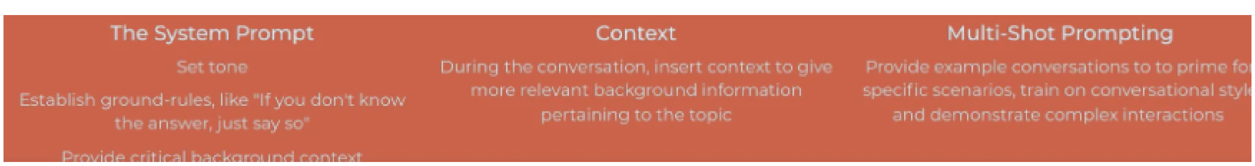
LLM Engineering key concepts

1. Introduction to Large Language Models (LLMs)

- ❖ Core Terminologies and Concepts in LLMs
- ❖ **Prompt**
  - ➢ The mechanism through which end users communicate with LLMs, providing task instructions or feedback using natural language.
  - ➢ A rapidly evolving field with dynamic best practices for designing effective prompts.
- ❖ **Token**
  - ➢ The basic unit of text processed by LLMs, representing words or subwords.
  - ➢ Input text is tokenized into a vocabulary, enabling the model to process requests and generate responses.
  - ➢ Tokens are also the basis for API cost calculations.
- ❖ **Embeddings**
  - ➢ Numerical representations of data (e.g., text) that reduce dimensionality for computational efficiency.
  - ➢ Generated via machine learning models, embeddings enable mathematical operations on data, such as identifying semantically similar text chunks.

System vs User prompt

Aspect	System Prompt	User Prompt
Definition	A hidden instruction given to the model to **control behavior and responses**.	The input text provided **by the user** to request information or actions.
Who Sets It?	The **developer** or system that runs the model.	The **end user** interacting with the model.
Purpose	Guides the model's **tone, style, and constraints**.	Asks for **specific answers or actions** from the model.
Visibility	**Not visible** to the user (usually hidden in the backend).	**Visible** to the user (entered directly).
Example	`"You are a helpful AI assistant. Answer concisely and avoid controversial topics."`	`"What are the benefits of fine-tuning an LLM?"`
Persistence	**Remains fixed** throughout a session or system usage.	**Changes dynamically** based on user input.



- **Fine-tuning (sub technique of transfer learning like Feature Extraction)**
  - The process of specializing a pre-trained LLM for specific tasks by training it on a smaller, task-specific dataset.
  - Involves adjusting the model's internal weights using techniques like adapter modules or LoRA.
  - Delivers higher-quality results than prompting, reduces token usage, and lowers costs and latency.
- **Retrieval-Augmented Generation (RAG)**
  - An architectural pattern for leveraging LLMs on custom domain data.
  - Involves chunking data, generating embeddings, and storing them in a vector database.
  - Queries are vectorized, matched with relevant chunks, and sent as prompts to the LLM for enhanced results.
- **Agentic AI Systems**
  - Autonomous AI agents capable of planning, executing, and iterating on tasks without human intervention.
  - Examples include AutoGPT and BabyAGI, which use LLMs for decision-making and task automation.
  - Requires careful design to mitigate risks like infinite loops or unintended actions.

Ethical Considerations and Limitations of LLMs

- **Recency Problem**
  - Training data has a cut-off date; LLMs lack knowledge of events or information beyond that date.
- **Hallucinations**
  - LLMs generate plausible but incorrect or nonsensical answers due to a lack of contextual understanding and mismatches between training data and inherent knowledge.
- **Lack of Lineage**
  - LLMs cannot trace the origin or source of information used to generate responses.
- **Inconsistency of Generated Text**
  - Non-deterministic outputs; different prompts or executions yield varying results, making reproducibility challenging.
- **Privacy, Trust, and Compliance Issues**
  - Training on public data risks exposing sensitive or private information, raising compliance concerns.
- **Context Loss**
  - LLMs struggle to maintain context over long texts or complex questions.
- **Bias in Training Data**

➢ Internet-sourced training data often contains biases, leading to inaccurate or skewed outputs.

❖ **Susceptibility to Prompt Injection Attacks**

➢ Vulnerable to malicious prompts, akin to data poisoning or adversarial attacks.

2. Working with LLMs Locally Using Ollama

❖ Ollama vs vLLM (Virtual Large Language Model)

- **Ollama**: A lightweight **local server** for running and serving LLMs efficiently on personal machines. It allows developers to deploy and interact with LLMs **offline** without relying on cloud services.
- **vLLM (Virtual Large Language Model)**: An optimized **LLM inference framework** designed for high-speed and memory-efficient model serving. It is built for **scalable, high-throughput deployments** in cloud and production environments.

Feature	vLLM	Ollama
Purpose	Optimized deployment and inference of very large language models (LLMs) locally.	Simplified deployment of LLMs locally with an easy-to-use interface.
Target Audience	Machine learning engineers, developers working with large models.	Developers, users looking for an easy way to run LLMs locally.
Ease of Use	Requires more technical expertise for setup and optimization.	User-friendly, quick setup for running LLMs locally.
Deployment Complexity	High complexity, more control over hardware and inference optimization.	Low complexity, focus on ease of deployment without deep technical knowledge.
Customization & Flexibility	High flexibility for scaling, fine-tuning, and optimizing large models.	Limited customization, focuses on ease and simplicity over advanced configuration.
Model Support	Supports a variety of large models (GPT, T5, etc.), optimized for local deployment.	Supports popular LLMs, but may have limitations in terms of scalability.
Performance Optimization	Advanced optimizations for memory and inference speed, supports GPUs/TPUs.	Basic optimization, more focused on ease of use and accessibility.
Hardware Requirements	Designed for environments with powerful GPUs/TPUs, scalable hardware setups.	Can be run on typical personal machines, with less emphasis on hardware optimization.

Main Use Case	Large-scale deployment, fine-tuning, and inference for high-performance applications.	Running LLMs locally for personal or smaller-scale use, prototyping, or casual usage.
Installation	Requires setting up dependencies and optimizations; more flexible.	Simple installation process with a focus on ease.
Community and Support	Community-driven, often requires more technical support or custom configurations.	Easier for beginners, with strong focus on supporting non-experts.

❖ Advantages of Local LLM Deployment

Local deployment with Ollama offers privacy (data remains on-premise), cost efficiency (no cloud API fees), and customization (fine-tuning for domain-specific tasks). On-device inference ensures low latency and offline capabilities, critical for real-time applications and environments with restricted internet access. This approach also avoids API rate limits and provides full control over model behavior.

❖ Hardware and Software Requirements

Ollama requires NVIDIA GPUs (e.g., A100, RTX 4090) for accelerated inference, at least 16GB RAM for smaller models, and SSD storage for efficient data processing. Supported operating systems include Linux, macOS, and Windows (via WSL), with Python 3.8+ and libraries like PyTorch or TensorFlow. Docker is optional for containerized deployment.

❖ Resource Optimization Techniques

Optimization techniques include:

- **Quantization**: Reducing model size and memory usage (e.g., 8-bit or 4-bit).
- **Batching**: Processing multiple inputs in parallel to maximize GPU utilization.
- **Offloading**: Balancing CPU-GPU workloads for memory-intensive tasks.
- **Caching**: Storing frequent queries to avoid redundant computations.
- **Pruning**: Removing less important model weights to improve inference speed.

3. Proprietary (Closed Source) LLM Platforms

- ➔ OpenAI ecosystem
  - ❖ https://platform.openai.com/docs/models
- ➔ Other commercial LLM providers
  - ❖ https://docs.anthropic.com/en/docs/about-claude/models
  - ❖ https://cloud.google.com/vertex-ai/generative-ai/docs/learn/models
- ➔ Comparing LLM platforms

- Free trials and pricing structures

LLM	Developer	Monthly Cost
GPT-4o	Open AI	$20
GPT-4o-mini	Open AI	Free
Claude 3.5 Sonnet	Anthropic	$20
Gemini 1.5 Flash	Google	Free
Gemini 1.5 Pro	Google	$20
Mistral Large 2	Mistral AI	Free

Tokenization methods

Tokenization is the process of breaking text into smaller units (**tokens**) that a machine learning model can process. In **LLMs**, tokens can be words, subwords, or characters, allowing efficient text representation.

- **Word-based**: Splits text into words (e.g., `"AI is powerful"` → `["AI", "is", "powerful"]`).
- **Character-based**: Splits text into individual characters (e.g., `"AI"` → `["A", "I"]`).
- **Subword-based** (**BPE, WordPiece, Unigram**): Efficiently handles rare words by breaking them into meaningful parts (e.g., `"unhappiness"` → `["un", "happiness"]`).
- **SentencePiece**: Works without pre-segmentation, ideal for multilingual models.
- **Subword-based tokenization (e.g., BPE, WordPiece) is the most used in LLMs** like GPT, BERT, and Llama for better efficiency and generalization.

- Context window sizes
  - In the context of LLMs (Large Language Models), a "context window" refers to the span of input tokens (words or characters) that the model can process at a given time. This is essentially the amount of text the model considers when making predictions .
  - The context window includes not just the current input but also the preceding text or interactions, such as previous questions and answers. For example, in a conversation with an LLM, the context window might include the most recent question you've asked as well as the model's response, allowing it to maintain continuity in the dialogue.

Model	Context Window Size	Notes
Claude (Anthropic)	~100,000 tokens	Large context window, ideal for maintaining long interactions.
GPT-4 (OpenAI)	Up to 32,768 tokens	Handles long inputs and documents, excellent for detailed tasks.
Gemini (Google)	8,000–32,000 tokens	Flexible window depending on model variant; suited for medium-to-large inputs.
LLaMA 3 (Meta)	Up to 8,000 tokens	Smaller context window, but optimized for high efficiency.
Mixtral (Mistral)	Up to 12,800 tokens	A balance between context and performance, tailored for flexibility.
DeepSeek	~100,000 tokens	Known for a large context window, designed for long-term memory and extended interactions.

- ## Model parameters and capabilities

❖ **Model parameters** are the internal variables (weights and biases) in a machine learning model that are learned during training. These parameters determine how input data is processed and how the model generates predictions or outputs. In neural networks, parameters control the connections between neurons in different layers, allowing the model to capture patterns and make decisions based on the data it has been trained on. The number of parameters is often used as a measure of the model's complexity and capacity to learn.

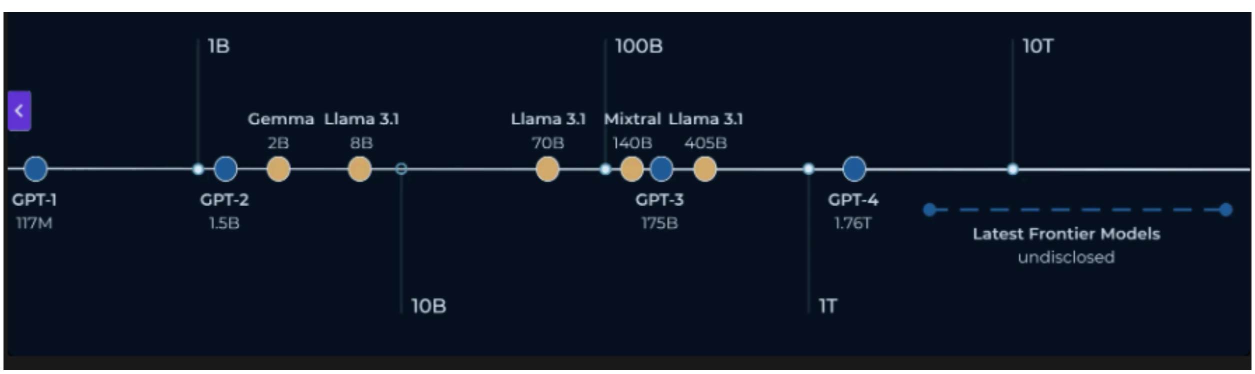
Metric	Description	Why It Matters
Number of Parameters	The total number of trainable weights in the model.	A larger number of parameters generally means better ability to capture complex patterns.
Context Window Size	The amount of text (tokens) the model can consider at once.	Larger context windows allow models to handle longer conversations and more complex tasks.
Training Data	The type and amount of data the model was trained on.	More diverse and extensive training data can improve the model's generalization and performance on a wider range of tasks.
Accuracy/Performance	How well the model performs on benchmark tasks or real-world applications.	Helps measure the model's practical utility. For example, GPT-4 might be better for certain tasks due to its larger training corpus and more parameters.
Inference Speed	The time it takes for the model to generate a response.	Important for real-time applications or systems requiring quick responses.
Memory Usage	The computational resources the model requires to run.	A model with more parameters generally requires more memory and computational power, which can affect deployment.
Fine-Tuning Capability	How easily the model can be adapted to specific tasks.	Models with more parameters may offer more flexibility, but they can be harder to fine-tune

- Cost calculation and optimization

LLM	Tokenizer Type	Approx. Characters per Token
GPT-4 / GPT-3.5	Byte Pair Encoding (BPE)	~4 characters per token
Claude (Anthropic)	Custom variant of BPE	~3-4 characters per token
Gemini (Google DeepMind)	SentencePiece (Unigram)	~3 characters per token
LLaMA 2 (Meta)	SentencePiece (BPE)	~3-4 characters per token
Mistral / Mixtral	BPE (similar to LLaMA)	~3-4 characters per token
T5 / mT5 / UL2	SentencePiece (Unigram)	~3 characters per token

Model	Training Cost	Inference Cost	Optimization Options
GPT-4	High (large model, expensive hardware required)	High (slower inference, high GPU cost)	Pruning, distillation, quantization, fine-tuning
Claude	High (similar to GPT-4, high compute needs)	High (large but efficient for specific tasks)	Pruning, quantization, task-specific fine-tuning
Gemini	High (high compute cost)	High (similar to GPT-4, requires strong infrastructure)	Distillation, model simplification
LLaMA 3	Moderate (optimized for efficiency)	Lower (more efficient at inference)	Quantization, pruning, fine-tuning
Mixtral	Moderate (optimized model architecture)	Lower (more efficient for inference)	Quantization, distillation, model simplification
DeepSeek	Low (lightweight models, optimized for efficiency)	Low (very efficient for small tasks)	Fine-tuning, knowledge distillation

Model	Training Cost	Inference Cost	Monthly Cloud/Infra Cost	Energy Cost (Annual)
GPT-4	$10M - $50M	$0.01 - $0.10/query	$100K - $500K	$100K - $1M
Claude	$10M - $30M	$0.02 - $0.10/query	$50K - $200K	$50K - $500K
Gemini	$15M - $50M	$0.05 - $0.15/query	$50K - $300K	$100K - $500K
LLaMA 3	$500K - $2M	$0.002 - $0.05/query	$10K - $100K	$10K - $100K
Mixtral	$500K - $2M	$0.01 - $0.05/query	$5K - $50K	$5K - $50K
DeepSeek	$50K - $500K	$0.001 - $0.01/query	$1K - $10K	$500 - $5K



- ## Multi-shot prompting and context enrichment
    - ❖ **Multi-shot prompting** is a prompt engineering technique where multiple examples are provided to a pre-trained model in a single request to guide its behavior and improve the quality of the response. By offering several examples, the model can better understand the desired format and context, leading to more accurate and relevant outputs during inference or API interactions.

```
Prompt to ChatGPT:
Convert text to professional tone

Example 1:
Casual: Hey, just checking if you got my email about the project
Professional: I am writing to follow up regarding the project-related email I sent.

Example 2:
Casual: The meeting was super fun and we got lots done!
Professional: The meeting was productive and we achieved several key objectives.

Now convert this:
Casual: Can you look at this when you get a chance?
```

4. Open Source Models and Hugging Face

- Understanding Hugging Face ecosystem

  https://huggingface.co/

  https://huggingface.co/docs/transformers/en/installation

- Implementation methodologies
    - ### Pipeline architecture

❖ Pipeline: Simplifies the process of running inference. It automatically handles preprocessing, inference, and postprocessing, which is great for quick tasks.

🛠️ `pipeline` Automates the Steps We Normally Code

Step	Without `pipeline`	With `pipeline`
Preprocessing	Tokenization	Done internally
Inference	Model forward pass	Done internally
Postprocessing	Convert logits to labels	Done internally
Code Complexity	Longer	Shorter, 1 line

- model inference

preprocessed inputs.It involves **preprocessing** (input transformation), **model inference** (the forward pass), and **postprocessing** (output conversion).

- Pipeline vs. Tokenization Method:
- **Pipeline**: Simplifies the process of running inference. It automatically handles preprocessing, inference, and postprocessing, which is great for quick tasks.
- **Manual Tokenization**: Gives you more control, especially for batch processing, fine-tuning, and customizing the input/output format. It’s useful for handling larger datasets or specific tasks that require flexibility.

Scenario	Use `pipeline`	Use Manual Tokenization
Quick inference (simple text classification, translation, etc.)	✅ Yes	❌ No need
Fine-tuning a model (training with new data)	❌ No	✅ Yes
Batch processing (handling multiple inputs efficiently)	❌ No	✅ Yes
Low-level control (custom tokenization, different embeddings, etc.)	❌ No	✅ Yes
Working with multiple models together (e.g., encoder + decoder separately)	❌ No	✅ Yes

❖ Transformer method vs pipeline

Aspect	Pipeline Method	Transformers Method (Manual)
Control Over Tokenization	Limited; automatic tokenization based on the task.	Full control over tokenization parameters.
Customization	No direct customization options for tokenization.	Highly customizable (padding, truncation, max_length).
Complexity	Easy-to-use with minimal code, suitable for quick tasks.	Requires manual setup, more complex but highly flexible.
Padding & Truncation	Handled automatically.	Manual control over padding and truncation.
Batch Handling	Automatic batching handled by pipeline.	Manual control over batching and input management.
Tokenizer Access	Implicitly used and abstracted.	Explicit access to tokenizer with full control.
Use Case	Best for quick tasks without the need for customization.	Best for tasks requiring deep customization or fine-tuning.

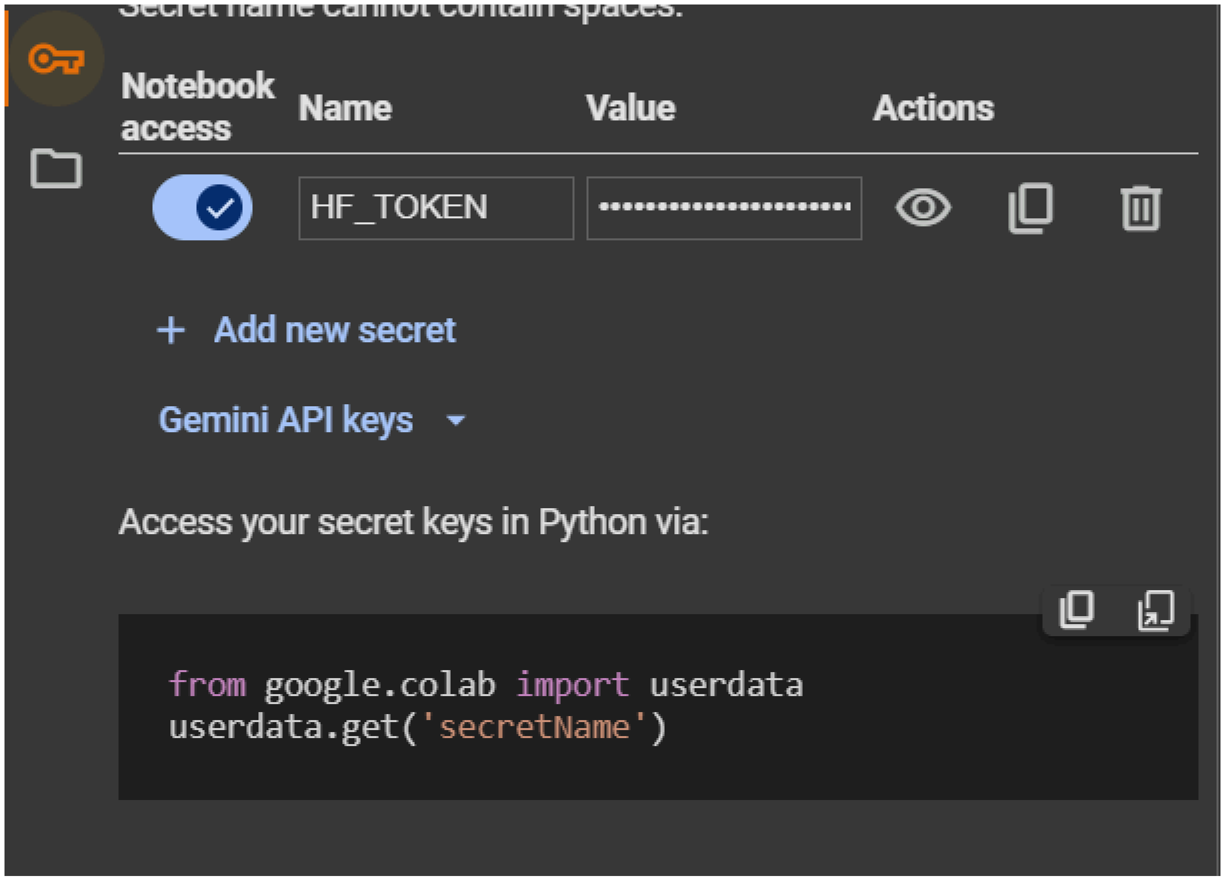
- **Pipeline Method**: Best for quick, simple tasks where you don't need to worry about the internals of tokenization or model configuration.
- **Transformers Method (Manual)**: Best when you need **full control** over the tokenization process, want to **customize** the input handling (e.g., padding, truncation), or are working with more complex use cases.

Approach	Resources Used	Complexity	Cost	Best For	Analogy
Pipeline	Local (CPU/GPU)	Simple	Free (after download)	Quick testing, basic use	"Quick test drive" of models
Direct Transformers	Local (CPU/GPU)	Advanced	Free (but you pay for your own compute)	Fine-tuning, custom training, production use	"Full control of the car"
Inference API	Cloud (Hugging Face's servers)	Very simple	Pay per request (even for open-source models)	No setup, scalable deployments	"Taking a taxi" – convenient but paid

Feature	Pipeline	Direct Transformers	Inference API
Code Complexity	Simplest	Most Complex	Moderate
Setup Required	Minimal	Full model setup	API setup
Memory Usage	Local memory	Local memory	Cloud-based
Fine-tuning Capability	Not supported	Full support with: `trainer = Trainer(model=model, args=args, train_dataset=train_data)trainer.train()`	Limited (only via AutoTrain)
Transfer Learning	Not supported	Full support: Can modify architecture, freeze layers, add custom heads	Not supported
Custom Architectures	Not supported	Fully supported: `class CustomModel(PreTrainedModel):     def __init__(self, config):         super().__init__(config)         # Custom layers`	Not supported
Knowledge Distillation	Not supported	Supported with custom implementation	Not supported
Model Pruning	Not supported	Supported via `optimize_model()`	Not supported

- Working with Hugging Face
    - Google Colab environment setup

```
from google.colab import userdata
token=userdata.get('HF_TOKEN')
print(token)
```



- API integration

➔ If things don’t work as expect you need to restart the session or remove and back with new token

- Model selection criteria

Criteria	Description	Considerations
Task Type	Determines if the model fits your specific NLP, CV, or speech task.	Text classification, summarization, translation, image recognition, speech processing, etc.
Model Architecture	Defines the structure and learning mechanism of the model.	BERT (classification), GPT (text generation), ViT (vision), Whisper (speech-to-text), etc.
Model Size	Affects inference speed, memory usage, and computational requirements.	Small (fast, low memory), Medium (balanced), Large (high accuracy, high cost).
Pretraining Data	Determines if the model was trained on relevant data.	General-purpose (Wikipedia, Books), domain-specific (medical, legal, finance).
Fine-Tuning Availability	Whether the model allows domain-specific improvements.	Fine-tuning improves accuracy but requires additional data and resources.
License & Access	Defines whether the model is freely available or gated.	Open-source (MIT, Apache 2.0), restricted access (requires approval).
Inference Speed	Affects response time and feasibility for real-time applications.	Fast models for chatbots, slower models for batch processing.
Evaluation Metrics	Measures performance on standard datasets.	Accuracy, F1 score, BLEU (translation), perplexity (language modeling).
Community Feedback	User experiences and support in the Hugging Face community.	High engagement suggests reliability and ease of troubleshooting.
Deployment Considerations	Ease of integration into applications.	Works with TensorFlow, PyTorch, FastAPI, ONNX, etc.
Hardware Requirements	Determines if the model fits your available compute resources.	Small models for CPU, large models for GPU/TPU.
Preprocessing & Tokenization	Ensures input data is formatted correctly.	WordPiece (BERT), Byte Pair Encoding (GPT), SentencePiece (T5).

- ### Local deployment requirements

Deploying an **LLM locally** requires **hardware readiness, optimized inference, containerization, MLOps automation, and real-time monitoring**. Using tools like **vLLM, Triton, Docker, FastAPI, MLflow, and LangChain** ensures a scalable and efficient workflow.

○ Inference optimization

Inference optimization refers to techniques and strategies used to improve the speed, efficiency, and resource utilization of machine learning models during inference (i.e., when the model is making predictions, rather than training). The goal is to **reduce latency, memory usage, and computational costs** while maintaining accuracy.

Technique	What It Does	Tools & Methods	Impact
Quantization 🔢	Reduces precision of model weights (e.g., FP32 → INT8/4-bit) to save memory and increase speed.	`bitsandbytes`, `GPTQ`, `AWQ`, TensorRT-LLM	Faster inference, lower RAM/GPU usage, slight accuracy drop.
KV Caching 📂	Stores past attention outputs to avoid recomputation in autoregressive models.	`vLLM`, FlashAttention, Native KV Cache in Mistral/LLaMA	**Drastically reduces latency** for long text generations.
Model Pruning ✂️	Removes unnecessary weights & neurons to shrink model size.	Hugging Face `optimum`, SparseGPT, L1/L2-based pruning	Smaller models, faster inference, but may lose some accuracy.
Model Distillation 🧑‍🏫	Transfers knowledge from a large "teacher" model to a smaller "student" model.	`DistilBERT`, `TinyLlama`, Knowledge Distillation frameworks	Keeps performance close to the original model with a **much smaller size**.
Tensor Parallelism 🖥️	Splits a model across multiple GPUs to handle larger models efficiently.	DeepSpeed, `FSDP`, `Megatron-LM`	**Speeds up inference** on multi-GPU setups.
Speculative Decoding 🚀	Uses a smaller model to predict multiple tokens, verified by a larger model.	`vLLM`, NVIDIA TensorRT-LLM	**Increases token generation speed** while maintaining quality.
Dynamic Batching 📊	Groups multiple requests together to improve efficiency.	`vLLM`, Triton Inference Server, Hugging Face `Text Generation Inference (TGI)`	**Improves throughput** for serving multiple users.
Hardware Acceleration ⚡	Uses optimized libraries and specialized hardware (TPUs, GPUs, NPUs).	CUDA, ROCm (AMD GPUs), ONNX Runtime, TensorRT	**Maximizes performance** on supported devices.

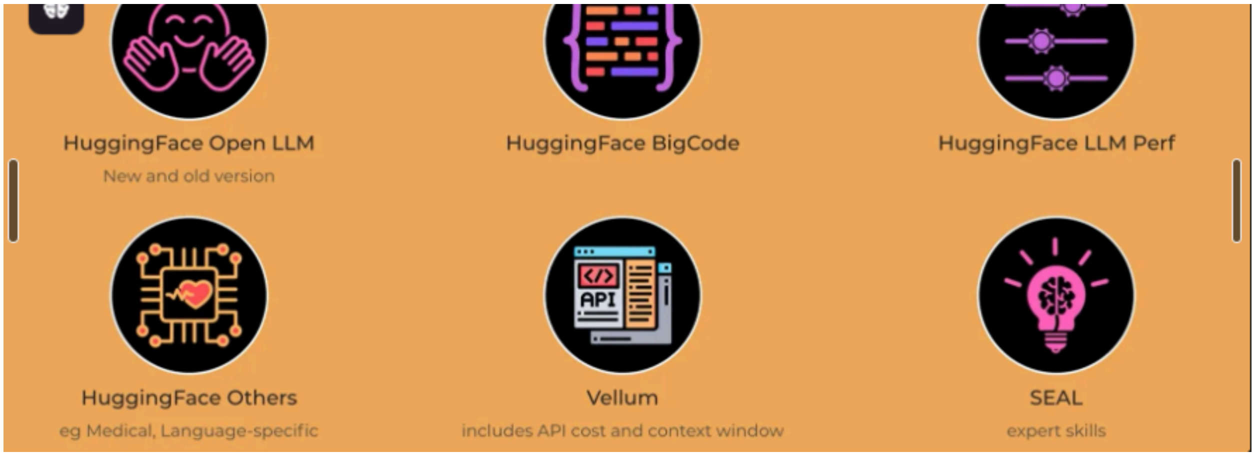
- Model quantization with Bits & Bytes

Advantage 💡	Impact 📈
🚀 Faster Inference	Quantized models require fewer computations, making them faster on consumer GPUs.
🖥️ Lower Memory Usage	Reduces VRAM requirements, allowing larger models to run on smaller GPUs (e.g., 4-bit LLaMA-2 on a 16GB GPU).
🔥 Minimal Accuracy Loss	Maintains performance while reducing precision using advanced quantization methods.
💸 Cost-Effective	Enables **local deployment** on lower-end GPUs instead of expensive cloud solutions.

Method	Description	Use Case
8-bit (INT8)	Balanced between speed & accuracy	Good for inference on most GPUs
NF4 (Normalized Float 4-bit)	Maintains high precision with minimal loss	Best for LLaMA-2, Mistral models
4-bit (Quantized FP16)	Extreme compression, lower accuracy trade-off	Used when GPU memory is very limited

5. LLM Evaluation and Comparison

- Online evaluation platforms
- ❖ **Chatbot Arena** – Developed by UC Berkeley, this platform allows you to compare AI chatbots side by side and ranks models based on user votes.
- ❖ LLM Arena – Enables direct comparison of multiple LLMs (2 to 10 models at a time) with interactive testing.
- ❖ **Vercel AI Playground** – Lets you test and compare various AI models like Llama 2, Claude, GPT-4, and Hugging Face models.
- ❖ **Hugging Face Open LLM Leaderboard** – Ranks open-source LLMs based on benchmarking metrics.
- ❖ **KDnuggets' AI Playgrounds** – Lists multiple free LLM comparison tools and playgrounds.



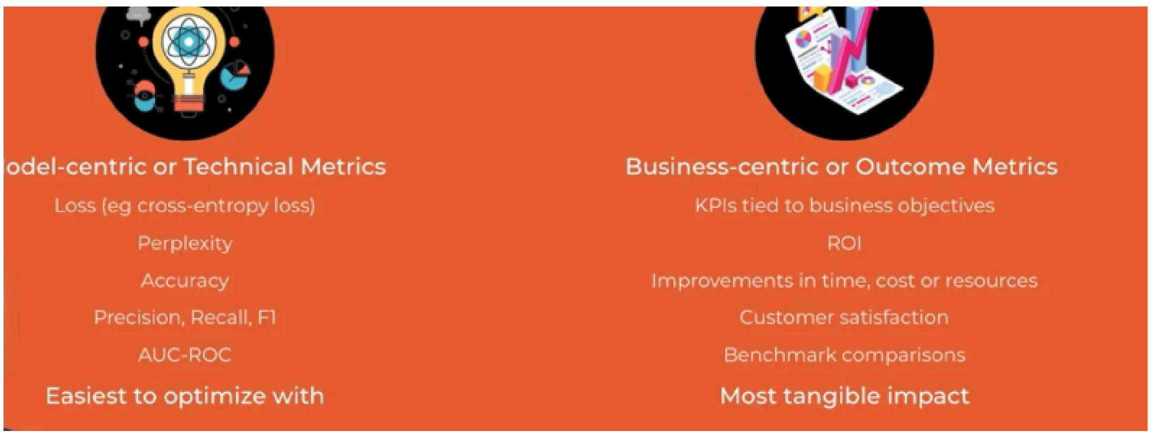
- Leading LLM benchmarking frameworks

ARC	Reasoning	A benchmark for evaluating scientific reasoning; multiple-choice questions
DROP	Language Comp	Distill details from text then add, count or sort
HellaSwag	Common Sense	"Harder Endings, Long Contexts and Low Shot Activities"
MMLU	Understanding	Factual recall, reasoning and problem solving across 57 subjects
TruthfulQA	Accuracy	Robustness in providing truthful replies in adversarial conditions
Winogrande	Context	Test the LLM understands context and resolves ambiguity
GSM8K	Math	Math and word problems taught in elementary and middle schools

ELO	Chat	Results from head-to-head face-offs with other LLMs, as with ELO in Chess
HumanEval	Python Coding	164 problems writing code based on docstrings
MultiPL-E	Broader Coding	Translation of HumanEval to 18 programming languages

GPQA	Graduate Tests	448 expert questions; non-PhD humans score 34% even with web access
BBHard	Future Capabilities	204 tasks believed beyond capabilities of LLMs (no longer!)
Math Lv 5	Math	High-school level math competition problems
IFEval	Difficult instructions	Like, "write more than 400 words" and "mention AI at least 3 times"
MuSR	Multistep Soft Reasoning	Logical deduction, such as analyzing 1,000 word murder mystery and answering: "Who has means, motive and opportunity?"
[illegible]	[illegible]	A more advanced and cleaned up version of MMLU including choice of 10

❖ Performance metrics



❖ Prompt engineering principles

Principle/Technique	Description	Example	Best For
Clarity & Specificity	Be clear and direct in your prompts to avoid ambiguity.	*"Explain the difference between generative AI and traditional AI in simple terms."*	Factual queries
Contextual Information	Provide relevant background to guide the model's responses.	*"You are a professor explaining quantum entanglement."*	Complex topics or when context is important
Step-by-Step Instruction	Encourage structured reasoning with explicit step-by-step requests.	*"Explain the process of training a neural network step by step."*	Logical reasoning or problem-solving
Role Assignment	Assign a persona or expertise to enhance model responses.	*"You are an expert data scientist. Explain preprocessing imbalanced datasets in Python."*	Technical explanations, expert advice
Output Formatting Guidance	Specify the format of the response (e.g., list, table, bullet points).	*"List the pros and cons of reinforcement learning in a table."*	Structured output (tables, bullet points)
Length Control	Specify the length of the output to avoid excessive or vague answers.	*"Summarize this research paper in 5 bullet points."*	Summarization or concise responses
Zero-Shot Prompting	Ask a direct question without prior examples or context.	*"What is the capital of France?"*	Direct, factual questions
Few-Shot Prompting	Provide a few examples before asking the main question.	*"Example 1: Input: 'fast', Output: 'quick' ... Now, Input: 'big', Output: ?"*	Tasks requiring consistency (e.g., classification)
Chain-of-Thought (CoT)	Request step-by-step reasoning for complex questions or tasks.	*"Solve this math problem and explain your reasoning: 24 × 17 = ?"*	Logical reasoning, complex tasks
Self-Consistency	Run multiple prompt variations and select the most consistent answer.	*"Generate multiple responses for this question and select the best one."*	Reducing hallucinations in open-ended queries
Prompt Chaining	Break down complex tasks into smaller prompts for easier management.	*"Step 1: Summarize this article... Step 2: Now rewrite the summary in simpler terms."*	Multi-step workflows

Contrastive Prompting	Ask for multiple perspectives or comparisons on a topic.	*"Explain the benefits and drawbacks of using transformers in NLP."*	Comparative analysis
Reinforcement Prompting	Request the model to critique and improve its own response.	*"Improve the clarity of this response and add more details."*	Iterative refinement of responses
Temperature Tuning	Control the randomness of the model's output by adjusting the temperature (lower = deterministic).	*"Generate a creative description of a sunset."* (low temp = factual, high temp = creative)	Control over creativity vs. factual output
Token Limits	Set a token (word or character) limit for the model's response to manage long or short outputs.	*"Provide a summary of this article in under 200 words."*	Managing response length

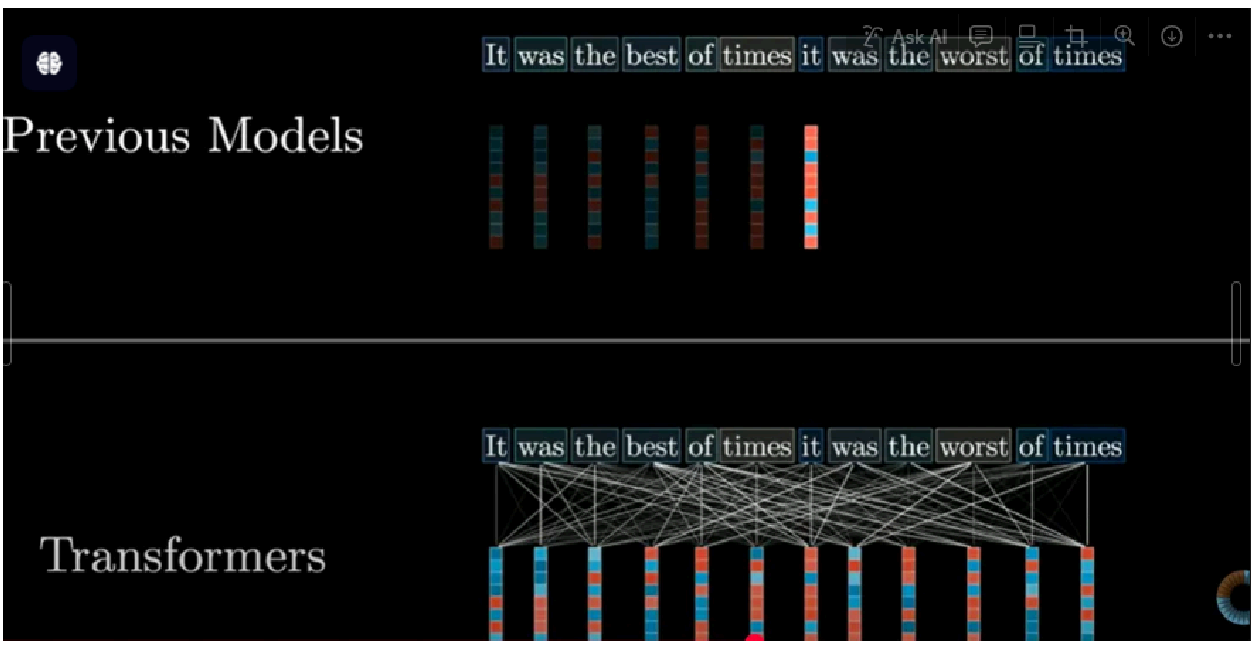
❖ Capability enhancement strategies

Strategy	Description	Examples/Applications
Model Fine-Tuning and Domain Adaptation	Customizing pre-trained models for specific domains or tasks by fine-tuning on domain-specific data or few-shot learning.	Fine-tuning GPT for legal or medical texts, domain-specific knowledge transfer.
Model Optimization for Efficiency	Techniques to make models more efficient in terms of computation and memory usage.	Model pruning, quantization, distillation, sparse models.
Scalability and Deployment	Strategies for efficient deployment and scaling of LLMs, including distributed training and cloud-based deployment.	Distributed training on TPUs/GPUs, cloud deployment, auto-scaling infrastructure.
Reducing Bias and Improving Fairness	Mitigating biases, ensuring fairness, and maintaining ethical standards in model outputs.	Bias mitigation algorithms, fairness constraints, diverse and inclusive training data.
Model Robustness and Safety	Techniques for improving model robustness against adversarial attacks and ensuring the safety of outputs.	Adversarial training, interpretability tools (e.g., LIME, SHAP), content moderation.

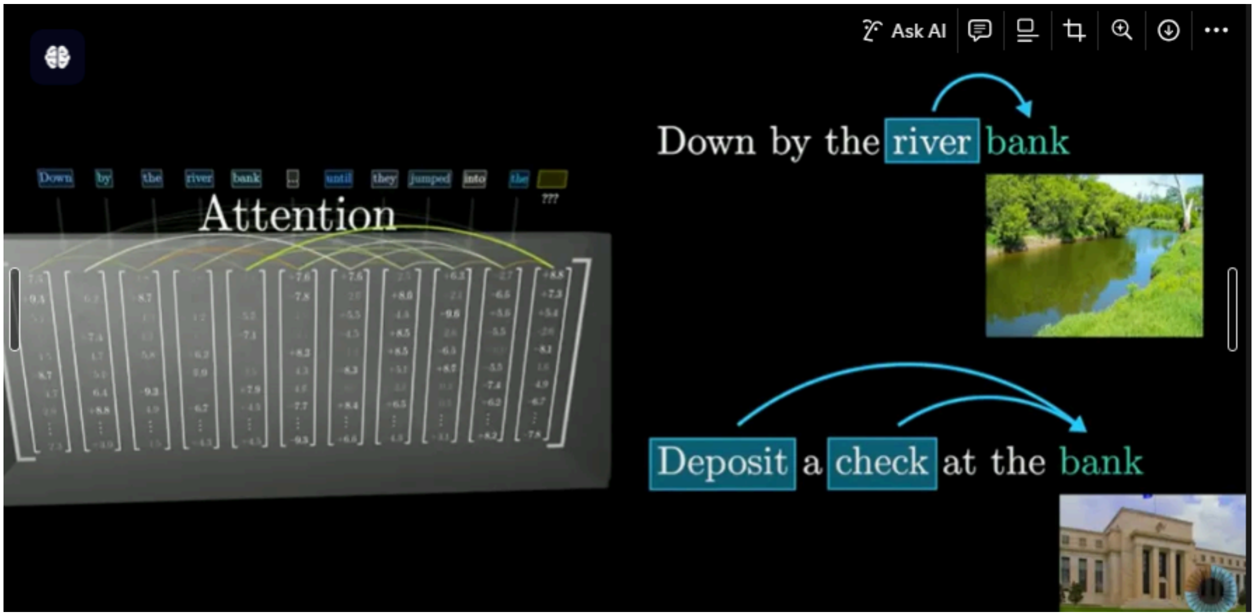
Interactive Learning and RLHF	Reinforcement learning from human feedback and active learning to improve model performance and alignment with user needs.	RLHF for fine-tuning generative models, active learning for efficient labeling.
Personalization and Customization	Adapting models to individual users or specific tasks by incorporating user preferences and contextual memory.	Personalized recommendations, user-specific responses, conversational memory.
Task-Specific Adaptation	Tailoring models to efficiently handle multiple or specific tasks using techniques like multi-task learning or zero-shot task transfer.	Multi-task learning (e.g., translation + summarization), zero-shot learning for diverse NLP tasks.

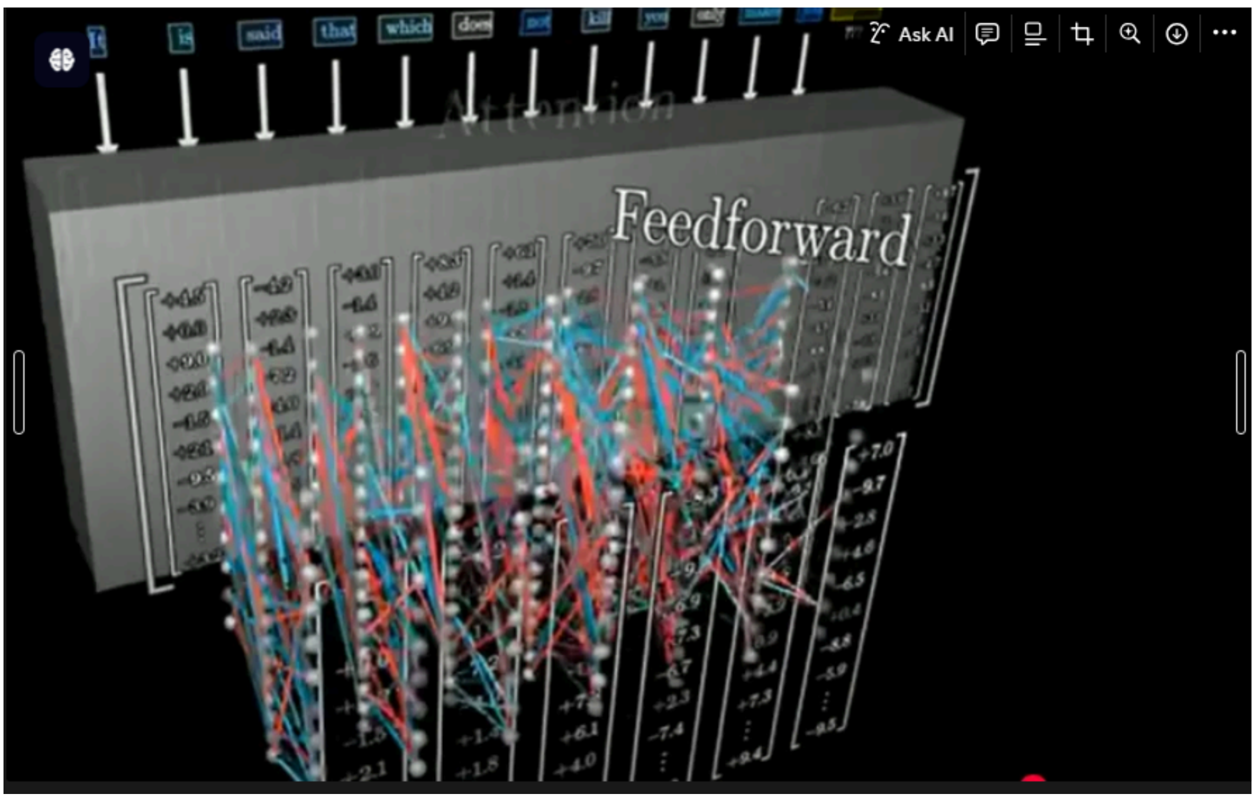
6. LLM Architecture

- **Transformer architecture fundamentals**



The model utilizes attention, allowing parallel processing of words, with tokenization influenced by context. Each word is encoded based on its surrounding context.





- ### Feedforward in Transformers:

In transformers, a **feedforward neural network** is applied to each position (or token) independently, after the self-attention mechanism processes the input. It operates in a **feedforward manner** meaning the data moves through the network in one direction, without any cycles or loops.

Self-Attention Mechanism:

- **Purpose**: To capture relationships and dependencies between different tokens (words or subwords) in the input sequence, regardless of their position in the sequence. It allows the model to focus on relevant parts of the input when processing each token.
- **Timing**: This is the **first step** in each transformer block. The input sequence is processed through the self-attention mechanism, where each token attends to every other token and creates a context-aware representation for each token.

Feedforward Layer:

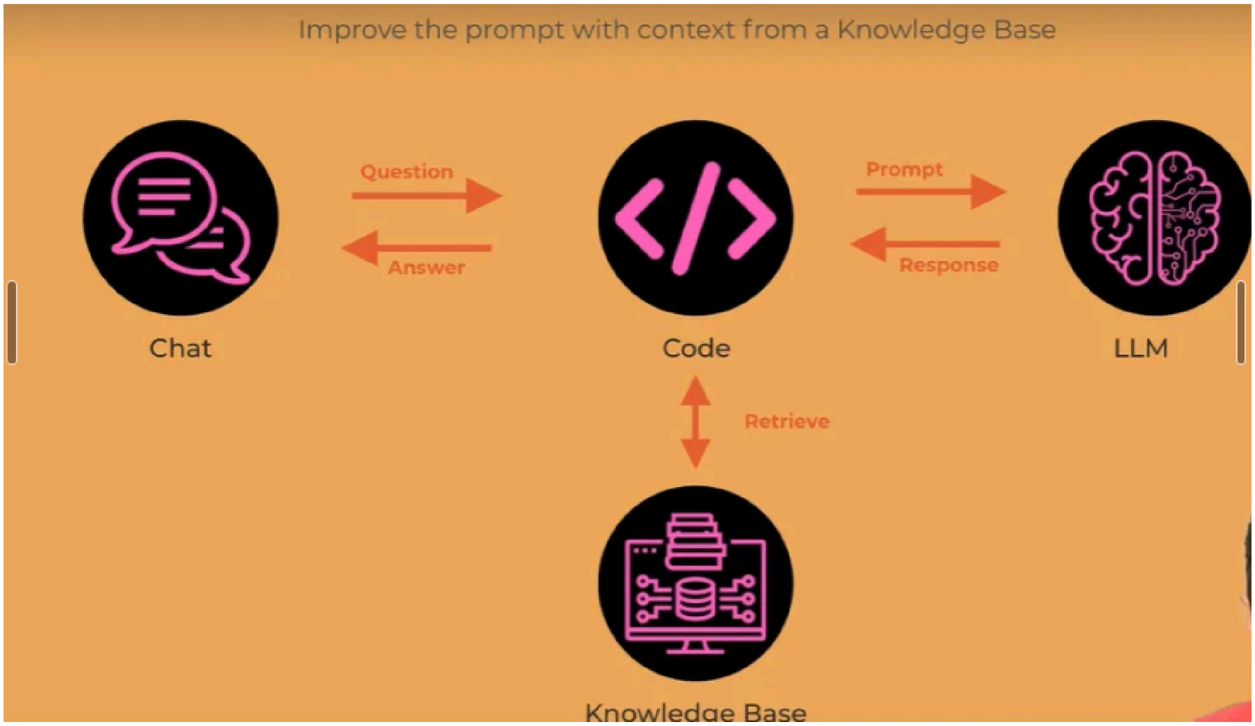
- **Purpose**: To process the output of the self-attention mechanism for each token independently, adding **non-linearity** and **depth** to the model. It helps the model learn more complex patterns.
- **Timing**: This comes **after the self-attention** in each transformer block. The output of the attention mechanism is passed through a feedforward network (which typically includes two linear layers with an activation function in between) to further refine the token representations.

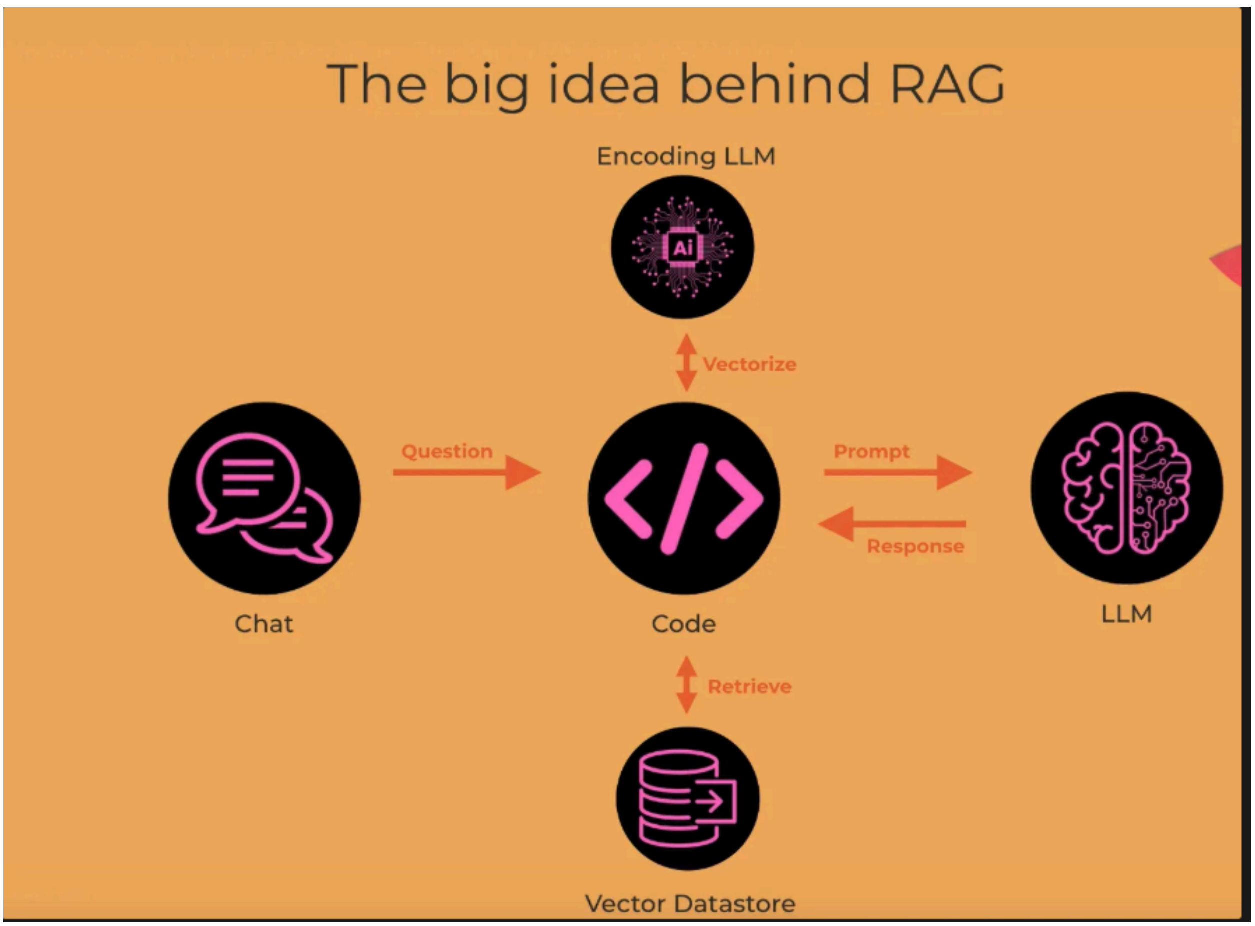
Multi-Layer Perceptron (MLP):

- **Purpose**: In transformers, the MLP is often used as the **feedforward network** itself, so they perform the same task. It provides an additional layer of processing to learn complex features after attention has been applied.
- **Timing**: The MLP comes **after the self-attention mechanism** in each transformer block, operating on the output of self-attention to process the data more thoroughly before passing it to the next block.

7. Building RAG (Retrieval-Augmented Generation) Applications

- RAG fundamentals and concepts


Improve the prompt with context from a Knowledge Base
Question
Answer
Prompt
Response
Chat
Code
LLM
Retrieve
Knowledge Base


The big idea behind RAG
Encoding LLM
Ai
Vectorize
Question
Prompt
Response
Chat
Code
LLM
Retrieve
Vector Datastore

- Vector Databases
  - **Text embedding methods**

Embedding Method	Definition	Use Cases
Word2Vec	A shallow neural network model that maps words to vectors in a continuous vector space.	- Word similarity tasks. - Text classification. - Named entity recognition.
GloVe	A model based on factorizing the word co-occurrence matrix to create word vectors.	- Semantic analysis. - Document similarity. - Word analogy tasks.
BERT	A transformer-based model that creates contextual embeddings for each word based on its surrounding context.	- Contextual word embeddings. - Sentiment analysis. - Question answering.
Sentence-BERT (SBERT)	A modification of BERT that generates sentence-level embeddings for similarity and clustering tasks.	- Text similarity. - Clustering and search.

  - Text splitting strategies

Strategy	Definition	Use Cases
Sentence Splitting	Breaking text into individual sentences, maintaining context while reducing the processing size.	- Named entity recognition. - Sentiment analysis. - Search optimization.
Word Splitting	Splitting text at the word level, often done in tokenization.	- Text classification. - Topic modeling.
Paragraph Splitting	Dividing text into paragraphs, keeping longer context while separating content logically.	- Document classification. - Summarization tasks.
Chunk-based Splitting	Breaking large text into semantic "chunks" or sections, typically based on natural boundaries.	- Text summarization. - Document retrieval.

- ○ Popular vector databases
  - ChromaDB

https://docs.trychroma.com/docs/overview/introduction

  - FAISS

https://ai.meta.com/tools/faiss/

  - Pinecone

https://www.pinecone.io/

  - Cassandra

https://cassandra.apache.org/_/index.html

Database	Definition	Key Features
ChromaDB	An open-source vector database designed to handle large-scale AI/ML and embeddings-based search.	- Real-time indexing and querying. - Scalable storage. - Optimized for similarity search and embedding-based retrieval.
FAISS	Facebook's AI Similarity Search library, designed for efficient similarity search and clustering of high-dimensional vectors.	- High-speed search. - Optimized for large-scale similarity searches. - Supports both CPU and GPU.
Pinecone	A managed vector database platform optimized for similarity search, often used for LLM applications.	- Highly scalable. - Fully managed solution. - Integrates easily with machine learning workflows.
Cassandra	A highly scalable NoSQL database known for handling large amounts of data across multiple servers.	- High availability and fault tolerance. - Suitable for high-volume, low-latency operations. - Supports wide-column data model.

- LangChain Framework

Aspect	LangGraph	LangChain	LangSmith
Definition	A graph-based approach to represent and model language with nodes (words, sentences, concepts) and edges (relationships, dependencies).	A framework to build applications with LLMs, allowing for chaining multiple LLM calls and integrating external tools.	A tool for managing and monitoring the behavior of LLMs, focusing on tracking, debugging, and improving model outputs.
Purpose	To enhance language understanding by leveraging graph-based relationships for reasoning tasks.	To make it easier to develop complex NLP applications using large language models by providing a structured pipeline and toolset.	To monitor, evaluate, and improve the performance of LLMs, focusing on ensuring better control and transparency.
Core Use Cases	- Semantic relationship representation - Structured NLP tasks like reasoning, document summarization, and QA	- Multi-step workflows with LLMs - Integrating external APIs, databases, and tools - Building robust LLM-powered applications	- Debugging and monitoring LLM outputs - Ensuring consistent model behavior - Tracking outputs and improving accuracy
Key Features	- Graph-based representation - Dependency modeling - Semantic relationships and reasoning	- LLM chaining - External tool integration - Long-term memory management	- Tracking model outputs - Monitoring workflows - Performance analysis and debugging
Integration Focus	Primarily focused on graph theory and relationships in language.	Focused on building pipelines for LLM applications, chaining calls, and interacting with APIs.	Focused on managing and improving the interaction and performance of LLMs.
Technology Type	Graph-based reasoning and knowledge representation.	Language model orchestration, workflow automation.	LLM monitoring and debugging.
Example Use Case	Using graphs for QA tasks where relationships between concepts in the text are essential.	Building an AI-powered assistant that queries APIs, processes data, and interacts with users in a multi-step process.	Tracking the response quality of an AI assistant over time to ensure consistency and improvement.
Popular For	Graph-based modeling and reasoning.	Automating workflows with LLMs and external tools.	Debugging and monitoring LLM behaviors.

https://www.langchain.com/langchain

- LangGraph

https://www.langchain.com/langgraph

- LangSmith

  https://www.langchain.com/langsmith

- MLOps integration

Concept	Definition	Key Characteristics
MLOps	The practice of combining machine learning and operations to automate the lifecycle of machine learning models, from development to deployment and monitoring.	- Automates model training, deployment, and monitoring. - Ensures reproducibility, scalability, and model management. - Uses tools like Docker, Kubernetes, CI/CD pipelines. - Involves monitoring models in production for performance and drift.
MLOps Integration	The integration of MLOps principles into the workflow of AI/ML systems to streamline and improve the efficiency of model development and deployment.	- Facilitates continuous integration and continuous delivery (CI/CD) for models. - Incorporates automated testing and version control for models. - Uses platforms like MLflow, Kubeflow, or SageMaker for model management and deployment. - Ensures collaboration between data scientists, engineers, and operations teams.

- Embedding visualization with t-SNE

Concept	Definition	Key Characteristics
Embedding Visualization with t-SNE	t-SNE (t-Distributed Stochastic Neighbor Embedding) is a dimensionality reduction technique used to visualize high-dimensional embeddings in lower-dimensional space (2D or 3D), often used in machine learning to understand how data points relate to each other.	- Effective for visualizing high-dimensional data (e.g., word embeddings, document embeddings). - Non-linear dimensionality reduction. - Useful for clustering and understanding relationships between data points. - Reduces large datasets to interpretable visual representations. - Commonly used for model introspection and exploring semantic similarity in NLP embeddings.

- Troubleshooting RAG systems

Concept	Definition	Key Characteristics
Troubleshooting RAG Systems	The process of diagnosing and fixing issues in Retrieval-Augmented Generation (RAG) pipelines, ensuring that the retriever and generator work together smoothly and efficiently to produce accurate outputs.	- **Retriever Issues:** Ensure that the retriever is fetching the right documents or passages (check embedding quality, search algorithms). - **Generation Issues:** Ensure that the generator uses retrieved information correctly (check model fine-tuning, tokenization issues). - **Latency Issues:** Check both retrieval and generation speeds (optimize retrieval algorithm, model inference time). - **Accuracy Issues:** Validate if the generated answers are relevant and coherent (check the retrieval context, model's understanding).

		model's understanding). - **Data Mismatch:** Ensure consistency between the type of data used for training the retriever and generator (semantic alignment). - **Logging & Debugging:** Use detailed logs to track both the retriever and generator outputs, which helps isolate the part of the system causing issues.