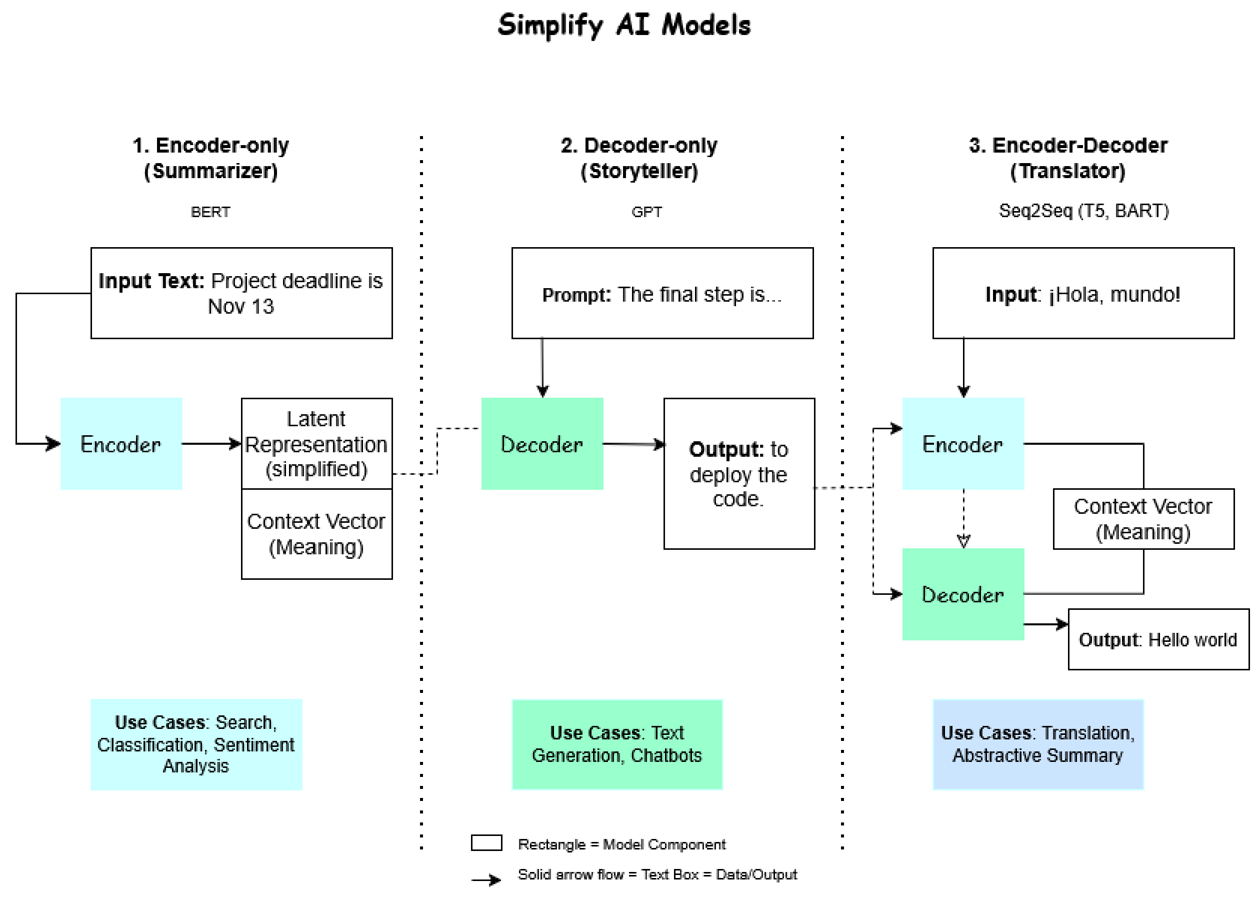

Simplify AI Models
1. Encoder-only (Summarizer)
BERT
Input Text: Project deadline is Nov 13
Encoder
Latent Representation (simplified)
Context Vector (Meaning)
Use Cases: Search, Classification, Sentiment Analysis
2. Decoder-only (Storyteller)
GPT
Prompt: The final step is...
Decoder
Output: to deploy the code.
Use Cases: Text Generation, Chatbots
3. Encoder-Decoder (Translator)
Seq2Seq (T5, BART)
Input: ¡Hola, mundo!
Encoder
Context Vector (Meaning)
Decoder
Output: Hello world
Use Cases: Translation, Abstractive Summary
Rectangle = Model Component
Solid arrow flow = Text Box = Data/Output

Figure4- 3

4.4 The Oven - Hardware and Computational Costs

- **The Challenge:** The core operation of a neural network is matrix multiplication. Training an LLM involves billions (or trillions) of parameters being updated across petabytes of data for millions of steps. This requires an astronomical number of calculations.
- **The Hardware:**
  - **GPUs (Graphics Processing Units):** Originally for rendering graphics, their architecture is designed for performing thousands of simple calculations in parallel. This is a perfect match for matrix multiplication. Companies like NVIDIA (e.g., A100, H100 GPUs) dominate this space.
  - **TPUs (Tensor Processing Units):** Custom-designed chips by Google specifically to accelerate neural network computations. They are even more specialized for matrix operations than GPUs.
- **The Cost:**
  - Training a state-of-the-art foundational model is a massive industrial-scale project.

- It requires thousands of high-end GPUs/TPUs running continuously for weeks or months.
- The cost for a single training run can be in the **tens to hundreds of millions of dollars** for electricity, hardware, and engineering. This is why only a handful of major tech companies can afford to pre-train models from scratch.

4.5: Adapting Models with Fine-Tuning and Prompting

Specializing the Giants: Techniques for Adapting Pre-trained LLMs

Learning Objectives: By the end of this lesson, learners will be able to:

1. Differentiate between full fine-tuning, parameter-efficient fine-tuning (PEFT), and prompt engineering.
2. Implement a simple fine-tuning pipeline for a text classification task using a pre-trained model.
3. Design zero-shot and few-shot prompts to solve simple tasks.
4. Explain the purpose and high-level process of Reinforcement Learning from Human Feedback (RLHF).

From Generalist to Specialist

- **Recap:** We have a pre-trained foundational model. It's like a brilliant university graduate with vast general knowledge but no specific job training.
- **The Goal:** We want to adapt this generalist model to become a specialist at a specific task (e.g., a customer support chatbot, a legal document summarizer, a sentiment analyst).
- **The Spectrum of Adaptation:** We'll explore methods from "just talking to the model" (prompting) to "sending it back to school" (fine-tuning).

4.5a Prompt Engineering - Guiding the Model Without Changing It

This is the cheapest and fastest way to adapt a model. You are not updating any model weights.

- **Concept:** The art and science of designing effective inputs (prompts) to elicit the desired output from a pre-trained model.
- **Zero-Shot Learning:** Asking the model to perform a task it was never explicitly trained on, simply by describing the task in the prompt.
  - Prompt:
  - Classify the sentiment of the following movie review as "positive", "negative", or "neutral".
  - Review: "The plot was predictable, and the acting was wooden."
  - Sentiment:

    - The model uses its general understanding of language to complete the task.
- **Few-Shot Learning:** Providing a few examples of the task within the prompt to show the model the pattern you want it to follow. This is extremely effective for guiding format and style.
    - Prompt:
    - A "tweet" is a message under 280 characters. Summarize the following text into a tweet.
    - Text: The new research paper highlights the importance of sleep for cognitive function, memory consolidation, and overall mental health. It recommends 7-9 hours for adults.
    - Tweet: New study confirms sleep is KEY for brain health & memory! Adults, aim for 7-9 hours a night. #Sleep #Health
    - ---
    - Text: Our company announced a record-breaking quarter, with revenue increasing by 20% year-over-year, driven by strong sales in our new product line.
    - Tweet:
    - The model learns from the example and is more likely to produce a concise, tweet-like summary with hashtags.

4.5b Fine-Tuning - Updating the Model's Knowledge

This involves updating the model's weights on a smaller, task-specific dataset.

- **Full Fine-Tuning:**
    - **Process:**
        1. Start with a pre-trained model (e.g., bert-base-uncased).
        2. Acquire a labeled dataset for your task (e.g., 10,000 movie reviews, each labeled "positive" or "negative").
        3. Continue the training process, but only on this new, smaller dataset. All (or most) of the model's weights (W) are updated.
    - **Pros:** Achieves the highest performance for the specific task.
    - **Cons:** Computationally expensive (though much less than pre-training). Requires storing a full copy of the model for every task. Risks "catastrophic forgetting," where the model loses some of its general abilities.
- **Parameter-Efficient Fine-Tuning (PEFT):**
    - **The Problem:** Full fine-tuning is too costly. What if we could achieve similar performance by only updating a tiny fraction of the parameters?
    - **Core Idea:** Freeze the vast majority of the pre-trained model's weights. Inject a small number of new, trainable parameters (an "adapter"). Train *only* these new parameters.
    - **Example Method: LoRA (Low-Rank Adaptation)**
        - **Analogy:** Imagine the original model is a giant, complex marble statue. Instead of re-carving the whole statue (full fine-tuning), LoRA attaches a small, lightweight, adjustable exoskeleton to it. You only train the exoskeleton.

1. **How it works (simplified):** For a large weight matrix W, LoRA learns two much smaller matrices, A and B. The update to the weights is their product , $W_{new} = W + A \times \mathrm{B}$

   We freeze W and only train A and B. Since A and B are small, the number of trainable parameters is tiny (e.g., 0.1% of the total).
2. **Benefits:** Drastically reduces memory and compute requirements. You can have one base model and many small, plug-and-play LoRA adapters for different tasks.

4.5c Reinforcement Learning from Human Feedback (RLHF) - Aligning with Our Values

- **The Problem:** A model trained with Next-Token Prediction is just trying to predict plausible text. It doesn't inherently know what is helpful, harmless, or truthful. Its goal is statistical, not ethical.
- **The Goal:** To "align" the model's behavior with human preferences for safety and helpfulness.
- **The Three-Step Process:**
  1. **(Supervised Fine-Tuning):** First, fine-tune a pre-trained model on a small, high-quality dataset of human-written prompt-response pairs. (This is a warm-up).
  2. **Train a Reward Model:**
     - Take a prompt and generate several different responses from the model.
     - Have human labelers rank these responses from best to worst.
     - Train a separate "Reward Model" on this data. Its job is to take a prompt and a response and output a score (a "reward") that predicts how a human would rate it.
  3. **Fine-tune with Reinforcement Learning:**
     - Use the trained Reward Model as an automated human preference simulator.
     - The LLM gets a prompt and generates a response.
     - The Reward Model scores that response.
     - Use an RL algorithm (like PPO) to update the LLM's weights, encouraging it to generate responses that get a higher score from the Reward Model. This steers the model towards being more helpful and harmless.

4.6 Activity : Group Discussion - The Ethics of the Data

Divide students into small groups and provide them with the following prompts. Each group discusses for 15 minutes and then share their key takeaways with the class for 10 minutes.

- **Prompt 1 (Bias Amplification):** Common Crawl is a snapshot of the internet. What societal biases (e.g., regarding gender, race, profession, nationality) exist on the internet? How would a model trained on this data learn and potentially *amplify* these biases in its responses? Give an example.
    - *Facilitator Notes:* Guide discussion towards stereotypes (e.g., "doctors are male, nurses are female"), underrepresentation of minority groups, and anglo-centric worldviews.
- **Prompt 2 (Misinformation and Harmful Content):** The web contains vast amounts of factually incorrect information (e.g., conspiracy theories, medical misinformation) and harmful content (e.g., hate speech, instructions for dangerous activities). What are the dangers if an LLM learns this content as "truth"? Who is responsible for the model's output if it generates harmful or false information?
- **Prompt 3 (Data Privacy and Copyright):** The training data includes copyrighted books, artists' work, and personal information scraped from blogs and forums without the creators' explicit consent.
    - Is it ethical to use this data to train a commercial product?
    - What are the arguments for ("fair use" for learning patterns) and against (theft of intellectual property/privacy)?

4.7 Training and Fine-Tuning of LLM using Python

This section introduces the core concepts, processes, and techniques behind training and fine-tuning large language models. You will learn how LLMs learn from data, the difference between pre-training and fine-tuning, and how Python frameworks like PyTorch and Hugging Face Transformers are used to implement these processes.

4.7.1 Understanding Model Training

Definition: Training is the process by which an LLM learns to predict the next token (word, subword, or character) from a given context.

Process:

- Tokenization
- Forward and backward propagation
- Loss calculation (e.g., Cross-Entropy Loss)
- Optimization (e.g., Adam optimizer)

Python Example: Training loop with PyTorch

```
for epoch in range(num_epochs):
    for batch in data_loader:
        optimizer.zero_grad()
```

```
            outputs = model(**batch)
            loss = outputs.loss
            loss.backward()
            optimizer.step()
```

4.7.2 Fine-Tuning Pre-Trained Models

Definition: Fine-tuning adapts a general-purpose LLM to a domain-specific task (e.g., cybersecurity text classification).

Steps:

1. Load a pre-trained model (e.g., bert-base-uncased)
2. Add task-specific layers (e.g., classification head)
3. Train on a smaller, domain-specific dataset

Python Example: Fine-Tuning a Transformer Model for Cybersecurity Threat Detection

Objective

In this exercise, you will fine-tune a pre-trained transformer model to classify cybersecurity-related text as either **benign** or **threat**.
You will use the **Hugging Face Transformers** and **PyTorch** libraries in Python.

Background

Large Language Models (LLMs) like **BERT**, **RoBERTa**, and **DistilBERT** have been pre-trained on large corpora of text to understand general language patterns.
Fine-tuning allows you to adapt these models to specific tasks, such as cybersecurity threat detection — where the model learns to identify potential security threats from text logs, alerts, or messages.

For example:

- “Unauthorized login attempt detected” → **Threat**
- “System backup completed successfully” → **Benign**

Dataset

A small, labeled dataset of cybersecurity-related messages:

Text	Label
"Multiple failed login attempts from unknown IP address."	Threat
"File transfer completed successfully."	Benign
"Detected port scanning from external host."	Threat
"User logged out normally."	Benign

Label Encoding:

0 = Benign
1 = Threat

Step 1: Import Required Libraries

```
# if modules are not installed install the following modules
#!pip install transformers torch scikit-learn
# Step 1: Import Required Libraries
from transformers import (
    AutoTokenizer, AutoModelForSequenceClassification,
    Trainer, TrainingArguments)
from sklearn.model_selection import train_test_split
from transformers.trainer import Trainer

from datasets import Dataset
import torch
```

Step 2: Prepare the Dataset

```
data = [
    {"text": "Multiple failed login attempts from unknown IP address.", "label": 1},
    {"text": "File transfer completed successfully.", "label": 0},
    {"text": "Detected port scanning from external host.", "label": 1},
    {"text": "User logged out normally.", "label": 0},
]

# Split into training and evaluation sets
train_data, eval_data = train_test_split(data, test_size=0.5, random_state=42)

train_dataset = Dataset.from_list(train_data)
eval_dataset = Dataset.from_list(eval_data)
```

Step 3: Tokenize the Text

```
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")

def tokenize(batch):
    return tokenizer(batch["text"], padding=True, truncation=True)

train_dataset = train_dataset.map(tokenize, batched=True)
eval_dataset = eval_dataset.map(tokenize, batched=True)
```

Step 4: Load Pre-Trained Model

```
# Check for GPU availability
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

model = AutoModelForSequenceClassification.from_pretrained("distilbert-base-
uncased", num_labels=2)
model.to(device)
```

Step 5: Define Training Arguments and Trainer

```
training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=3,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    eval_strategy="epoch",
    logging_dir="./logs",
    logging_steps=10,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
)
```

Step 6: Fine-Tune the Model

```
trainer.train()
```

Step 7: Evaluate the Model

```
predictions = trainer.predict(eval_dataset)
preds = predictions.predictions.argmax(-1)
print("Predicted Labels:", preds)
print("True Labels:", eval_dataset["label"])
```

Example Output:

Predicted Labels: [1 0]
True Labels: [1 0]

The fine-tuned model correctly classified both examples.

Step 8: Test with New Data

```
text = "Suspicious activity detected on admin account."
inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True)

# inputs is a dict (as in Hugging Face Transformers) transfer to GPU/CPU
inputs = {k: v.to(device) for k, v in inputs.items()}

outputs = model(**inputs)
prediction = torch.argmax(outputs.logits).item()

print("Prediction:", "Threat" if prediction == 1 else "Benign")
```

Output:

```
Prediction: Threat
```

4.8 Enhancing LLMs with User Data: Retrieval-Augmented Generation (RAG)

While Large Language Models (LLMs) have vast general knowledge from their training data, they are inherently unaware of private, proprietary, or real-time information. **Retrieval-Augmented Generation (RAG)** is a powerful technique that addresses this limitation by connecting an LLM to an external, up-to-date knowledge base.

In essence, RAG gives the LLM "notes" or "source material" to consult before answering a question. This allows it to provide accurate, relevant responses based on specific user data without the need to retrain the entire model. It's the difference between a closed-book exam (standard LLM) and an open-book exam (RAG).

4.8a How RAG Works

The RAG process combines a retrieval system with the generative power of an LLM. It involves a few key steps:

1. **Indexing**: First, your collection of user data (e.g., PDFs, documents, database entries) is processed and stored in a specialized database, often a **vector database**. Each piece of information is converted into a numerical representation called an **embedding**.

2. **Retrieval**: When a user asks a question, the system converts the query into an embedding and uses **semantic search** to find the most relevant chunks of information from the vector database. It's searching for contextual similarity, not just keyword matches.
3. **Augmentation**: The relevant information retrieved from the database is then combined with the user's original question. This combined text is inserted into a new, detailed prompt that is sent to the LLM.
4. **Generation**: The LLM receives the augmented prompt, which now contains both the question and the necessary context. It uses this provided information to generate a precise and factually grounded answer.

4.8b Example: Customer Support Chatbot

Imagine a company that makes a drone called the "Starlight X1." A standard LLM wouldn't know the specific details of this product. By implementing RAG, the company can create a highly effective support chatbot.

- **Knowledge Base**: The company's internal documents, including the Starlight X1 user manual, troubleshooting guides, and return policies, are indexed in a vector database.
- **User Query**: A customer asks the chatbot, "How do I calibrate the gimbal on my Starlight X1?"
- **RAG in Action**:
    1. **Retrieve**: The system searches the database for content related to "Starlight X1 gimbal calibration" and finds the exact paragraph from the user manual detailing the process.
    2. **Augment**: It creates a new prompt for the LLM that looks something like this:

        Context from the manual: "To calibrate the Starlight X1 gimbal, first place the drone on a flat, level surface. Power it on and connect to the controller. In the Starlight Navigator app, go to Settings > Gimbal Settings and tap 'Calibrate Gimbal'. The process takes approximately 30 seconds. Do not move the drone during calibration."

        User Question: "How do I calibrate the gimbal on my Starlight X1?"

        Instruction: "Based only on the context provided, answer the user's question."

    3. **Generate**: The LLM processes this augmented prompt and generates a perfect, step-by-step answer based *only* on the provided manual text, ensuring the information is accurate and specific to the user's product.

Diagram4-4 illustrates the step-by-step interactions between RAG and LLM to answer the question: “How do I calibrate the gimbal on my Starlight X1?"

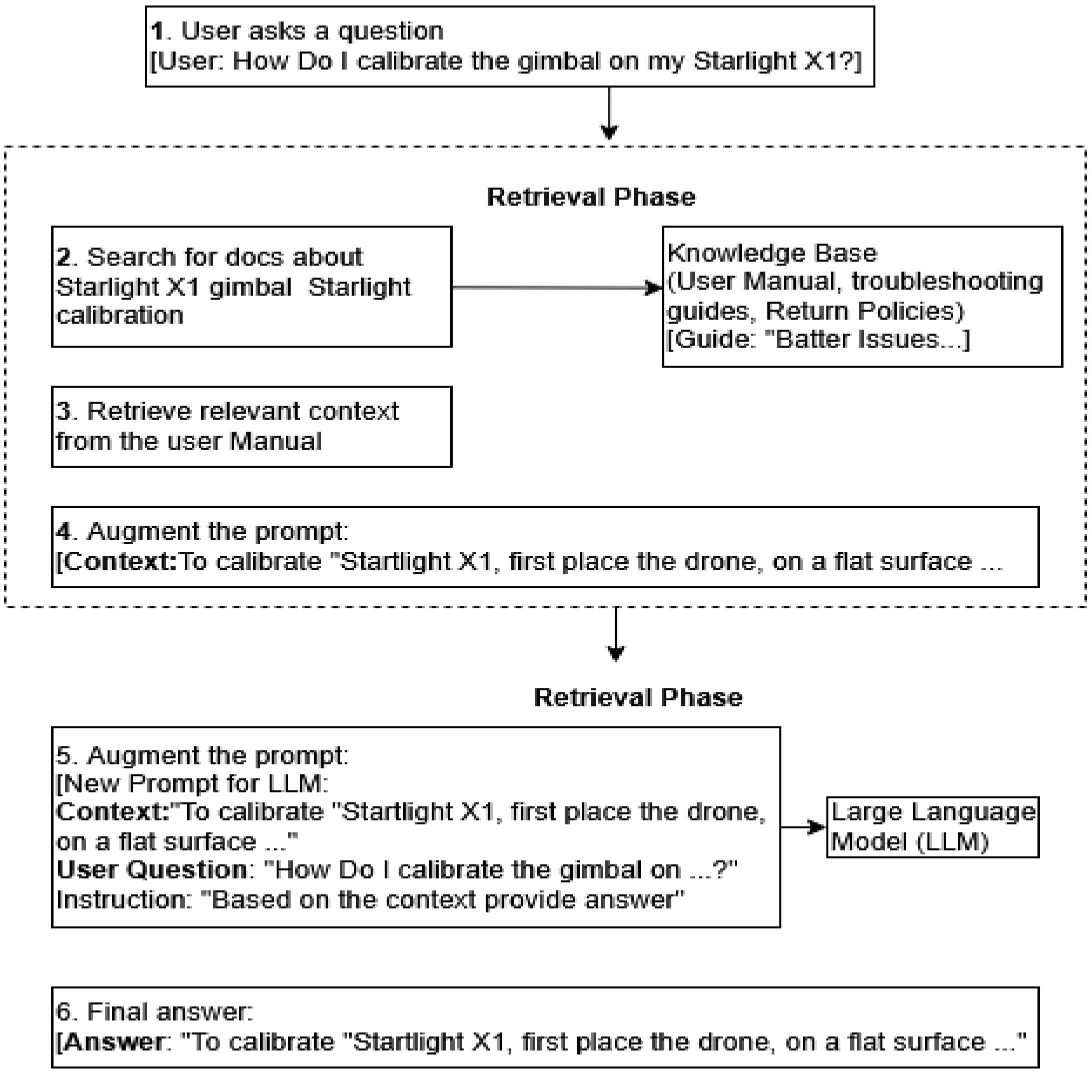

1. User asks a question
[User: How Do I calibrate the gimbal on my Starlight X1?]
Retrieval Phase
2. Search for docs about Starlight X1 gimbal Starlight calibration
Knowledge Base (User Manual, troubleshooting guides, Return Policies) [Guide: "Batter Issues...]
3. Retrieve relevant context from the user Manual
4. Augment the prompt: [**Context:**To calibrate "Startlight X1, first place the drone, on a flat surface ...
Retrieval Phase
5. Augment the prompt: [New Prompt for LLM: **Context:**"To calibrate "Startlight X1, first place the drone, on a flat surface ..." **User Question**: "How Do I calibrate the gimbal on ...?" Instruction: "Based on the context provide answer"
Large Language Model (LLM)
6. Final answer: [**Answer**: "To calibrate "Startlight X1, first place the drone, on a flat surface ..."

Figure4- 4

Module 4: Learning Materials & References

a) Large Language Models explained briefly
b) Large Language Models (LLMs) - Everything You NEED To Know (Video)
c) Hugging Face Quick Start Fine-tune a model
d) Fine-tuning GPT models Open AI
e) Fine-tune a LLM and Use it with Ollama (Video)
f) Introduction to google Colab Video
g) What is Retrieval-Augmented Generation (RAG)?

Assessment: Module 4: Training and Fine-Tuning of LLMs

Part A: Quiz

Multiple choice

4.1a. What is the primary characteristic that defines a Foundational Model?

A) It is trained exclusively using the Next-Token Prediction (NTP) objective.
B) It is a small-scale model adapted only for a single, specific task.
C) It is a large-scale model pre-trained on a massive, broad dataset that can be adapted to a wide range of downstream tasks.
D) It must achieve 100% accuracy on sentiment analysis tasks before deployment.

4.2a. How does the context used in Masked Language Modeling (MLM) differ from that used in Next-Token Prediction (NTP)?

A) MLM uses only words from the right side of the mask, while NTP uses words from the left.
B) MLM uses a unidirectional/causal context (only sees words before), while NTP uses a bidirectional context (sees words before and after).
C) MLM sees words before but not after, whereas NTP sees words after but not before.
D) MLM uses a bidirectional context (sees words before and after), while NTP uses a unidirectional/causal context (only sees words before).

4.3a What key capability does the inclusion of Code repositories (like GitHub) primarily teach a Large Language Model during pre-training?

A) Sheer volume and diversity of language.
B) Reasoning, logic, and structured thinking due to strict syntax and logical flow.
C) Long-range coherence, narrative structure, and high-quality grammar.
D) The ability to rank responses based on human feedback.

4.4a What is the fundamental operation of a neural network that necessitates the use of specialized hardware like GPUs and TPUs during LLM training?

A) Matrix multiplication.
B) Data deduplication and filtering.
C) Tokenization.
D) Prompt engineering.

4.5a. What is the specific objective of the Next-Token Prediction (NTP) training technique, also known as Causal Language Modeling (CLM)?

A) To predict the original words that were in randomly masked positions in a sentence.
B) To predict the very next word in a given sequence.
C) To perform sentiment analysis on text.
D) To fine-tune a model on a small, high-quality dataset of human-written prompt-response pairs.

4.6a. Which statement accurately describes a key feature of Parameter-Efficient Fine-Tuning (PEFT)?

A) PEFT achieves the highest possible performance for a specific task by updating all model weights.
B) PEFT is a multi-step process involving a Reward Model trained on human rankings.
C) PEFT freezes the vast majority of the pre-trained model's weights and trains only a small number of new, injected parameters (adapters).
D) PEFT is primarily used for zero-shot learning where no weights are updated.

4.7a. Which ethical challenge is directly associated with training LLMs on vast, unfiltered internet data like Common Crawl?

A) The hardware cost for training is reduced drastically due to data volume.
B) The model automatically mitigates misinformation and harmful content.
C) The model suffers from catastrophic forgetting, losing its ability to generate text.
D) The model learns and potentially amplifies societal biases (e.g., regarding gender, race, profession, nationality) that exist on the internet.

4.8a. When adapting a foundational model, what is the fastest and cheapest method that involves designing effective inputs but does *not* involve updating any model weights?

A) Prompt Engineering.
B) Full Fine-Tuning.
C) Parameter-Efficient Fine-Tuning (PEFT).

D) Reinforcement Learning from Human Feedback (RLHF).

4.9a. What is the stated goal of using Reinforcement Learning from Human Feedback (RLHF) during the LLM adaptation process?

A) To drastically reduce memory and compute requirements compared to full fine-tuning.
B) To achieve the highest performance for a specific text classification task.
C) To align the model's behavior with human preferences for safety and helpfulness.
D) To convert text data into numerical tokens for processing.

4.10a. What is a significant risk associated with performing Full Fine-Tuning where all or most of the model's weights are updated?

A) The model will be unable to learn high-quality grammar and narrative structure.
B) The risk of "catastrophic forgetting," where the model loses some of its general abilities.
C) The model will exclusively learn unidirectional context, preventing its use in classification.
D) The computational cost is lower than Prompt Engineering.

4.11a. Which data source is primarily responsible for teaching an LLM long-range coherence, high-quality grammar, and narrative structure during pre-training?

A. The Web (Common Crawl)
B. Human-written prompt-reponse pairs
C. Code (e.g., Git Hub)
D. Books (e.g, Google Books, Project Gutenberg)

4.12a. A developer needs to build a model that excels at summarizing legal documents, a task requiring a deep understanding of the entire document's context. Which pre-training objective is fundamentally better suited for this type of understanding-based task?

A. Masked Language Modeling (MLM)
B. Reinforcement Learning from Human Feedback (RLHF)
C. Parameter-Efficient Fine-Tuning (PEFT)
D. Next-Token Prediction (NTP)

4.13a. A user provides a prompt that first describes a task and then includes a complete example of the input and desired output before asking the model to perform the task on new data. What is this technique called?

A. Zero-Shot learning

B. Reinforcement learning
C. Few-Shot learning
D. Full time-tunning
4.14a. What is the primary problem that Retrieval-Augmented Generation (RAG) is designed to solve for Large Language Models (LLMs)?

A. The slow speed at which LLMs generate responses.
B. The LLM's inability to understand complex grammar.
C. The need to permanently add new skills to the LLM's core model.
D. The LLM's lack of access to private, real-time, or specific external knowledge.
Answer D:

4.15a. In the RAG workflow, what is the main purpose of the 'retrieval' step?

A. To check the LLM's generated answer for factual accuracy.
B. To find and pull the most relevant pieces of information from a knowledge base based on the user's query.
C. To generate the final, human-readable answer to the user's question.
D. To convert the entire knowledge base into a single text file.

Answer B:

4.16a. A developer is building a RAG system using a collection of company policy documents. What technology is most commonly used to enable the 'retrieval' of relevant information?

A. A vector database that performs semantic similarity searches.
B. A SQL database that searches for exact keyword matches.
C. A fine-tuned LLM that has memorized all the policy documents.
D. A spell-checking and grammar correction API.
Answer A.

Part B: Short Answer

4.1b Compare the core pre-training objectives of MLM and NTP.

4.2b Explain the spectrum of model adaptation techniques and how they differ in approach and complexity.

Part C: Python Exercises

4.1a Fine-Tuning for Cybersecurity Analysis

Lab Exercise: Fine-Tuning a Transformer Model for Cybersecurity Threat Detection
Tools: Python, Hugging Face Transformers, PyTorch

Resources: See section 4.7.2

Objective 1: Use a labeled dataset of cybersecurity texts to fine-tune a BERT model that classifies texts as “benign” or “threat.”

Text	**Label**
"Multiple failed login attempts from unknown IP address."	Threat
"File transfer completed successfully."	Benign
"Detected port scanning from external host."	Threat
"User logged out normally."	Benign

Label Encoding:
0 = Benign
1 = Threat
Test with New Data
text = "Suspicious activity detected on admin account."

Output:
Prediction: Threat

Objective 2: Modify program to make sure we return benign for the following text:

1. Server rebooted as part of weekly patch cycle.
2. Disk cleanup completed and 2 GB of space freed.
3. User logged in successfully from known device."
4. Password changed successfully by user.
5. Database query executed without error.
6. Application health check returned status OK.

4.1b Fine-Tuning for Sentiment Analysis

Tool: Google Colab (to ensure access to a free GPU). Estimated training time without GPU ~ 1 hour, with GPU < 5 minutes.

Objective: Fine-tune the small distilbert-base-uncased model on the imdb movie review dataset for sentiment classification using colab. The notebook should contain all the code, with markdown cells explaining each step.

Resources:
Students may use AI tools to assist with this exercise. However, proper attribution is required, including the AI engine used, any prompts entered, and a brief description of how the tool contributed to the work. Module 4: Learning Materials & References Reference e) Introduction to google Colab Video

Lab Outline (as seen in the Colab Notebook):

1. **Setup:**
    - Install necessary Hugging Face libraries: transformers datasets evaluate torch.
    - pip install transformers datasets torch scikit-learn
2. **Load the Dataset:**
    - Load the imdb dataset from the Hugging Face Hub.
    - Explore its structure (train and test splits, text and label columns).
    - from datasets import load_dataset; imdb = load_dataset("imdb",split='train[:10%]') only 10% of the database for speed.
3. **Load Pre-trained Model and Tokenizer:**
    - Choose a small, efficient model like distilbert-base-uncased.
    - Load the model and its associated tokenizer. The tokenizer converts text into numbers the model can understand.
    - from transformers import AutoTokenizer, AutoModelForSequenceClassification
    - tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
    - model = AutoModelForSequenceClassification.from_pretrained("distilbert-base-uncased", num_labels=2)
4. **Preprocess the Data:**
    - Write a function to tokenize the text.
    - Use the .map() method to apply this function to the entire dataset. This is done in batches for efficiency.
    - def tokenize_function(examples):
    -   """Tokenizes the text data."""
    -   return tokenizer(examples['text'], padding='max_length', truncation=True)def preprocess_function(examples): return tokenizer(examples["text"], truncation=True)
    - train_dataset = train_dataset.map(tokenize_function, batched=True)
    - test_dataset = test_dataset.map(tokenize_function, batched=True)
5. **Fine-Tuning:**
    - Define TrainingArguments to specify hyperparameters like learning rate, number of epochs, and batch size.

- Instantiate the Trainer, which is a high-level Hugging Face class that handles the entire training loop.
- from transformers import TrainingArguments, Trainer
- training_args = TrainingArguments(output_dir="./results", eval_strategy="epoch")
- Start the training process with trainer.train(). This is the main event!

6. **Evaluate and Infer:**
    - After training, the Trainer automatically evaluates the model on the test set.
    - text = "This was a masterpiece. The acting, the score, everything was perfect."
    - inputs = tokenizer(text, return_tensors="pt")
    - outputs = model(**inputs)
    - logits = outputs.logits
    - predicted_class_id = logits.argmax().item() -> This will show the predicted class (0 for negative, 1 for positive).

4.1c Fine-tune a LLM and Use it with Ollama

Replicate the python exercise mentioned Module 4: Learning Materials & References Reference d) Fine-tune a LLM and Use it with Ollama (Video)

Objective: Use a different dataset type of Electric Vehicles instead of computer products with the following attributes: name, range, MSRP, model, manufacturer.

Example:

name: Ioniq 5 range:303 MSRP: $45,850 model: SEL RWD manufacturer: Hyundai	name: Chevrolet Equinox EV range:312 MSRP: $41,000 model:FWD, manufacturer: GM

Figure4- 5

Randomly select 25 entries from the following reference:
https://coltura.org/electric-car-battery-range/

4.1d Mini-RAG.

This exercise demonstrates the core Retrieval-Augmented Generation workflow using scikit-learn for retrieval and a Hugging Face transformers model for question-answering. It's a simplified approach that runs on your local machine without needing API keys or a dedicated vector database.

Goal: Build a simple RAG System to Answer Questions About Planets

In this example, a simple "knowledge base" with facts about planets is created. Then, implement an RAG pipeline that can retrieve the relevant fact and use it to answer a specific question.

1. Prerequisites

First, you'll need to install the necessary libraries.
pip install scikit-learn transformers torch

2. The RAG Exercise Code

Copy and paste the following code into a Python file (e.g., rag_exercise.ipynb) and run it.

```python
#2. The RAG Exercise Code

import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from transformers import pipeline

# --- Step 1: Create the Knowledge Base ---
# This is our collection of documents that the RAG system can pull from.
knowledge_base = [
    "Mars is the fourth planet from the Sun and is known as the Red Planet.",
    "Jupiter is the largest planet in our solar system and has a Great Red Spot, a giant
storm.",
    "Saturn is the sixth planet from the Sun, famous for its extensive ring system made of ice
particles.",
    "The Earth is the third planet from the Sun and the only known planet to support life.",
    "Venus is the second planet from the Sun and is the hottest planet in the solar system.",
    "Mercury is the smallest planet in our solar system and the closest to the Sun."
]
print("Knowledge Base created.")

# --- Step 2: Build the Retriever ---
# The retriever's job is to find the most relevant document from the knowledge base.
# We'll use a simple TF-IDF vectorizer and cosine similarity.

# Initialize the vectorizer
vectorizer = TfidfVectorizer()

# Vectorize the knowledge base
knowledge_base_vectors = vectorizer.fit_transform(knowledge_base)
print("Knowledge Base vectorized.")
```

```python
def retrieve_context(query):
    """
    Retrieves the most relevant document from the knowledge base based on the query.
    """
    # Vectorize the user's query
    query_vector = vectorizer.transform([query])

    # Calculate cosine similarity between the query and all documents
    similarities = cosine_similarity(query_vector, knowledge_base_vectors).flatten()

    # Find the index of the most similar document
    most_similar_index = np.argmax(similarities)

    # Return the most relevant document
    return knowledge_base[most_similar_index]

print("Retriever function is ready.")
print("Depending on your computer, this will take a few moments to load the model.")
# --- Step 3: Initialize the Generator ---
# We'll use a pre-trained Question-Answering model from Hugging Face.
# This model is designed to find an answer to a question within a given context.
generator = pipeline("question-answering", model="distilbert-base-cased-distilled-squad")

print("Generator (QA model) loaded.")
# --- Step 4: Create the RAG Workflow ---
def ask_question_with_rag(query):
    """
    The main RAG function that combines retrieval and generation.
    """
    print(f"\n Query: {query}")

    # 1. Retrieve the most relevant context from the knowledge base.
    retrieved_context = retrieve_context(query)
    print(f"Retrieved Context: \"{retrieved_context}\"")

    # 2. Generate the answer by feeding the context and query to the QA model.
    result = generator(question=query, context=retrieved_context)

    # 3. Return the answer.
    answer = result['answer']
    print(f"Answer: {answer}")
    return answer
# --- Step 5: Test the RAG System ---
```

```
ask_question_with_rag("Which planet has rings?")
ask_question_with_rag("What is the name of the largest planet?")
ask_question_with_rag("Which planet is called the Red Planet?")
```

Sample output:

```
Query: Which planet has rings?
Retrieved Context: "Jupiter is the largest planet in our solar system and has a Great Red Spot, a giant storm."
Answer: Jupiter

Query: What is the name of the largest planet?
Retrieved Context: "Venus is the second planet from the Sun and is the hottest planet in the solar system."
Answer: Venus

Query: Which planet is called the Red Planet?
Retrieved Context: "Mars is the fourth planet from the Sun and is known as the Red Planet."
Answer: Mars
```

4.1e Mini-RAG with PDF knowledge base

Repeat exercise 4.1d but using a PDF file for knowledge base. Create a PDF file **planets.pdf** with the same information as before.

1. Setup: Create Your PDF Knowledge Base

Text for your PDF:

The solar system is a gravitationally bound system of the Sun and the objects that orbit it.

Mercury is the smallest planet and is closest to the Sun. It has no moons.

Venus is the second planet from the Sun and is the hottest, with surface temperatures hot enough to melt lead. It is sometimes called Earth's "sister planet" due to their similar size and composition.

Earth is the third planet from the Sun and the only place known to harbor life. It has one moon.

Mars, known as the Red Planet, is the fourth planet. It has a thin atmosphere and two small moons, Phobos and Deimos.

Jupiter is a gas giant and the largest planet in the solar system. Its Great Red Spot is a massive storm that has raged for centuries.

Saturn is the sixth planet, famous for its beautiful and extensive ring system composed mainly of ice particles.

Uranus and Neptune are the outermost planets, known as ice giants.

2. Prerequisites

pip install pypdf scikit-learn transformers torch

3. The RAG Exercise Code

```
import numpy as np
import pypdf
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from transformers import pipeline
```

```python
# --- Step 1: Load and Process the PDF Knowledge Base ---
# This function reads the PDF and splits the text into manageable chunks.

def load_and_chunk_pdf(file_path):
    """
    Reads a PDF, extracts text, and splits it into paragraphs.
    """
    pdf_reader = pypdf.PdfReader(file_path)
    full_text = ""
    for page in pdf_reader.pages:
        full_text += page.extract_text() + "\n"

    # Split the text into chunks (paragraphs in this case)
    chunks = [chunk.strip() for chunk in full_text.split('\n\n') if chunk.strip()]
    return chunks

# Load the knowledge from our PDF
knowledge_base = load_and_chunk_pdf("planets.pdf")
print(f"Loaded {len(knowledge_base)} chunks from the PDF.")

# --- Step 2: Build the Retriever ---
# (This part is the same as the previous exercise)
vectorizer = TfidfVectorizer()
knowledge_base_vectors = vectorizer.fit_transform(knowledge_base)
print("Knowledge Base vectorized.")

def retrieve_context(query):
    query_vector = vectorizer.transform([query])
    similarities = cosine_similarity(query_vector, knowledge_base_vectors).flatten()
    most_similar_index = np.argmax(similarities)
    return knowledge_base[most_similar_index]

print("Retriever function is ready.")

# --- Step 3: Initialize the Generator ---
# (This part is the same as the previous exercise)
generator = pipeline("question-answering", model="distilbert-base-cased-distilled-squad")
print("Generator (QA model) loaded.")

# --- Step 4: Create the RAG Workflow ---
def ask_question_with_rag(query):
    print(f"\n Query: {query}")
    # 1. Retrieve the most relevant context.
    retrieved_context = retrieve_context(query)
    print(f"Retrieved Context: \"{retrieved_context}\"")

    # 2. Generate the answer.
    result = generator(question=query, context=retrieved_context)
    answer = result['answer']
    print(f"Answer: {answer}")
    return answer

# --- Step 5: Test the RAG System ---
ask_question_with_rag("Which planet has a Great Red Spot?")
ask_question_with_rag("What is the hottest planet?")
ask_question_with_rag("Which planet is known as the Red Planet?")
```

4. Expected Output

```
Loaded 8 chunks from the PDF.
Knowledge Base vectorized.
Retriever function is ready.
Generator (QA model) loaded.
Query: Which planet has a Great Red Spot?

Retrieved Context: "Jupiter is a gas giant and the largest planet in the solar system. Its Great Red Spot is a massive storm
that has raged for centuries."

Answer: Jupiter

Query: What is the hottest planet?

Retrieved Context: "Venus is the second planet from the Sun and is the hottest, with surface temperatures hot enough to
melt lead. It is sometimes called Earth's "sister planet" due to their similar size and composition."

Answer: Venus

Query: Which planet is known as the Red Planet?

Retrieved Context: "Mars, known as the Red Planet, is the fourth planet. It has a thin atmosphere and two small moons,
Phobos and Deimos."

Answer: Mars
```

4.1f Mini-RAG using a custom PDF knowledge base

Repeat Exercise 4.1e with a knowledge base of your choosing, demonstrating your ability to adapt the procedure to a different informational source.

Module 5: Applications of LLMs

Welcome to Module 5! In this module, we'll explore the powerful and diverse applications of Large Language Models. We'll divide our exploration into two main categories: how LLMs **create new content** (generative applications) and how they **understand and process existing information** (comprehension applications).

5.1 Generative Applications

Using LLMs to create new content: Generative applications are all about using LLMs as creative partners or powerful assistants to produce original text. The model isn't just finding information; it's synthesizing and generating something entirely new based on your instructions.

5.1a Text Generation:

This is the most direct application. By providing a prompt, you can guide an LLM to write in virtually any style for any purpose.

- **Creative Writing:** You can ask for a poem in the style of Shakespeare, a short story about a robot detective, or dialogue for a play.
- **Marketing Copy:** Generate catchy slogans, product descriptions, social media posts, or entire advertising campaigns. For example, "Write three taglines for a new brand of sustainable coffee."
- **Email Drafting:** Quickly compose professional emails, friendly follow-ups, or even difficult responses by simply stating your goal (e.g., "Draft a polite email declining a meeting invitation for Tuesday").

5.1b Summarization:

LLMs are incredibly skilled at distilling long pieces of text into concise, easy-to-digest summaries. This is a huge time-saver for reading dense reports, articles, or research papers. There are two main approaches:

- **Extractive Summarization:** This is the "highlighter" method.[6] The model identifies the most important sentences and phrases from the original text and pulls them out verbatim to form a summary. It doesn't create new sentences.
- **Abstractive Summarization:** This is a more advanced, human-like approach. The LLM reads and "understands" the entire text and then generates a **brand-**

new summary in its own words. This often results in more fluent and natural-sounding summaries that capture the core essence of the original document. Modern LLMs primarily use this method.

5.1c Machine Translation:

LLMs have revolutionized machine translation, moving far beyond the rigid, literal translations of the past.

- **The Leap from Statistical Methods:** Older systems translated phrase by phrase, often missing nuance and leading to grammatically awkward results.
- **Fluent, Context-Aware Translation:** Modern LLMs, like those powering Google Translate, analyze the entire sentence and its surrounding context. This allows them to understand idioms, cultural nuances, and grammatical structures, resulting in translations that are remarkably fluent and accurate. For example, an LLM understands that "breaking the ice" doesn't literally involve ice.

5.2 Comprehension and Retrieval Applications

Using LLMs to extract information and understand text: These applications focus on an LLM's ability to "read" and process text to find specific information, categorize it, or structure it in a useful way. It's about turning unstructured text into knowledge and data.

5.2a.Question Answering (QA):

This is the ability to answer specific questions based on a provided piece of text (a "context"). Think of it like an open-book test where the LLM can only use the document you give it to find the answer. This is extremely useful for customer support bots that answer questions from a knowledge base or for analysts trying to find facts in a long report.

5.2b. Text Classification:

This involves assigning a label or category to a piece of text. It's a powerful tool for organizing and making sense of large volumes of text data automatically.

- **Sentiment Analysis:** Is this customer review **positive, negative, or neutral**? This helps businesses quickly gauge customer satisfaction. 👍 👎 😐
- **Topic Classification:** Does this article belong in the **sports, politics, or technology** section? This is how news websites automatically categorize content.

- **Spam Detection:** Is this incoming email **spam or not**? Email services use this to keep your inbox clean.

- **Information Extraction:** This is the process of identifying and pulling out specific pieces of structured information from unstructured text. It turns a block of prose into a database-like format.
    - **Key Entities:** These are things like people's names, organizations, locations, dates, and monetary values.
    - **Relationships:** The LLM can also understand how these entities relate. For example, in the sentence, "*Dr. Anya Sharma, a researcher at OmniCorp, published her findings on October 17, 2025.*", an LLM can extract:
        - **Person:** Dr. Anya Sharma
        - **Organization:** OmniCorp
        - **Role:** Researcher
        - **Date:** October 17, 2025
        - **Relationship:** Dr. Anya Sharma *works for* OmniCorp.

Module 5: Learning Materials & References

h) The 5 Types of LLM Apps
i) How to Use LLMs for Marketing Strategies
j) AI Email Writer
k) Evaluate the text summarization capabilities of LLMs for enhanced decision-making on AWS
l) How LLMs Are Changing the Translation Industry | Interview (2025)

Assessment: Module 5: Applications of LLMs

Part A: Quiz

Multiple choice

5.1a Which of the following is the best example of a generative application of an LLM?

A. Extracting a company's name from a news article.
B. Answering a question based on a provided paragraph.

C. Composing a short poem about autumn.
D. Identifying the sentiment of a customer review.

5.2a A summarization method that involves an LLM reading a document and then writing a new summary in its own words is called:

A. Extractive Summarization
B. Abstractive Summarization
C. Translative Summarization
D. Compressive Summarization

5.3a. What is the primary advantage of modern LLM-based machine translation over older statistical methods?

A. It is significantly faster at translating individual words.
B. It can handle a much larger vocabulary.
C. It requires less computing power to operate.
D. It understands the context of the entire sentence, leading to more fluent and accurate translations.

5.4a. Which of the following tasks falls under the category of Text Classification?

A. Detecting if an incoming email is spam or not.
B. Identifying all the dates mentioned in a document.
C. Creating a one-paragraph summary of a report.
D. Writing an email to a colleague.

5.5a. A chatbot that uses a company's internal documents to answer customer questions primarily using which LLM application?

A. Creative Writing
B. Sentiment Analysis
C. Question Answering (QA)
D. Machine Translation

5.6a. In the sentence, 'Dr. Isla Nevarez from the Institute of AI presented her findings in Berlin,' what would an Information Extraction model identify as the 'Organization'?

A. Dr. Isla Nevarez
B. Institute of AI
C. Berlin
D. Findings

5.7a. Which of these is a comprehension application of LLMs?

A. Drafting marketing copy for a new product.
B. Translating a user manual from Japanese to English.
C. Writing a sequel to a famous novel.
D. Categorizing customer support tickets by topic.

5.8a. True or False: Extractive summarization creates new sentences to summarize a text.

False
True

5.9a. Sentiment analysis is a subfield of which broader LLM application?

A. Text Classification
B. Text Generation
C. Information Extraction
D. Machine Translation

5.10a. Asking an LLM to 'Write three different taglines for a new coffee brand' is an example of what?

Part B: Short Answer

5.1b. The Three-Tier Summary

i. **Find a News Article:** Choose any recent news article from a reputable source (e.g., BBC, Telegraph (UK), Reuters, Fox News, The Washington Times, The New York Times) that is at least five paragraphs long.

ii. **Use an LLM:** Copy the text of the article and paste it into an LLM interface (like Gemini).

iii. **Generate Three Summaries:** Use the following prompts (or similar ones) to generate three different summaries of the same article:
   1. **Short Summary:** "Summarize this article in a single sentence." or "Give me a TL;DR of this."
   2. **Medium Summary:** "Write a concise, one-paragraph summary of this article."
   3. **Detailed Summary:** "Provide a detailed summary of this article using bullet points to highlight the key information."

iv. **Write short answer in your own words:** how LLM adjusts the level of detail while retaining the core information for each request.

5.2b Become a Prompt Engineer for QA

1. **Use the Provided Text:**
   Below is a short passage. Your task is to use this as the context for a Question-Answering session with an LLM.
   The Voyager 1 space probe, launched by NASA on September 5, 1977, from Cape Canaveral, Florida, is the most distant human-made object from Earth. Its primary mission was to study the outer planets Jupiter and Saturn. After completing its planetary flybys in 1980, its mission was extended. It is now on a journey into interstellar space, carrying a golden record containing sounds and images selected to portray the diversity of life and culture on Earth, intended for any intelligent extraterrestrial life that may find it. The project was managed for NASA by the Jet Propulsion Laboratory (JPL) in California.

2. **Formulate Effective Prompts:**

   Your goal is to write at least five different questions (prompts) to make an LLM extract specific information from this text. Paste the text and your question into the LLM. Your questions should test different retrieval skills.
   a) **Simple Question:** such as, "When was the Voyager 1 probe launched?".,Write your own simple question.

   b) **Relationship Question:** i.e., "Which organization ... or NASA?"

   c) **Purpose Question:** "What was the purpose ... ?"

   d) **Try Your Own:** Think of other questions about the launch location, its original mission, and its current status. Notice how you must phrase your questions precisely to get the right answer from the provided context.

   e) What conclusion can be drawn about the relationship between prompt complexity and the quality of AI-generated results?

Part C: Python Exercises

5.1c: Automated Email Drafter

Write a Python function that generates personalized emails by populating pre-defined templates with specific data. This mimics how you would instruct an LLM to perform a structured text generation task like drafting an email. Do not use LLM in exercise.

Setup:

Here are three dictionaries, each containing the specific details for a different type of email.

```python
import datetime

# --- SETUP: DATA DICTIONARIES ---
marketing_data = {
    "customer_name": "Alex",
    "product_name": "Nexus Pro Keyboard",
    "discount": "25%",
    "sale_end_date": "October 26, 2025"
}

bug_report_data = {
    "username": "Casey",
    "ticket_id": "7B-4821-9F0A",
    "bug_summary": "User profile page fails to load on mobile.",
    "support_agent": "Jane"
}

meeting_invite_data = {
    "guest_name": "Dr. Evans",
    "meeting_date": "October 23, 2025",
    "meeting_time": "2:00 PM PDT",
    "topic": "Q4 Project Sync",
    "sender_name": "Pat"
}
```

Your Tasks:

Create Email Templates: Create three multi-line string variables to serve as templates for your emails. Use f-string placeholders (e.g., `{customer_name}`) to mark where the data from the dictionaries will go.

- `marketing_template`
- `bug_report_template`
- `meeting_invite_template`

Hint: Use triple quotes (`"""..."""`) to create strings that span multiple lines easily.

Example start for the marketing template: `marketing_template = """Subject: Big News! A {discount} discount just for you!`

Write the Generator Function: Create a function called `draft_email(email_type, data)`.

Test Your Function: Call your draft_email function for each of the three scenarios (marketing, bug report, and meeting invite) using the provided data dictionaries. Print the result of each call to see your generated emails.

Sample output:

```
--- MARKETING EMAIL ---
Subject: Big News! A 25% discount just for you!

Hi Alex,

We have an exciting offer for you! For a limited time, get 25% off the amazing Nexus Pro Keyboard.
This special promotion ends on October 26, 2025, so don't miss out!

Best,
The Marketing Team

========================================

--- BUG REPORT EMAIL ---

Subject: Update on your bug report (Ticket ID: 7B-4821-9F0A)

Hello Casey,
This email is to confirm that we have received your bug report regarding: "User profile page fails to load on mobile.".
Our team is now looking into the issue. Your ticket ID is 7B-4821-9F0A.
Thank you for helping us improve our service.


Sincerely,

Jane

Customer Support

========================================

--- MEETING INVITE EMAIL ---
Subject: Meeting Invitation: Q4 Project Sync
Dear Dr. Evans,
I would like to invite you to a meeting to discuss our Q4 Project Sync.
Please let me know if you are available on October 23, 2025 at 2:00 PM PDT.
Looking forward to speaking with you.

Best regards,

Pat
```

Module 6: Ethical, Legal, and Societal Implications of LLMs

This module examines the moral, legal, and social dimensions of large language models — helping readers understand the responsibilities, risks, and frameworks for building and deploying AI responsibly.

6.1 Ethical Challenges in LLMs

This section introduces the core ethical dilemmas presented by Large Language Models (LLMs). Because these models are trained on vast amounts of human-generated text, they inevitably inherit the complexities, biases, and flaws present in that data.

- **Bias and Discrimination in Training Data:** LLMs learn patterns from text scraped from the internet, books, and other sources. This data reflects historical and societal biases related to gender, race, religion, and culture.[16]
    - **Example:** An LLM trained on historical text might associate nurses with women and engineers with men, perpetuating gender stereotypes when generating job descriptions or stories.
- **Fairness and Representational Harms:** Bias in data leads to unfair outcomes. This can manifest in two primary ways:
    - **Allocative Harm:** When an AI system unfairly distributes resources or opportunities. For example, an LLM used to screen résumés might downgrade candidates from underrepresented backgrounds.
    - **Representational Harm:** When an AI system perpetuates negative stereotypes or misrepresents a particular group, reinforcing social stigma. For example, an image model generating stereotypical or demeaning images for certain demographic queries.
- **Hallucinations and Misinformation:** A "hallucination" occurs when an LLM generates text that is plausible and grammatically correct but factually inaccurate or nonsensical. Since the model's goal is to predict the next word, not to state the truth, it can confidently invent facts, sources, and events. This poses a significant risk for the spread of misinformation and propaganda.[17]
- **Environmental Costs:** Training a single large-scale LLM requires immense computational power. This translates to substantial energy consumption and a significant carbon footprint from the data centers that house the hardware. Furthermore, these data centers require vast amounts of water for cooling, raising concerns about environmental sustainability.[18]
- **Transparency and Accountability:** Many LLMs function as "black boxes," meaning even their creators cannot fully explain why the model produced a specific output. This lack of transparency makes it difficult to assign

[16] Large language model — Algorithmic bias
[17] Large language model — Hallucination
[18] The Uneven Distribution of AI's Environmental Impacts

accountability when an LLM causes harm. Is the developer, the deploying company, or the end-user responsible for its output? Establishing clear lines of responsibility is a major ethical and legal challenge.[19]

6.2 Privacy and Security Considerations

This lesson explores how the data-intensive nature of LLMs creates unique privacy and security vulnerabilities that must be managed to protect users and their information.

- **Data Collection and User Consent:** LLMs are trained on public data, but they are also often fine-tuned on user conversations and inputs to improve performance. This raises critical questions about informed consent. Do users truly understand how their data is being used, stored, and leveraged to train future models?
- **Risks of Data Leakage and Memorization:** LLMs can inadvertently "memorize" and store sensitive information from their training data, including personally identifiable information (PII) like names, phone numbers, and addresses. A carefully crafted prompt could cause the model to regurgitate this private data in response, leading to a serious data breach.[20]
- **Prompt Injection and Adversarial Attacks:** These are security vulnerabilities specific to LLMs.
    - **Prompt Injection:** A malicious user crafts a prompt to trick the LLM into ignoring its original instructions and performing an unintended action, such as revealing its system prompt or generating harmful content.[21]
    - **Adversarial Attacks:** A user makes subtle, often imperceptible changes to an input to cause the model to misclassify or misunderstand it, leading to incorrect or dangerous outputs.
- **Privacy-Preserving Techniques:** To mitigate these risks, researchers have developed several techniques:
    - **Differential Privacy:** This method adds a mathematically controlled amount of statistical "noise" to the training data. This noise is small enough to allow the model to learn broad patterns but large enough to make it impossible to identify any single individual's data within the dataset.
    - **Federated Learning:** A decentralized training approach where the model is trained on user data directly on their local devices (e.g., smartphones). Instead of sending raw data to a central server, only the model updates (gradients) are sent back, keeping user data private.
- **Case Studies:** Examining real-world incidents, such as when user chat histories from an AI service were inadvertently exposed, highlights the critical need for

[19] The Blackbox Problem
[20] Large language model — Copyright and content memorization
[21] Large language model — Prompt Injection

robust security and privacy protocols in all stages of LLM development and deployment.

6.3 Legal and Regulatory Frameworks

As AI becomes more integrated into society, governments and international bodies are working to establish legal frameworks to govern its development and use. This lesson provides an overview of the emerging regulatory landscape.

- **Overview of AI Regulations:**
  - **EU AI Act:** A landmark proposal that takes a risk-based approach, categorizing AI systems into four tiers (unacceptable, high, limited, and minimal risk), with stricter regulations for higher-risk applications.
  - **U.S. Executive Orders:** The U.S. has focused on establishing principles for "safe, secure, and trustworthy AI," emphasizing security standards, fairness evaluations, and transparency.
  - **OECD AI Principles:** These influential, non-binding principles promote human-centered, transparent, robust, and accountable AI. They serve as a guideline for many national AI strategies.
- **Copyright Issues in LLM Training Datasets:** A central legal battle revolves around whether training LLMs on copyrighted material (books, articles, art) constitutes "fair use."Content creators argue it is infringement, while AI companies argue it is a transformative use necessary for innovation. The outcomes of these court cases will shape the future of generative AI.[22]
- **Intellectual Property and Generative Content Ownership:** Who owns the output of an LLM? The user who wrote the prompt? The company that created the model? Or is the work in the public domain? Current laws were not designed for AI-generated content, and legal precedent is still being established.
- **Data Governance and Compliance:** Regulations like Europe's **GDPR** (General Data Protection Regulation) and the **CCPA** (California Consumer Privacy Act) impose strict rules on how personal data is collected, processed, and stored. Companies deploying LLMs must ensure they comply with these rules, especially regarding user data.
- **Corporate Responsibility and Auditability:** Companies are increasingly expected to demonstrate that their AI systems are fair, safe, and compliant. This requires establishing internal governance structures, conducting regular audits for bias and performance, and maintaining documentation (like "model cards") that describes a model's capabilities and limitations.

[22] Industry Today: AI Training Data — The Copyright Controversy

6.4 Societal Impacts

The widespread adoption of LLMs is poised to transform society in profound ways, affecting jobs, education, and social trust. This lesson examines these broader impacts.

- **Impact on Employment and Automation:** LLMs can automate tasks involving writing, summarizing, and coding, which could displace workers in certain fields. However, they also create new roles (e.g., prompt engineer, AI ethics officer) and act as powerful tools that **augment** human capabilities, potentially increasing productivity and enabling new forms of creativity.
- **Education and Misinformation Risks:** In education, LLMs present challenges to academic integrity (e.g., plagiarism) but also offer opportunities for personalized learning. A greater societal challenge is teaching "AI literacy"—the ability to critically evaluate AI-generated content, identify potential misinformation, and use these tools responsibly.
- **Social Trust and Public Perception of AI:** Public trust in AI is fragile. High-profile failures, biased outputs, or privacy breaches can erode confidence, while successful and beneficial applications can build it. Maintaining this trust requires transparency, reliability, and clear communication from developers and policymakers.
- **Ethical Design and Human-in-the-Loop Systems: Ethical design** involves embedding ethical considerations into the AI development process from the very beginning. A key component of this is creating **human-in-the-loop (HITL)** systems. In a HITL system, a human retains oversight and has the final say in critical decisions, especially in high-stakes domains like medicine, finance, and law. This ensures that autonomous systems remain accountable to human judgment.
- **Role of Open Research and Transparency:** The debate continues over whether to open-source powerful LLMs. Open access allows for greater public scrutiny, faster innovation, and accountability. However, it also carries the risk that malicious actors could misuse the technology for harmful purposes like creating large-scale propaganda campaigns.

6.5 Toward Responsible AI Development

This final section outlines the principles, practices, and future directions for building AI systems that are safe, fair, and aligned with human values.

- **AI Ethics Principles:** Much like medical ethics, AI ethics is guided by a set of core principles:
    - **Beneficence:** AI should be used for good and promote human well-being.
    - **Non-maleficence:** AI should not cause harm.[23]
    - **Justice:** The benefits and risks of AI should be distributed fairly and equitably.
    - **Autonomy:** AI should respect human autonomy and decision-making.

[23] Nonmaleficence

- **Explainability:** The operations of an AI system should be understandable to humans.

Module 6: Learning Materials & References

a) Wikipedia: Large language model — section: *Stereotyping, Selection bias, Political bias* Limitations and challenges
b) Wikipedia: Large language model — AI Safety.
c) Wikipedia: Large language model —Hallucination
d) Wikipedia: Large language model —Algorithmic bias
e) The Uneven Distribution of AI's Environmental Impacts
f) The Black Box Problem: Opaque Inner Workings of Large Language Models
g) Large language model — Copyright and content memorization
h) Large language model — Prompt Injection
i) Nonmaleficence

Assessment: Module 6: Applications of LLMs Ethical, Legal, and Societal Implications of LLMs

Part A: Quiz

T/F Multiple choice

6.1a. A model 'hallucination' refers to an LLM intentionally generating creative or fictional content as part of its designed function.

True
False

6.2a. 'Allocative harm' occurs when an AI system perpetuates negative stereotypes or misrepresents a particular group.

True
False

6.3a The EU AI Act is a regulatory framework that applies the same strict rules to all AI systems, regardless of their application.

True
False

6.4a. An AI system used for screening loan applications consistently denies applicants from a specific neighborhood, even though they are qualified. This is a primary example of which ethical failure?

A. Prompt Injection
B. Allocative Harm
C. AI Hallucination
D. Representational Harm

6.5a A company wants to train its LLM using sensitive customer data stored on individual user devices without that data ever leaving the devices. Which privacy-preserving technique should they use?

A. Red-Teaming
B. Explainable AI (XAI)
C. Federated Learning
D. Differential Privacy

6.6a A user crafts a special prompt that tricks an LLM into ignoring its safety guidelines and revealing its confidential system instructions. This type of security vulnerability is known as:

A. Prompt Injection
B. Data Leakage
C. An Adversarial Attack
D. Memorization

6.7a What is the primary goal of Explainable AI (XAI) tools like LIME and SHAP?

A. To secure the model against adversarial attacks.
B. To increase the accuracy and performance of the model.
C. To reduce the environmental cost of training models.
D. To make the model's decision-making process understandable to humans.

6.8a The ethical principle of 'Non-maleficence' in AI development is best described as:

A. The obligation to prevent AI systems from causing harm.
B. The need to respect human decision-making and control.
C. The responsibility to actively do good and benefit humanity.
D. The duty to ensure AI systems are fair and equitable.

6.9a A 'human-in-the-loop' (HITL) system is an ethical design pattern where:

A. Humans provide the initial training data, but the AI operates fully autonomously afterward.
B. An AI is designed to replace human roles entirely to eliminate human error.
C. A human retains oversight and has the ability to intervene or make the final decision.
D. The AI is only used to simulate human behavior for research purposes.

Part B: Short Answer

6.1b The Open-Source Dilemma

An AI company has created a state-of-the-art Large Language Model. They can either open source it, allowing anyone to access and build upon the technology, which promotes transparency and innovation but also allows bad actors to use it for creating misinformation. Alternatively, they can keep it proprietary and closed, controlling its use through an API, which allows for better safety monitoring but concentrates power and reduces public oversight.

Considering the principles of transparency, corporate responsibility, and the societal risk of misinformation, which path presents the greater ethical challenge? Justify your reasoning by outlining the primary risk of your chosen path and why it outweighs the benefits of the alternative.

6.2b The Accountability Problem

An AI-powered financial advisor, used by a human consultant at a bank, recommends a high-risk investment that ultimately fails, causing a client to lose their life savings. The AI's recommendation was based on subtle, biased patterns in its training data that favored riskier products.

Who bears the primary ethical responsibility for the client's financial loss: the AI developers who created the biased model, the bank that deployed the tool, or the human consultant who accepted the AI's recommendation? Explain your choice and discuss how the concept of a "human-in-the-loop" system complicates the assignment of blame in this scenario.

Part C: Activity

6.1c Ethical AI Red-Teaming

Objective: To identify and analyze potential ethical failures in a real-world AI application scenario. In a "red-team" exercise, you proactively search for flaws and vulnerabilities in a system.

Scenario:

A city's transportation department deploys a new AI-powered system called "FlowCity" to manage traffic lights. The AI's goal is to optimize traffic flow and reduce overall commute times. It was trained on traffic data from the past five years. After six months, the city notices two outcomes:

1. Commute times from the wealthy suburbs to the downtown business district have decreased by an average of 15%.
2. Commute times within lower-income neighborhoods, which rely more on public bus routes, have increased by 10% because the AI prioritizes major arterial roads over smaller streets used by buses.

Your Task:

As an ethical review board, analyze the "FlowCity" system. Answer the following questions:

- **Identify the Harms:** What specific type of ethical harm (**allocative** or **representational**) is occurring? Who is being negatively affected?
- **Pinpoint the Cause:** What is the most likely cause of the model's biased performance? (Hint: Consider the training data and the model's primary objective).
- **Assign Responsibility:** Who holds the most responsibility for this negative outcome? The **AI developers**, the **city officials** who deployed it, or both? Explain your reasoning.
- **Propose Solutions:** What are two concrete steps the city could take to mitigate this issue and make the system more equitable? (Hint: Think about the model's goals and "human-in-the-loop" concepts).

6.2c Create a "Model Card"

Objective: To practice creating transparent documentation for an AI model, a key component of responsible AI governance. A **Model Card** is a short, standardized report that explains what an AI model does, its limitations, and how it should be used.

Scenario:

You are on the responsible AI team for a company that has just developed "CritiqueBot," an LLM designed to be integrated into word processors. It provides students with feedback on their essays, checking grammar, style, and the strength of their arguments. It was trained primarily on a large dataset of acclaimed academic and literary essays.

Your Task:

Fill out the following Model Card template for "CritiqueBot." Be specific and think critically about potential blind spots and risks.

Model Card: CritiqueBot v1.0

- **Model Details:**
    - **Developer:** *[Your Company Name]*
    - **Model Version:** 1.0
    - **Model Type:** Generative Large Language Model
- **Intended Use:**
    - *Describe the primary, responsible way for a student to use this tool.*
- **Out-of-Scope Use Cases:**
    - *List at least two ways this tool should NOT be used (e.g., for final grading, writing the essay for the user).*
- **Performance Metrics:**
    - *Briefly describe how you might measure its success (e.g., "Tested against a dataset of professionally graded essays...").*
- **Ethical Considerations & Limitations:**
    - **Bias:** *What kind of biases might exist in the model, given its training data? (Hint: Think about writing styles, dialects, and cultural perspectives).*
    - **Hallucinations:** *What kind of incorrect feedback might the model "hallucinate"?*
    - **Over-Reliance Risk:** *What is the risk if students trust the tool too much and stop thinking critically for themselves?*

Module 7: Evaluation Metrics for LLMs

This module introduces the key methods for evaluating **Large Language Models (LLMs)**. It is divided into two main sections — metrics for *generation and fluency* (how well the model produces natural text) and *benchmarks for comprehension and safety* (how well the model understands, reasons, and behaves responsibly).

7.1: Metrics for Generation and Fluency

This section covers the fundamental automated metrics used to measure the quality, coherence, and fluency of text generated by LLMs. Evaluating how "good" generated text is can be tricky — we want numerical measures, but true quality often depends on human interpretation. Three automated metrics are widely used: Perplexity, BLEU and ROUGE.

7.1.1 Perplexity (PPL)

Perplexity is a core metric used to evaluate how well a language model predicts a sample of text. In simple terms, it measures the **model's uncertainty or "surprise"** when processing a sequence of words.[24]

- A **lower** perplexity score indicates that the model is more confident and accurate in its predictions, meaning the text is less "perplexing" to it. A higher score means the model was very surprised by the text, suggesting it's a poor fit for the data.
- Perplexity is calculated as the exponentiated average negative log-likelihood of a sequence. A model assigns a probability to each word in a sequence. If the probabilities for the correct words are high, the perplexity will be low.

  Given a test set of words $W = (w_1, w_2, \dots, w_N)$, the perplexity (PPL) is calculated as

  $$PPL(W) = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P\,(w_i|w_1, \dots, w_{i-1})\right)$$

  or equivalently (using base 2 logarithm):

  $$PPL(W) = 2^{-\frac{1}{N}}\sum_{i=1}^{N} log_2 P(w_i|w_{1:i-1})$$

[24] Perplexity

Where:

- N= total number of tokens in the test set
- $P(w_i \mid w_{1:i-1})$= probability assigned by the model to token $w_i$given all previous tokens

Interpretation

- Lower **PPL** → better performance (model is less "perplexed").
- $\text{PPL} = 1$means **perfect prediction** (no uncertainty).
- Higher **PPL** means the model assigns lower probability to the actual tokens, indicating worse predictions.

Example

If a model assigns:

- $P(\text{"the"}) = 0.4$
- $P(\text{"cat"}) = 0.3$
- $P(\text{"sat"}) = 0.2$

Then perplexity for this sequence is:

$$\text{PPL} = \exp\left(-\frac{1}{3}[\ln(0.4) + \ln(0.3) + \ln(0.2)]\right) \approx 2.83$$

In Practice

- **Used for:** language modeling tasks (e.g., GPT, BERT pretraining)
- **Not suitable for:** open-ended text generation quality — use BLEU, ROUGE, or human evaluation instead.
- **Toolkits:** PyTorch and Hugging Face compute perplexity by taking `torch.exp(loss)` when loss = cross-entropy.

- **Use Case:** It's primarily used during the training phase to compare the performance of different language models on a fixed dataset.

7.1.1a Using PyTorch to compute perplexity

When working with deep learning models in PyTorch, perplexity is often derived from the cross-entropy loss `torch.exp(loss)`.

```python
#Using PyTorch (manual cross-entropy)
import torch
import torch.nn as nn

# Example: predicted logits from the model (batch_size=1, seq_len=3,
vocab_size=5)
logits = torch.tensor([[[2.0, 0.5, 0.3, 0.1, 0.1],
                        [1.5, 2.0, 0.3, 0.1, 0.1],
                        [0.2, 0.3, 1.5, 2.0, 0.1]]])

# True target token indices (ground truth)
targets = torch.tensor([[0, 1, 3]])

# Cross-entropy loss
loss_fn = nn.CrossEntropyLoss()
loss = loss_fn(logits.view(-1, logits.size(-1)), targets.view(-1))

# Compute perplexity
ppl = torch.exp(loss)

print(f"Cross-Entropy Loss: {loss.item():.4f}")
print(f"Perplexity (PPL): {ppl.item():.4f}")
```

```
Sample output:
Cross-Entropy Loss: 0.6713
Perplexity (PPL): 1.9567
```

Interpreting the metrics

1. Cross-entropy loss (0.6713)

    - A low cross-entropy loss indicates high model accuracy and confidence. The loss measures the difference between the model's predicted probability distribution and the true distribution.
    - 0.6713 is a good score. As a general benchmark, a loss below 0.20 is often considered good, and anything close to zero is perfect. A loss below 1.0 is a positive sign.
    - The closer to 0, the better. For example, a loss of 0 would mean the model made perfect predictions with 100% confidence.

- It heavily penalizes confident wrong answers. The loss function uses a logarithm, which means it increases significantly when the model predicts the wrong answer with high confidence. This pushes the model to be more careful and resolve uncertainty.

2. Perplexity (1.9567)

- A low perplexity score is better and indicates strong predictive performance. Perplexity is the exponentiation of the cross-entropy loss and is more intuitive to interpret.
- 1.9567 is a very good score. It means the model is, on average, only "perplexed" or confused between about 1.96 equally probable choices when predicting the next item.
- It can be interpreted as an effective branching factor. For example, a perplexity of 10 would mean the model is as uncertain as if it had to choose uniformly from 10 options at each step. A score under 2 suggests the model is very confident and accurate.
- The upper limit of the perplexity is dependent on the model's vocabulary size.

7.1.1b Using Hugging Face models to compute perplexity

For evaluating pre-trained or fine-tuned language models from the Hugging Face ecosystem, the **evaluate** library provides a convenient way to calculate perplexity.

```
import evaluate

# Load the perplexity metric
perplexity = evaluate.load("perplexity", module_type="metric")

# Define your input texts
input_texts = [
    "The quick brown fox jumps over the lazy dog.",
    "I love natural language processing and machine learning."
]

# Compute perplexity using a pre-trained model (e.g., 'gpt2')
results = perplexity.compute(model_id='gpt2', add_start_token=False,
predictions=input_texts)

# Access the results
print(f"Perplexities for each text: {results['perplexities']}")
print(f"Mean perplexity: {results['mean_perplexity']}")
Sample output:
```

```
Perplexities for each text: [162.4564971923828, 89.34370422363281]
Mean perplexity: 125.90010070800781
```

Interpreting the metrics

a) Perplexity Score for sentence “The quick brown fox jumps over the lazy dog." is 162.4, indicating that the model is not confident or uncertain in predicting the next word.

b) Perplexity Score for Sentence "I love natural language processing and machine learning." has a lower perplexity score of 89.3, suggesting the model was more confident about predicting the next word.

c) Mean Perplexity Score: Mean perplexity score for the batch of texts is 125.9, which gives an overall sense of how well the model performed on these two sentences.

7.1.2 BLEU (Bilingual Evaluation Understudy) Score

The BLEU score is the industry standard for evaluating the quality of machine-translated text. It measures how similar a candidate translation (from the machine) is to one or more high-quality human translations (references).

- **BLEU (Bilingual Evaluation Understudy)** is a **precision-based metric** used to evaluate machine translation quality. It doesn’t assess meaning or grammar directly; instead, it measures how many words and phrases in a machine-generated output also appear in reference human translations.[25]
- **How It Works:** BLEU compares overlapping *n-grams*—that is, contiguous sequences of words such as unigrams (1-word), bigrams (2-word), and trigrams (3-word). It computes the precision of these n-gram matches across multiple lengths (typically 1 to 4) and combines them into a single overall score. To discourage overly short translations, BLEU also applies a **brevity penalty** when the candidate translation is shorter than the reference.
- **Use Case:** Commonly applied in **machine translation evaluation**, a higher BLEU score indicates closer alignment with human reference translations.
  A BLEU score of **1.0 (100%)** represents a perfect match, while **0** indicates no overlap.
  Mathematical Definition

[25] BLEU

$$\text{BLEU} = \text{BP} \times \exp\left(\sum_{n=1}^{N} w_n \log\ p_n\right)$$

Where:

- p_n: precision for n-grams of size *n* (1-gram, 2-gram, ...)
- w_n: weight for each n-gram level (commonly uniform, e.g., 0.25 each for BLEU-4)
- BP: brevity penalty to penalize overly short translations

n-gram precision p_n:

$$p_n = \frac{\text{Number of matching n-grams}}{\text{Total number of candidate n-grams}}$$

Brevity Penalty (BP):

$$\text{BP}=\begin{cases} 1, & \text{if } c > r \\ \exp\left(1 - \frac{r}{c}\right), & \text{if } c \le r \end{cases}$$

- c: length of candidate sentence
- r: length of reference sentence

- **BP = 1.0** → output length is acceptable (no penalty applied).
- **BP < 1.0** → output is too short, so its BLEU score is reduced exponentially.

- **BLEU = 1.0 (100%)** → Perfect match with reference text
- **BLEU = 0.0** → No overlap
- Real-world machine translation or LLM text often scores **0.2–0.6** range

Example
Reference:
"The cat is on the mat."
Candidate (Model Output):
"The cat sat on the mat."

- Overlaps in 1-grams (e.g., *The, cat, on, the, mat*) → 5 matches
- Some mismatched bigrams (*cat sat* vs *cat is*)
- Resulting BLEU (roughly BLEU-2) ≈ **0.67**

7.2.2a Using NLTK to compute BLUE

The Natural Language Toolkit (NLTK) is a powerful Python library for working with human language data. It provides a comprehensive set of tools for various natural language processing (NLP) tasks.

```
# Using NLTK for BLEU Score
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction

reference = [['the', 'cat', 'is', 'on', 'the', 'mat']]
candidate = ['the', 'cat', 'sat', 'on', 'the', 'mat']

smooth = SmoothingFunction().method1
score = sentence_bleu(reference, candidate, weights=(0.5, 0.5),
smoothing_function=smooth)

print(f"BLEU Score: {score:.4f}")
```

Sample output:
BLEU Score: 0.7071

7.2.2b Using Hugging Face (Evaluation Library)

Hugging Face's Evaluation library makes it straightforward to calculate BLEU scores for machine translation or text generation tasks without manually handling n-gram matching or brevity penalties.

```
import evaluate

bleu = evaluate.load("bleu")

predictions = ["the cat sat on the mat"]
references = [["the cat is on the mat"]]

result = bleu.compute(predictions=predictions,
references=references,smooth=True)
print(f"BLEU Score: {result['bleu']:.4f}")
print(result)
```

Sample output:

BLEU Score: 0.4889

```
{'bleu': 0.488923022434901, 'precisions': [0.8571428571428571,
0.6666666666666666, 0.4, 0.25], 'brevity_penalty': 1.0, 'length_ratio': 1.0,
'translation_length': 6, 'reference_length': 6}
```

Explanation of Results

- **bleu**: Final BLEU score (weighted geometric mean of n-gram precisions times brevity penalty)
- **precisions**: Precision for n-grams of lengths 1–4
- **brevity_penalty**: Penalizes short outputs = 1 so no penalty
- **length_ratio**, **translation_length**, **reference_length**: Provide length comparison context

7.1.3 ROUGE (Recall-Oriented Understudy for Gisting Evaluation) Score

The ROUGE score is a set of metrics used to evaluate the quality of automatically generated summaries. It is the counterpart to BLEU. ROUGE evaluates how well a generated text (e.g., summary, translation, model output) matches reference (human-written) texts by comparing overlapping units such as:

- words,
- word sequences (n-grams),
- or longest common subsequences (LCS).

Unlike *BLEU*, which emphasizes *precision* (how much of the generated output matches reference), ROUGE emphasizes recall — *how much of the reference text is covered by the generated text.*[26]

Use Case: Text Summarization. A higher ROUGE score generally indicates a better summary.

7.1.3a Common Variants of ROUGE

Variant	Measures	Description	Typical Use
ROUGE-N	Recall of overlapping *n-grams*	e.g., ROUGE-1 (unigrams), ROUGE-2 (bigrams)	Basic overlap of content words
ROUGE-L	Longest Common Subsequence (LCS)	Captures sentence-level structure similarity	Summarization and long text evaluation

[26] ROUGE (metric)

ROUGE-W	Weighted LCS	Penalizes non-consecutive matches	Better for fluent text
ROUGE-S / ROUGE-SU	Skip-bigram overlap	Considers pairs of words allowing skips	Captures partial ordering

Formula for ROUGE-N

$$\text{ROUGE-N} = \frac{\sum_{\text{n-gram}\in\text{Ref}} \min\left(\text{Count}_{\text{Gen}}(\text{n-gram}), \text{Count}_{\text{Ref}}(\text{n-gram})\right)}{\sum_{\text{n-gram}\in\text{Ref}} \text{Count}_{\text{Ref}}(\text{n-gram})}$$

Where:

- *Ref* = reference (human text)
- *Gen* = generated text
- Measures **recall** of n-grams from reference captured in the generated output

Example
Reference:
“The cat is on the mat.”

Candidate (Model Output):
“The cat sat on the mat.”

Overlaps:

ROUGE-1 (word overlap): 5 / 6 = 0.83
ROUGE-2 (bigram overlap): 2 / 5 = 0.40
ROUGE-L (LCS = “the cat on the mat” → 5 words / 6) = 0.83

7.1.3b Python Example (Using Hugging Face evaluate)

```
# The ROUGE metric depends on a small extra package called rouge_score that
isn’t installed automatically.
# pip install rouge_score

import evaluate

# Load the metric
rouge = evaluate.load("rouge")

# Example texts
predictions = ["the cat sat on the mat"]
references = ["the cat is on the mat"]
```

```
# Compute ROUGE
result = rouge.compute(predictions=predictions, references=references)
print({k: round(v, 4) for k, v in result.items()})
```

```
sample output:
{'rouge1': 0.8333, 'rouge2': 0.4, 'rougeL': 0.8333,
'rougeLsum': 0.8333}
```

Explanations of metrics:

ROUGE-1 (Unigram Recall = 0.8333)
ROUGE-1 measures the recall of single words (unigrams) from the reference that appear in your generated text.
Example:
Reference: “The cat is on the mat.”
Generated: “The cat sat on the mat.”
Matching words: the, cat, on, the, mat → 5 matches out of 6 total in the reference.

$$\text{ROUGE-1} = \frac{5}{6} = 0.8333$$

So your model’s output contains 83% of the same words as the reference — that’s very good coverage.

ROUGE-2 (Bigram Recall = 0.4000)
ROUGE-2 measures recall of word pairs (bigrams) — basically, how many adjacent word pairs match between the two texts.
Example:
Reference bigrams:
the cat, cat is, is on, on the, the mat
Generated bigrams:
the cat, cat sat, sat on, on the, the mat
Matching bigrams: the cat, on the, the mat → 2 or 3 out of 5 depending on tokenization

$$\text{ROUGE-2} \approx 0.4$$

So about 40% of phrase-level relationships were correctly captured.

ROUGE-L (Longest Common Subsequence = 0.8333)
ROUGE-L focuses on sentence-level structure by finding the longest sequence of words that appear in both sentences in the same order (not necessarily adjacent).
Here, the LCS = “the cat on the mat” (5 words).

$$\text{ROUGE-L} = \frac{5}{6} = 0.8333$$

So, the generated text preserved about 83% of the reference's sequential structure.

ROUGE-Lsum (Summary-Level LCS = 0.8333)
ROUGE-Lsum is similar to ROUGE-L but is used for multi-sentence summaries.
In this case (single sentence), it's identical to ROUGE-L.

7.2 Limitations and Human Judgment

While fast and scalable, automated metrics like BLEU and ROUGE have significant limitations.

- **Surface-Level Analysis:** They rely on exact word overlap and cannot truly comprehend semantics, nuance, or grammatical correctness. A generated sentence can be perfectly valid and convey the same meaning as the reference but receive a low score for using different words (synonyms).
- **Ignoring Coherence:** These metrics do not effectively measure the overall coherence or factual accuracy of the generated text.
- **The Need for Humans:** Because of these flaws, **human evaluation remains the gold standard**. Humans are needed to assess subtle qualities like creativity, style, safety, and whether the text actually makes sense in context.

7.3: Benchmarks for Comprehension and Safety

This section explores the standardized tests and adversarial methods used to assess an LLM's understanding, reasoning, and safety alignment.

7.3.1 Classification Metrics

For many NLP tasks, the LLM must classify text into categories (e.g., positive/negative sentiment, spam/not spam). In these cases, we use standard classification metrics.

- **Accuracy:** The percentage of total predictions the model got right. It's a good general measure but can be misleading if the classes are imbalanced.
    - *Accuracy = (True Positives + True Negatives) / Total Predictions*
- **Precision:** Of all the times the model predicted a positive outcome, what percentage was correct? It measures **exactness**.
    - *Precision = True Positives / (True Positives + False Positives)*
- **Recall:** Of all the actual positive outcomes, what percentage did the model correctly identify? It measures **completeness**.
    - *Recall = True Positives / (True Positives + False Negatives)*
- **F1-Score:** The harmonic mean of Precision and Recall. It provides a single score that balances both, which is useful when you care about both false positives and false negatives.

    - *F1-Score = 2 * (Precision * Recall) / (Precision + Recall)*

7.3.2 Standardized Benchmarks

Benchmarks are comprehensive test suites composed of multiple datasets and tasks, designed to provide a holistic evaluation of a model's language capabilities.

- **GLUE & SuperGLUE (General Language Understanding Evaluation):** These are collections of challenging tasks designed to test a model's understanding of sentence structure, inference, and sentiment. SuperGLUE was developed to be a harder successor to GLUE.
- **MMLU (Massive Multitask Language Understanding):** This benchmark measures an LLM's breadth of world knowledge and problem-solving ability. It consists of multiple-choice questions on 57 different subjects, including mathematics, U.S. history, law, and computer science, at levels ranging from elementary to professional.

7.3.3 Red Teaming

Red teaming is a form of **adversarial testing** where a dedicated team actively tries to break the model's safety protocols.

- **Core Concept:** It's like ethical hacking for LLMs. The goal is to proactively discover flaws, biases, and unsafe behaviors before they can be exploited by malicious users.
- **Process:** Red teamers craft clever and often non-obvious prompts to try and coax the model into:
    - Generating harmful, biased, or explicit content.
    - Revealing sensitive information.
    - Assisting in dangerous or illegal activities.
- **Importance:** This process is crucial for identifying and fixing safety vulnerabilities that are not typically found through standard testing or benchmarks.

Module 7: Learning Materials & References

a) LLM Perplexity
b) BLEU Score Explained
c) ROUGE (Metric)
d) What is the ROUGE metric?

Part A: Quiz

T/F Multiple choice

7.1a What does perplexity measure in the context of LLM-generated text?
A. The similarity between generated and reference text.
B. A model's uncertainty in predicting the next token.
C. The recall of key phrases in generated summaries.
D. The factual accuracy of generated responses.

7.2a Which metric is primarily used to measure the quality of machine translation in LLM outputs?
A. ROUGE Score
B. Perplexity
C. BLEU Score
D. F1 Score

7.3a Which of the following classification metrics balances precision and recall?
A. ROUGE Score
B. Perplexity
C. BLEU Score
D. F1 Score

7.4a What is the key purpose of Red Teaming in evaluating LLMs?
A. To find flaws, biases, and unsafe behaviors in the model.
B. To train models on new data.
C. To measure BLEU and ROUGE scores automatically.
D. To calculate perplexity values.

7.5a Why are human evaluations still necessary despite automated LLM metrics?
A. Because automated metrics are too slow.
B. Because automated metrics perfectly capture semantic meaning.
C. Because automated metrics often miss nuances like factuality and coherence.
D. Because human evaluations are cheaper to perform.

Part C: Activity

7.1c Discussion on True Understanding

Benchmarks like MMLU are impressive, but they often test a model's ability to recall information or recognize patterns from its vast training data.

- **Prompt for Discussion:** "How could we design a benchmark that moves beyond pattern matching and more effectively tests for genuine 'understanding,' 'reasoning,' or 'common sense' in an AI? What are the inherent challenges in creating such a test?"

7.2c Simplified BLEU-1 Score Calculation

Let's manually calculate a simplified BLEU score that only considers unigrams (single words).

- **Candidate Translation (Machine):** "the cat sat on the mat"
- **Reference Translation (Human):** "the cat is on the mat"

1. **Count words in the candidate:** There are 6 words.
2. **Count matching words:** "the" (appears twice), "cat", "on", "mat". The word "sat" from the candidate is not in the reference.
3. **Clip the counts:** The word "the" appears twice in the candidate but only once in the reference. So we "clip" its count to 1. The other matching words ("cat", "on", "mat") appear once in both.
4. **Calculate Precision:**
    - Sum of clipped counts = 1 (for "the") + 1 (for "cat") + 1 (for "on") + 1 (for "mat") = **4**.
    - Total words in candidate = **6**.
    - **Simplified BLEU-1 Score** = 4 / 6 ≈ **0.67**.

Part D: Python Exercises

7.1d. BLEU Score Calculation

Write a Python program using the evaluate library to compute the BLEU score between the following sentences:

Reference: "The quick brown fox jumps over the lazy dog."
Candidate: "The quick brown fox jumped over the lazy dog."

Tasks:

1. Load the BLEU metric.
2. Compute and print the BLEU score.
3. Add smoothing to handle short sentences.
4. Explain your outputs what do they mean?

```
Sample output:
BLEU Score: 0.7017
```

7.2d ROUGE Evaluation

Use the **ROUGE** metric to evaluate how similar two short summaries are.

Reference summary: "Large language models are trained on massive text data."
Generated summary: "LLMs learn from huge amounts of text data."

Tasks:

1. Compute ROUGE-1, ROUGE-2, and ROUGE-L.
2. Display each score with four decimal precisions.
3. Interpret which ROUGE score best captures structural similarity.
4. Explain your outputs what do they mean?

```
Sample output:
{'rouge1': 0.2353, 'rouge2': 0.1333, 'rougeL': 0.2353}
```

7.3d Perplexity (PPL)

Assume a model's average **cross-entropy loss** after evaluation is loss = 4.25.
Write a Python snippet to compute and print the **Perplexity (PPL)** using the standard formula:

$$PPL = e^{loss}$$

Tasks:

1. Compute PPL in Python.
2. Print both the loss and the resulting PPL.
3. Explain what a high PPL value means for the model's language prediction quality.
4. Explain your outputs what do they mean?

```
Cross-Entropy Loss: 4.2500
Perplexity (PPL): 70.1054
```

7.4d Combined Metric Comparison

Using the **Hugging Face evaluate** library, compute both **BLEU** and **ROUGE** scores for:

Reference: "Artificial intelligence is transforming the world."
Prediction: "AI is changing the world."

Tasks:

1. Print BLEU, ROUGE-1, and ROUGE-L.
2. Compare which metric gives a higher similarity value and explain why.

3. Discuss which metric is more appropriate for summarization vs. translation.
4. Explain your outputs what do they mean?

```
Sample output
BLEU Score: 0.3680
ROUGE-1: 0.5455
ROUGE-L: 0.5455
```

Addendum: Applied Projects for Introduction to LLM

This addendum introduces a capstone-style applied project designed to consolidate foundational knowledge from the Introduction to Large Language Models course.

Project Title: Development of a Mini LLM-Powered Assistant

Purpose:

The project enables students to integrate theoretical concepts—including tokenization, embeddings, context windows, and prompt engineering—into a functioning prototype demonstrating real-world relevance.

Project Description:

Students shall design and implement a lightweight interactive system using an open-source LLM framework (e.g., Ollama, Hugging Face, or OpenAI API). The system should accept user input, generate responses based on context, and illustrate the impact of prompt design and model parameters.

Key Requirements:

- Implement programmatic interaction with an LLM in Python.
- Apply embeddings or structured prompts to manage conversational context.
- Include documentation detailing model selection and design rationale.
- Present a short demonstration showcasing prompt processing.

Deliverables:

1. Source code repository with documentation.
2. Concise written report (1–2 pages) summarizing design, function, and learning outcomes.
3. Recorded or live presentation of the working prototype.
4. Grading see **Table Add-1.**

Project Title: Development of a Cybersecurity Chatbot Using LLMs

Purpose:

Students will integrate course concepts—such as tokenization, prompt engineering, and embeddings—into a functional chatbot capable of answering common cybersecurity questions or assisting in security-related workflows.

Project Description:

Design and implement a Python-based chatbot utilizing an open-source LLM (e.g., Ollama, Hugging Face, OpenAI API) that can interpret user queries about cybersecurity topics and generate contextually relevant responses. The bot should demonstrate understanding of prompt design, contextual awareness, and response tuning using model parameters.

Key Requirements:

- Programmatic interaction with an LLM focused on cybersecurity knowledge.
- Use embeddings, FAQs, or targeted prompts to manage context and accuracy.
- Provide documentation explaining model choice and system design.
- Deliver a demonstration illustrating how user inputs lead to informed cybersecurity outputs.

Deliverables:

1. Source code repository with comments and documentation.
2. Written report (1–2 pages) discussing design, cybersecurity content integration, and lessons learned.
3. Recorded or live demo showing the chatbot in action.
4. Grading see **Table Add-1**.

Project Title: Comparing LLMs for Cybersecurity Threat Report Summarization and Analysis

Purpose:

This project introduces students to **practical applications of LLMs in cybersecurity** — specifically, using language models to **summarize, analyze, and benchmark** threat intelligence reports.

Key Requirements:

- Use **pretrained LLMs** (like GPT-2, BART, and T5) for text summarization.
- Evaluate models using **ROUGE, BLEU, and Perplexity** metrics.
- Compare model performance and interpret their usefulness in cybersecurity analysis.
- Reflect on **ethical and security implications** of using LLMs in this domain.

Deliverables:

- Which model produced more **accurate** and **concise** summaries?
- Did the model **miss or hallucinate** any key threat details?
- What are the **risks** of using LLMs in cybersecurity (e.g., misinformation, data leaks)?
- How can evaluation metrics guide model selection in real-world applications?
- Written report (1–2 pages) discussing design, cybersecurity content integration, and lessons learned.
- Recorded or live demo showing the chatbot in action.
- Grading see **Table Add-1**.

Component	Description		Weight
Model Evaluation Code	Comparison setup, metric calculation		40%
Performance Report	Tables, visualizations, analysis		25%
Generated Samples	3+ side-by-side model outputs		10%
Ethical Reflection	Fairness, bias, and transparency discussion		15%
Final Presentation	Short video or slides summary		10%

TABLE Add-1