Mahavir Education Trust's

Shah and Anchor Kutchhi Engineering College

(An Autonomous Institute Affiliated to University of Mumbai)

Organnized by
Information Technology Department

HARNESSING GENERATIVE AI

Revolutionizing for better future

Nitin Kukreja, NPCI

12th April 2025

Information Technology Program Vision & Mission statements

Institute Vision

To become a globally recognized Institution offering quality education and enhancing professional standards.

Institute Mission

To impart high-quality technical education to the students by providing an excellent academic environment, well-equipped laboratories, and training through motivated teachers.

Department Vision

To achieve excellent standards of quality education by keeping pace with rapidly changing technologies.
To create technical manpower of global standards with capabilities of accepting new challenges in Information Technology.

Department Mission

M1:To create competent and trained professionals in Information Technology who shall contribute towards the advancement of engineering, science and technology useful for the society.

M2: To impart quality and value based education to raise satisfaction level of all stakeholders.

M3:To generate technically sound professionals and entrepreneurs to become part of the industry and research organizations at national levels.

PSOs and PEOs

Program Specific Outcomes(PSOs)

1. The Information Technology graduates are able to analyse, design, develop, test and apply management principles, mathematical foundations in the development of IT based solutions for real world and open-ended problems.

1. The Information Technology graduates are able to perform various roles in creating innovative career paths: to be an entrepreneur, a successful professional, pursue higher studies with realization of moral values & ethics.

Program Educational Objectives (PEOs)

1. A graduate will excel in professional career and contribute to social needs through Information Technology.
2. A graduate will be integrated into the world of practicing professionals for collaborations, pursue higher education, conduct research, demonstrate professionalism and ethics.
3. A graduate will Exhibit innovation, team work, leadership and communication skills through lifelong learning.

Harnessing Generative AI

Revolutionizing for better future

Nitin Kukreja
Data Scientist -NPCI

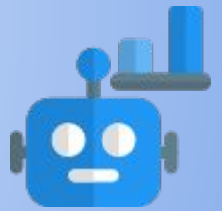
Quick Glance into AI

What is AI?

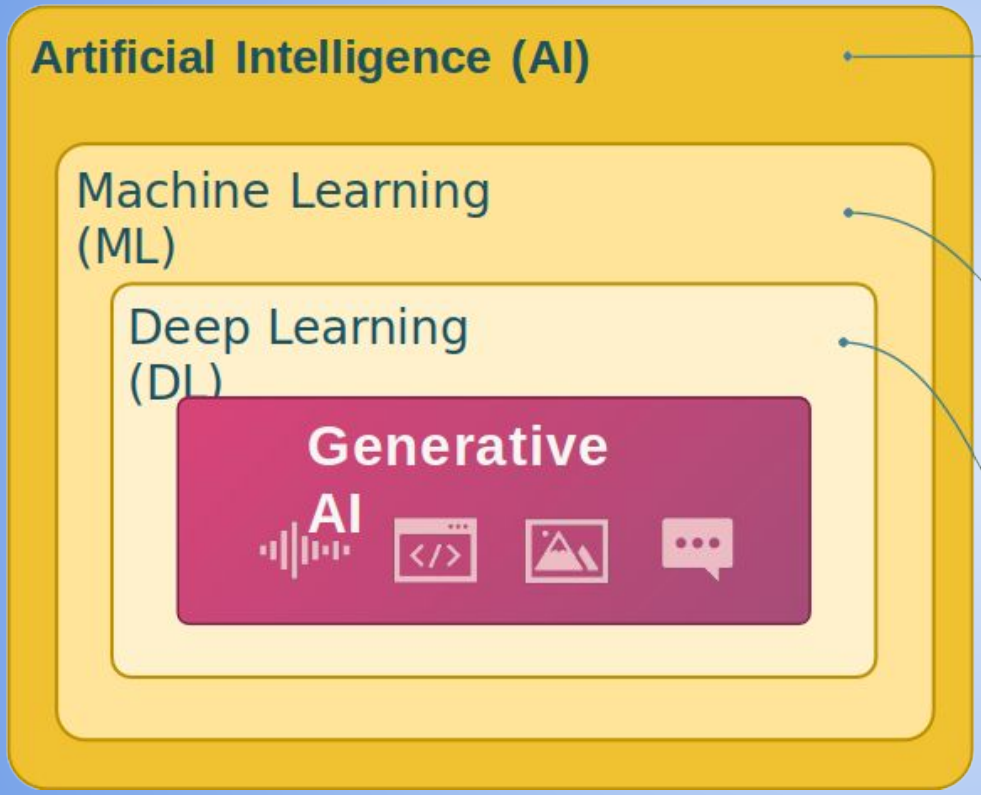
AI is a computer system that can be smart in a way similar to humans, intelligence agents know as Abilities

Think & Solve Problems

Improve their knowledge and skills from experience



Make decisions and take actions independently



Machine Learning:

Learn from existing data and make predictions/prediction without being explicitly programmed.

ML vs AI


AI
≠
ML
Artificial Intelligence
is a discipline
Machine Learning
Subfield of AI

Discriminative vs. Generative AI

Discriminative AI

Goal: Classify data or predict an outcome. Learns the boundary *between* classes.

Examples: Spam detection (spam/not spam), image classification (cat/dog), sentiment analysis (positive/negative).

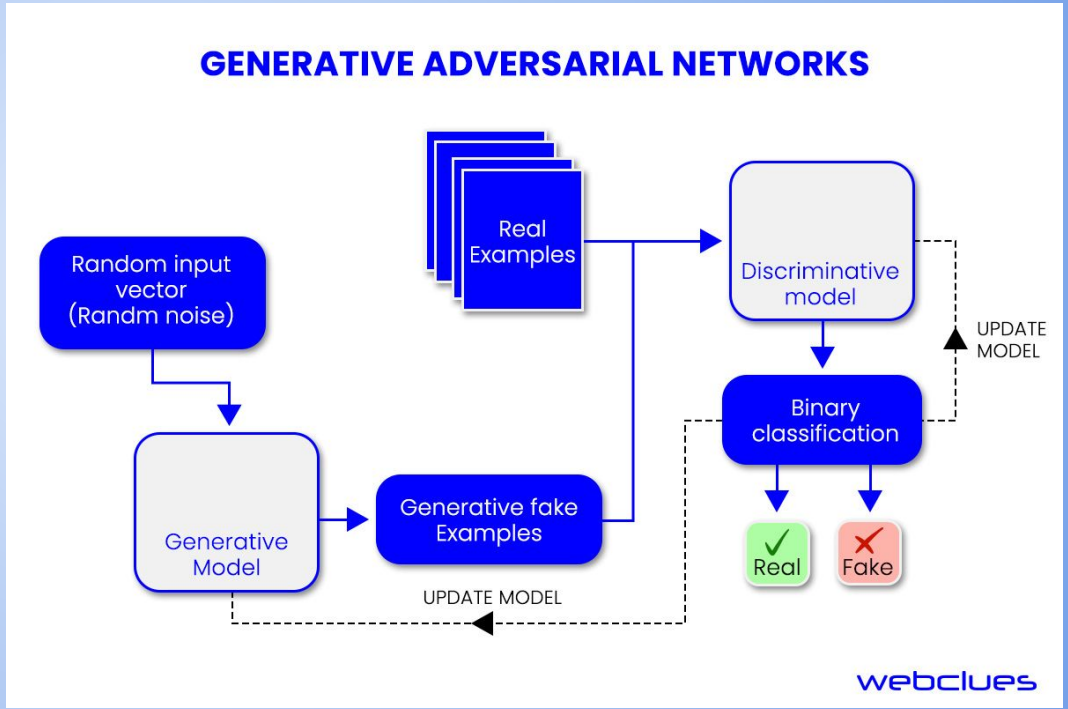
Asks: "Is this A or B?"

Generative AI

Goal: Create *new* data samples similar to the training data. Learns the underlying *distribution* of the data.

Examples: Generating text (this presentation!), images (DALL-E), music, code.

Asks: "Generate something like A."



Top Gen AI Use-Cases

Customer Service

AI chatbots reduce support costs and increase resolution speed.

Content Creation

Marketing teams produce more content faster using Gen AI.

Code Development

Developers improve productivity with AI coding assistants.

Product Design

Design teams accelerate iteration cycles using generative tools.

Personalized Experiences

Businesses achieve higher customer engagement through AI.

Deep Dive into LLMs

LLMs are not just hype — they’re changing the AI game.

> Generative AI & LLMs are a once-in-a-generation shift in technology.

What are LLMs?

An LLM is an advanced AI system designed to process and generate natural language.
Built using deep neural networks, trained on massive datasets of human text.
They understand, generate, and respond to human-like text, making machines feel more intuitive and responsive.

Capabilities

- ✍️ **Text Generation**
- 🌍 **Machine Translation**
- ❓ **Question Answering**
- 📝 **Summarization**
- 🤔 **Reasoning**

Why LLMs Matter

> “These technologies can assist, complement, empower, and inspire—across almost any domain.”

LLMs have broad potential in industries like healthcare, education, law, and customer service.
They enable more natural interactions between humans and machines.

Why Language Models Are Important

1. Massive Performance Leap & Capability

- Deliver impressive performance gains over previous state-of-the-art NLP models.
- Excel at complex tasks requiring understanding, reasoning, and generation (e.g., Q&A, complex problem-solving).

2. Broad Versatility & New Applications

- Effective across tasks like Translation, Code Generation, Text Creation, Classification, Q&A, and more.
- Display emergent abilities—solving tasks they weren't explicitly trained on.

3. Adaptability & Customization

- **Fine-Tuning:** Efficiently adapt foundational models for specific tasks with minimal data.
- **Prompt Engineering:** Guide models precisely with structured prompts and parameters.

> LLMs mark a fundamental shift—enabling more sophisticated human-computer interaction, automating cognitive tasks, and accelerating innovation across industries.

Understanding LLMs: The Foundational Transformer Architecture

The Core

- Modern LLMs rely on the Transformer architecture, introduced by Google in 2017 for machine translation.

What it Does

- A sequence-to-sequence model converting input sequences (like text) to output sequences (like translated text).

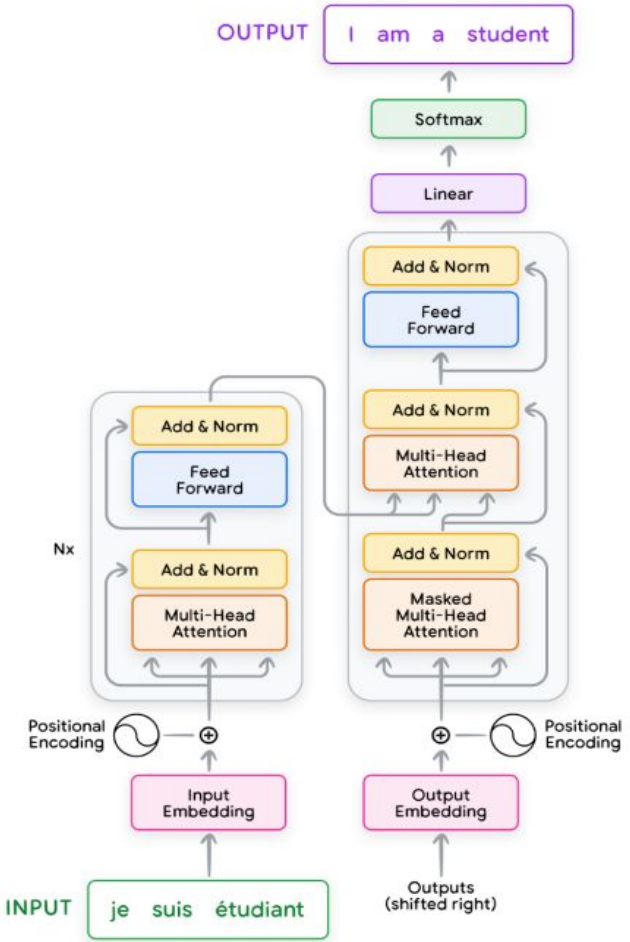
Key Components

- **Encoder:** Converts input text into rich numerical representations.
- **Decoder:** Autoregressively generates output text from encoded representations.

How it Works

- Composed of layers like Input Embeddings, Multi-Head Attention, Feed-Forward Networks, and Output Softmax.
- Each layer adds complexity and nuance to how the model understands and generates language.

> This architecture's mastery of sequence data laid the foundation for the LLM revolution—enabling scale, accuracy, and versatility.


OUTPUT
I am a student
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
INPUT
je suis étudiant
Outputs (shifted right)

Key Steps in Generating Input Embeddings

Normalization (Optional)

- Standardizes text by removing redundant whitespace, accents, etc.

Tokenization

- Breaks sentences into words or subwords.
- Maps tokens to integer token IDs from a vocabulary.

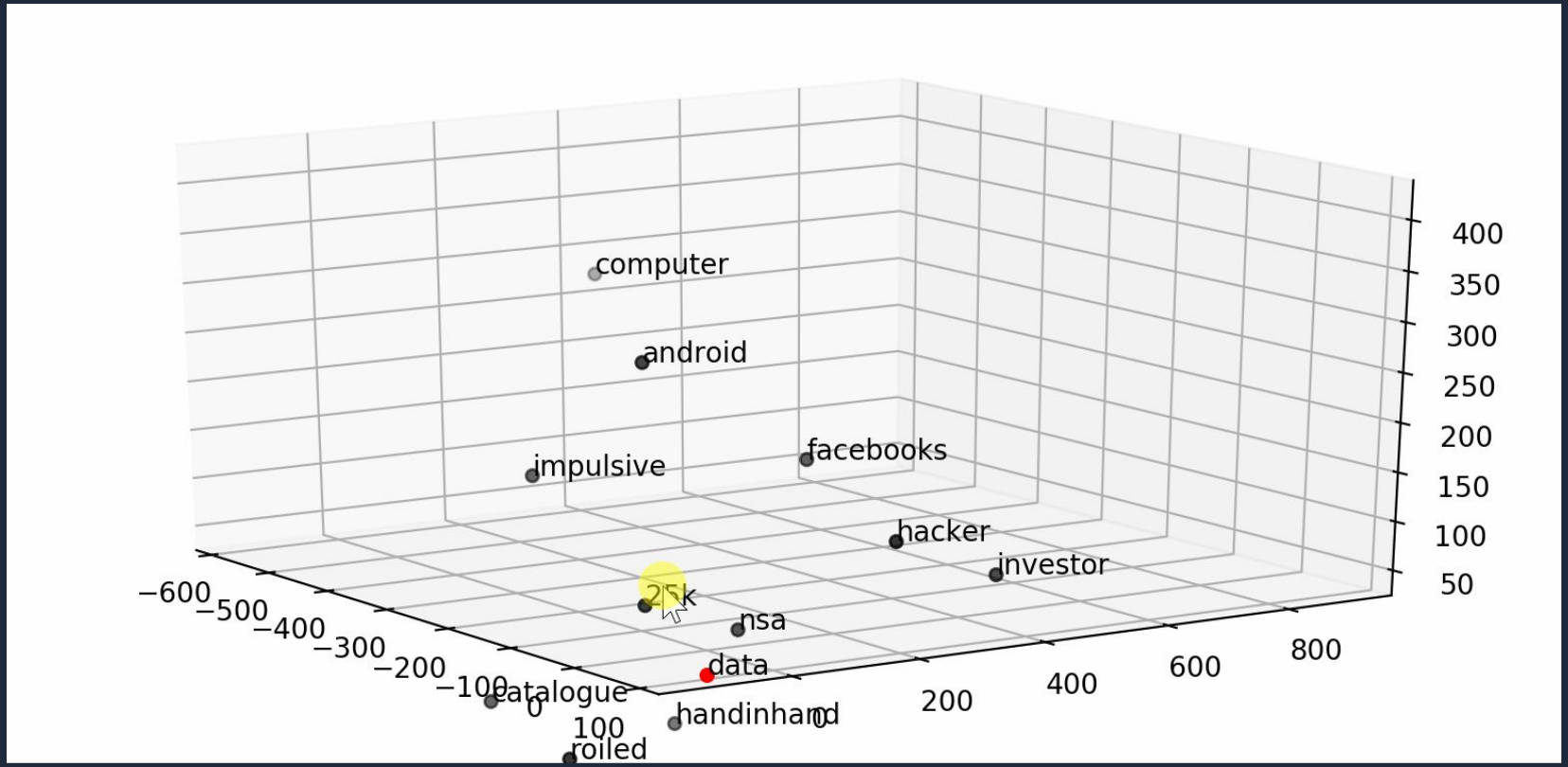
Embedding

- Converts token IDs into high-dimensional vectors (e.g., 68+ dimensions).
- Uses a lookup table, with vectors learned during training.

Positional Encoding

- Adds positional information to embeddings.
- Helps transformers understand token order in the sequence.

> These steps prepare input so that transformers can effectively interpret and process textual meaning.


computer
android
impulsive
facebooks
hacker
investor
nsa
data
handinhand
roiled
−600
−500
−400
−300
−200
−100
0
100
200
400
600
800
400
350
300
250
200
150
100
50

Tokenization

Embedding Layer

What is Self-Attention?

If you're wondering whether **self-attention** is similar to regular attention, the answer is **yes!** They share the same core concept and mathematical foundations.

A **self-attention module** takes in **n inputs** and returns **n outputs**. Each output is a weighted combination of all inputs, based on how much attention each token pays to the others.

In simple terms, it allows inputs to **interact with each other** ("self") and determine **who to focus on** ("attention"). These interactions generate outputs that are smart, context-aware summaries of the sequence.

Self-attention

input #1

1	0	1	0

input #2

0	2	0	2

input #3

1	1	1	1

Self-attention

input #1

1	0	1	0

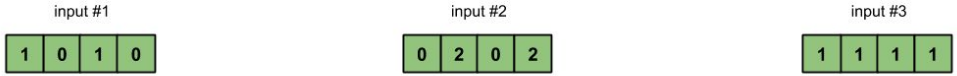
input #2

0	2	0	2

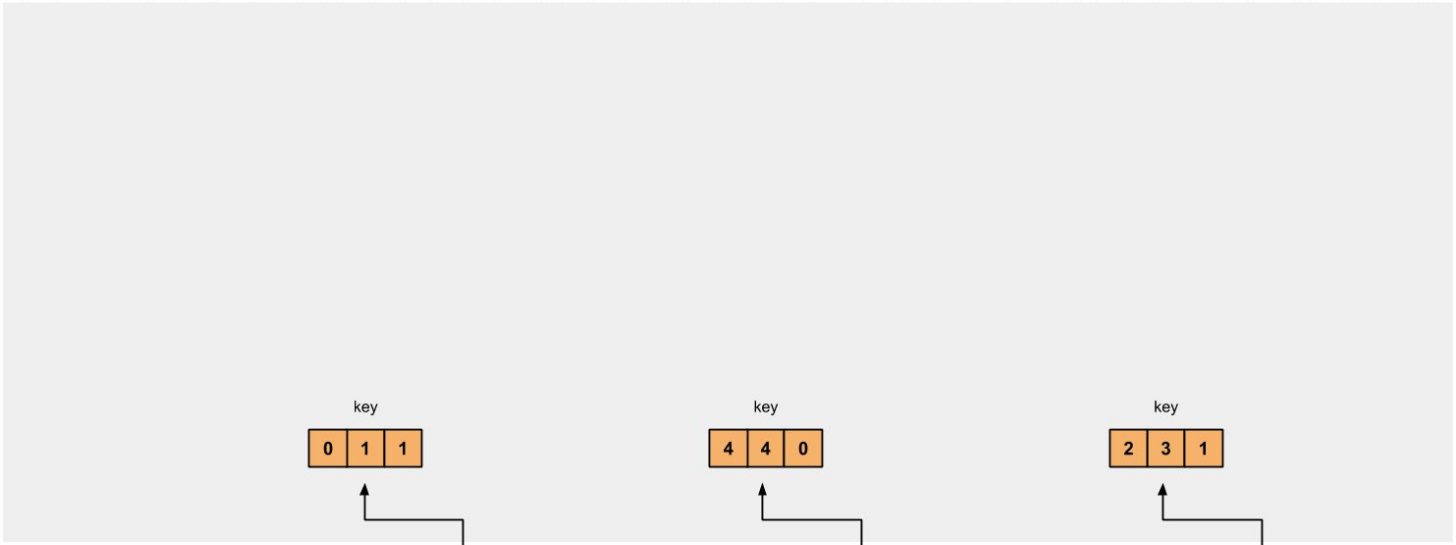
input #3

1	1	1	1

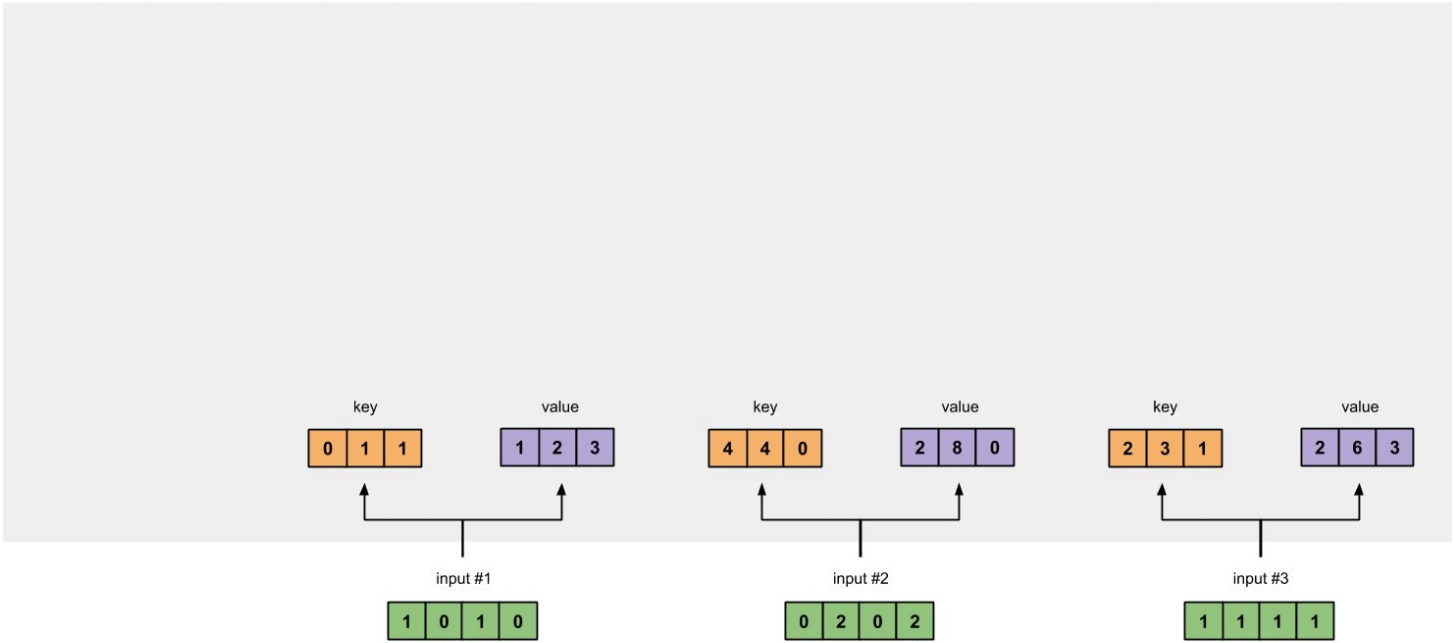
Self-attention



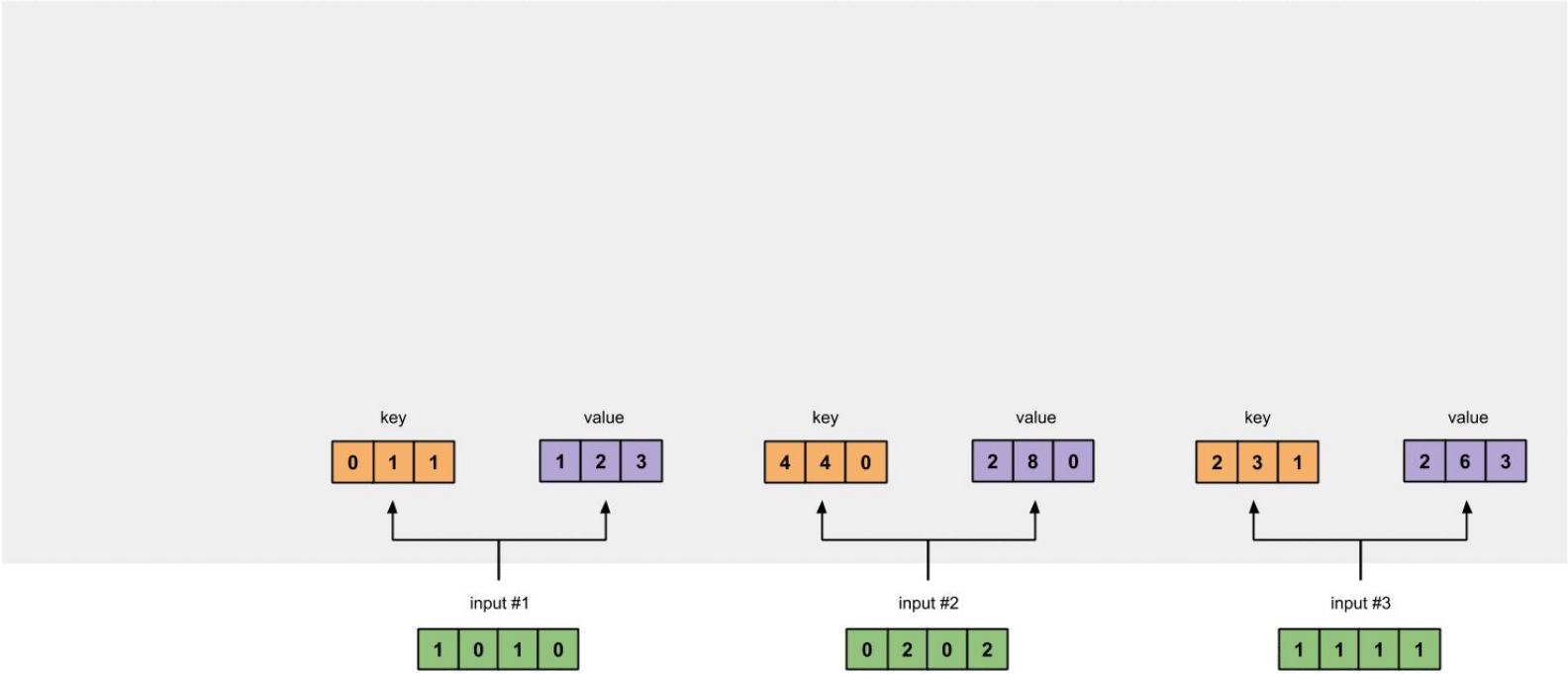
Self-attention



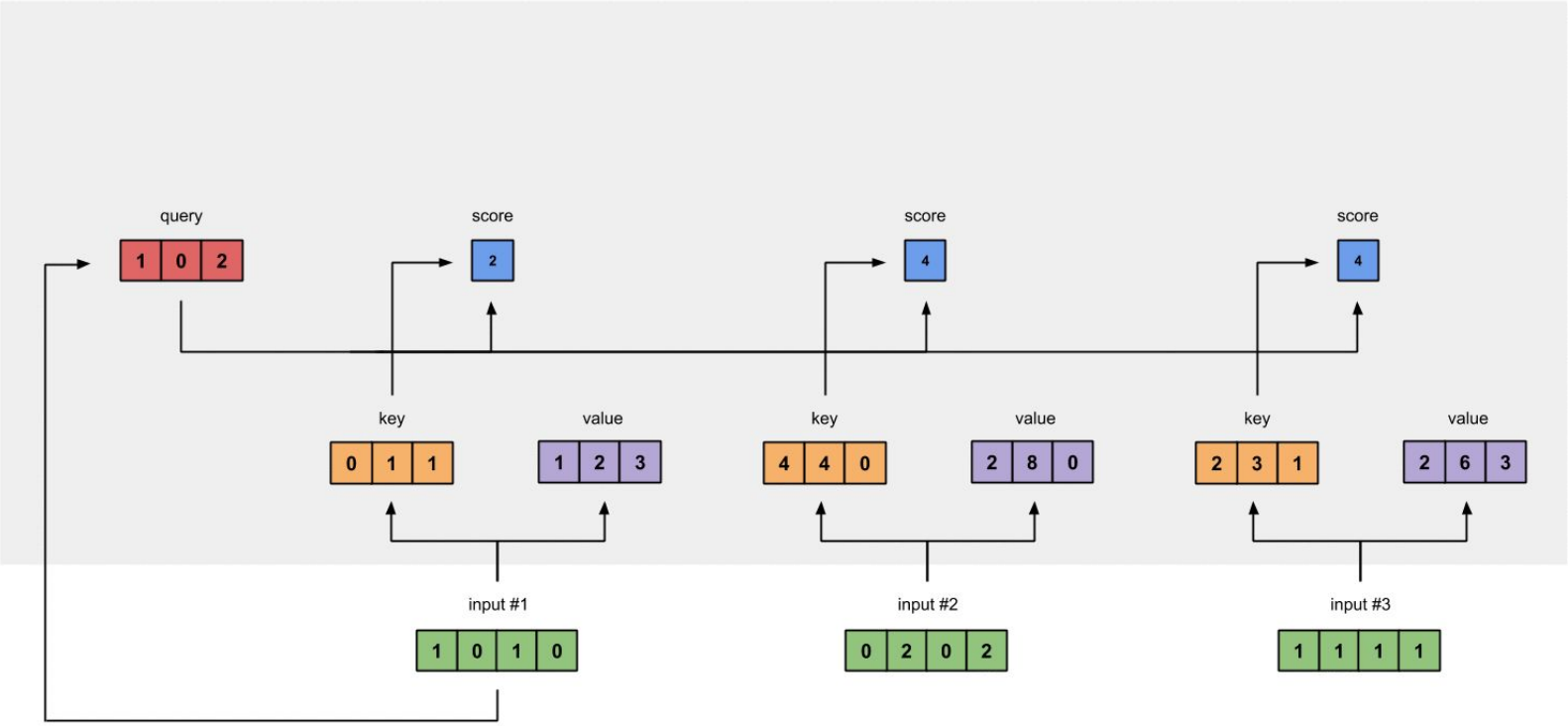
Self-attention



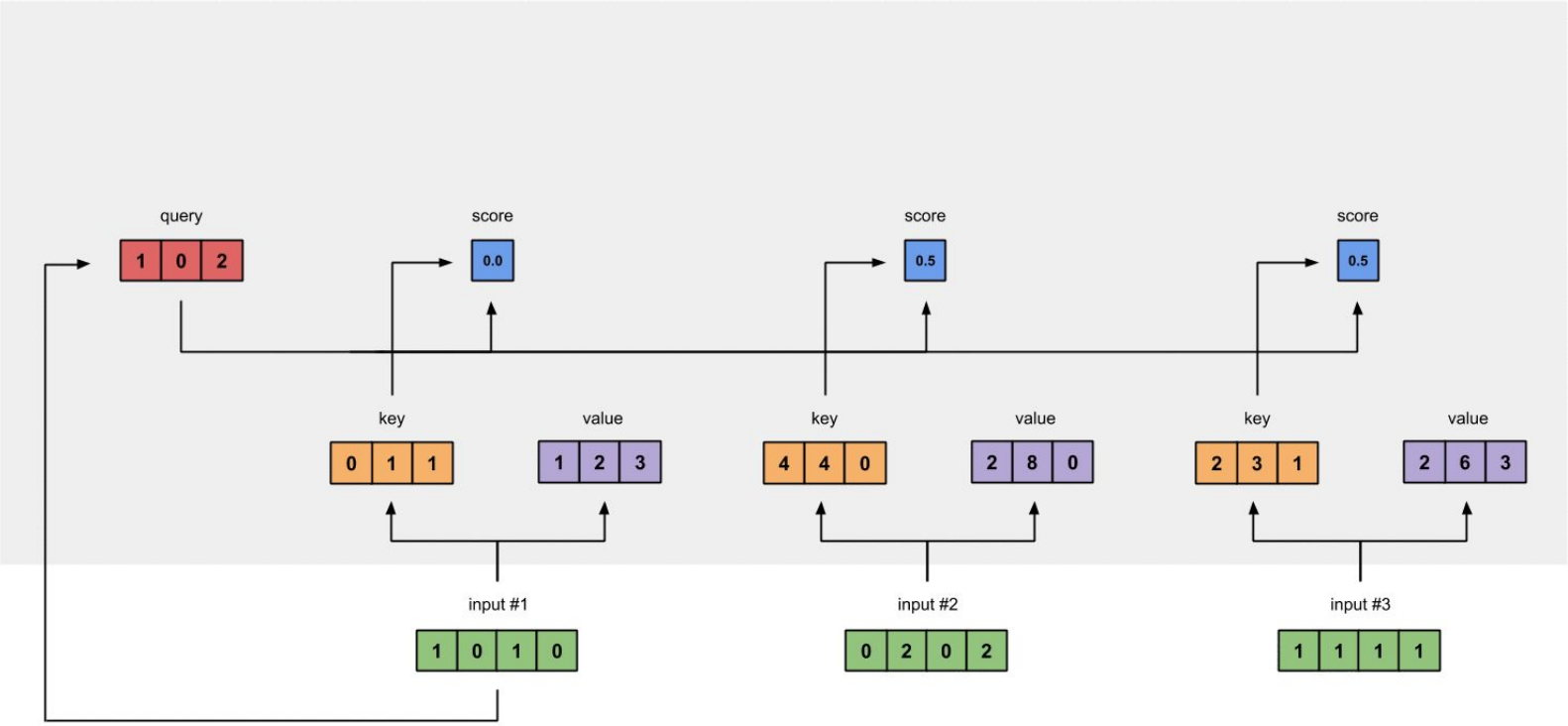
Self-attention



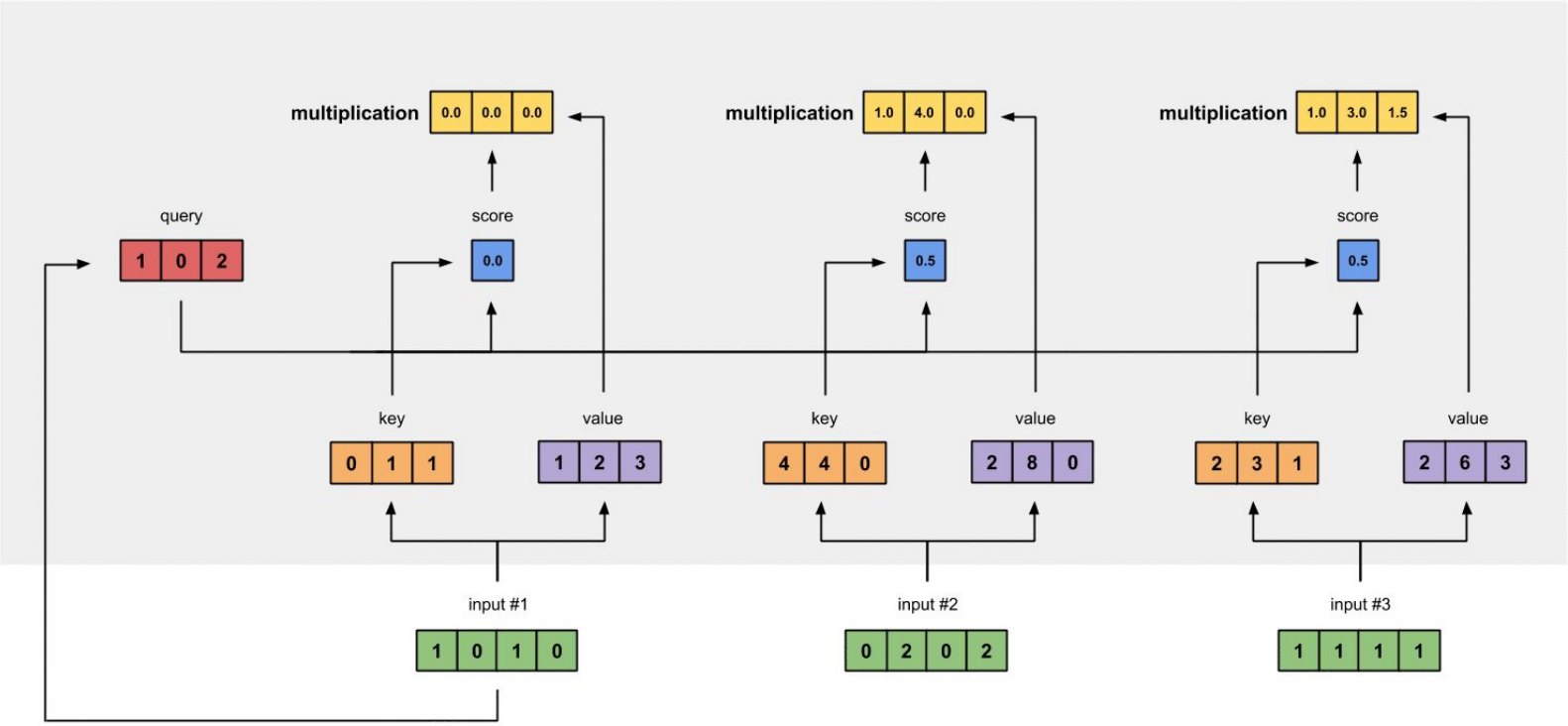
Self-attention



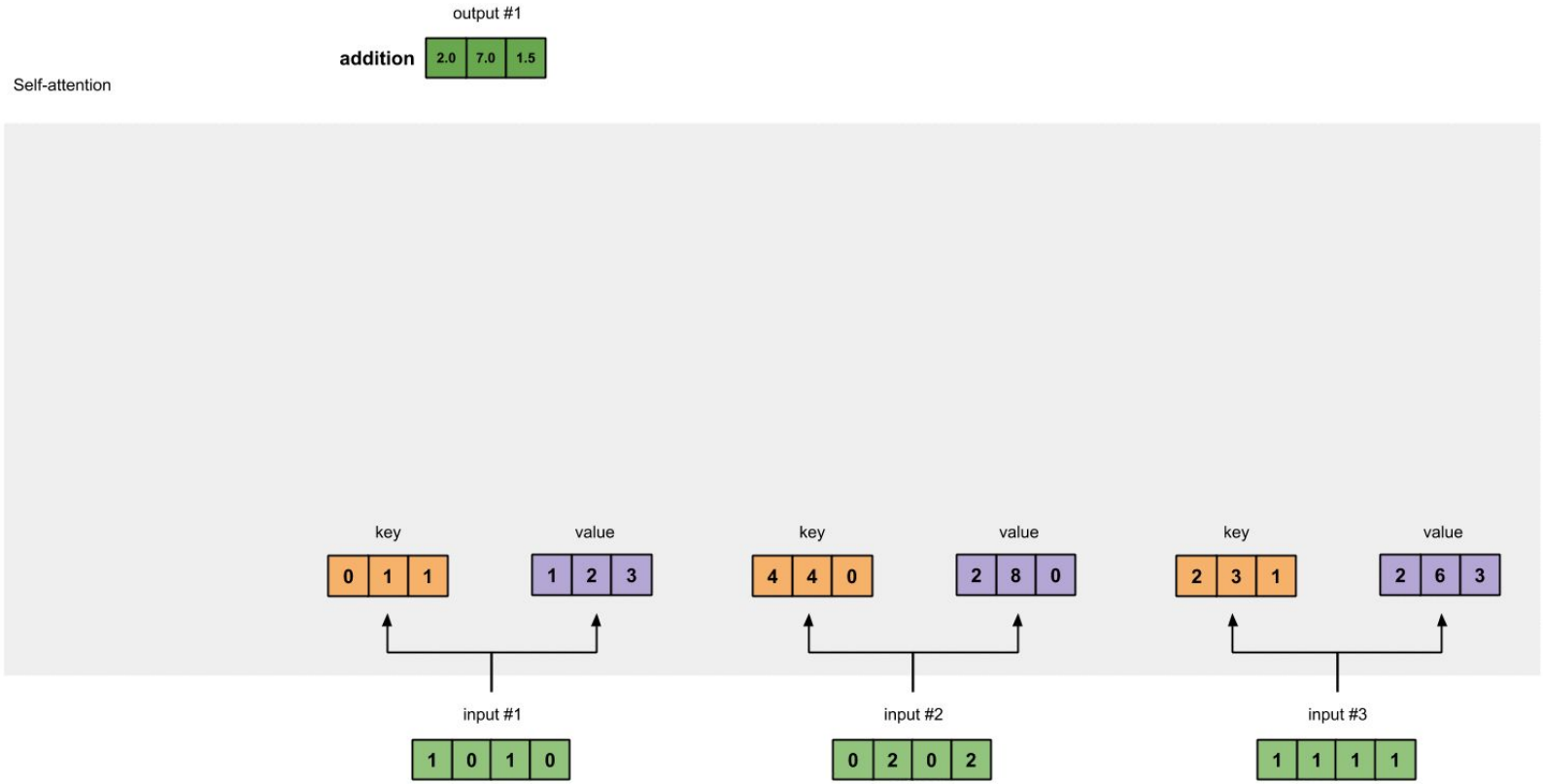
Self-attention



Self-attention



Self-attention

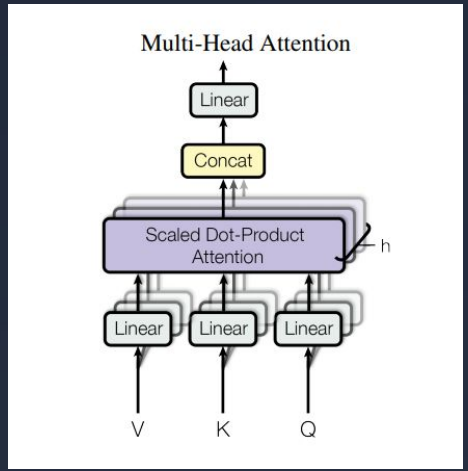


Multi-Head Attention

Instead of relying on a **single attention head**, modern transformer models use multiple heads by splitting the **Q (Query)**, **K (Key)**, and **V (Value)** matrices into several parts.

This allows the model to **jointly attend** to different positions in the sequence from different representational subspaces—capturing richer contextual relationships.

Each attention head operates on a lower-dimensional space. After computation, their outputs are concatenated and linearly transformed. This ensures that the **overall computational cost** remains comparable to using a single full-dimension head.



Position Encodings

In any language, the **order of the words** matters. Changing word positions can alter the entire meaning.

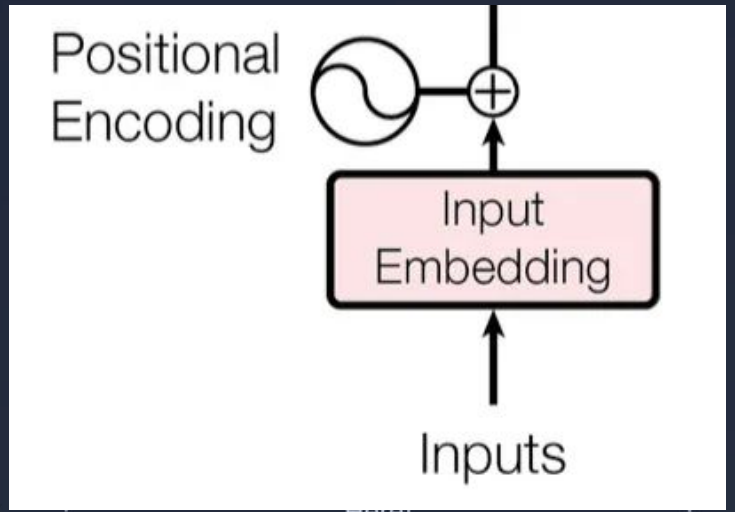
Problem: Transformer models don't use recurrence or convolution, so they lack a built-in sense of word order.

Solution: Positional encodings are added to word embeddings to inject position-based context into the model.

The original Transformer paper used **sine and cosine functions** based on token position to compute these encodings.

These are precomputed into a matrix of shape `max_length × embedding_dim`, where each row encodes position-specific features.

Note: These encodings are static and do not change during training.



$$PE_{(pos,2i)} = sin\left(pos/10000^{2i/d_{\text{model}}}\right)$$

$$PE_{(pos,2i+1)} = cos\left(pos/10000^{2i/d_{\text{model}}}\right)$$

Layer Normalization and Residual Connections

🔁 Residual Connections (Add)

- Acts like a shortcut — the input is added back to the output.
- ✅ Helps gradients flow better.
- ✅ Prevents forgetting or overwhelming important features.
- ✅ Makes learning easier and avoids vanishing/exploding gradients.

🎛️ Layer Normalization (Norm)

- Balances the output by normalizing activations across features.
- ✅ Reduces shifts in data distribution.
- ✅ Stabilizes and speeds up training.
- ✅ Leads to better model performance and convergence.

Feedforward Layer

🔄 What Happens After Attention?

- Once the model finishes focusing (via Multi-Head Attention and Add & Norm), data flows into the:

🧱 Feedforward Layer

- Applies transformations separately at each position in the sequence.
- Adds non-linearity and depth to the model's thinking.
- Helps the model learn complex patterns beyond attention.

🧬 Structure:

- 🔹 Linear → Activation (e.g., ReLU or GELU) → Linear
- Functions like a mini neural network inside each Transformer block.

✅ Why It's Useful:

- Gives the model more power to represent and understand language.
- Works identically across positions — ensuring consistency.
- Followed by another Add & Norm step for stable learning.

Encoder

Encoder Layer

Each encoder layer consists of two main sublayers:

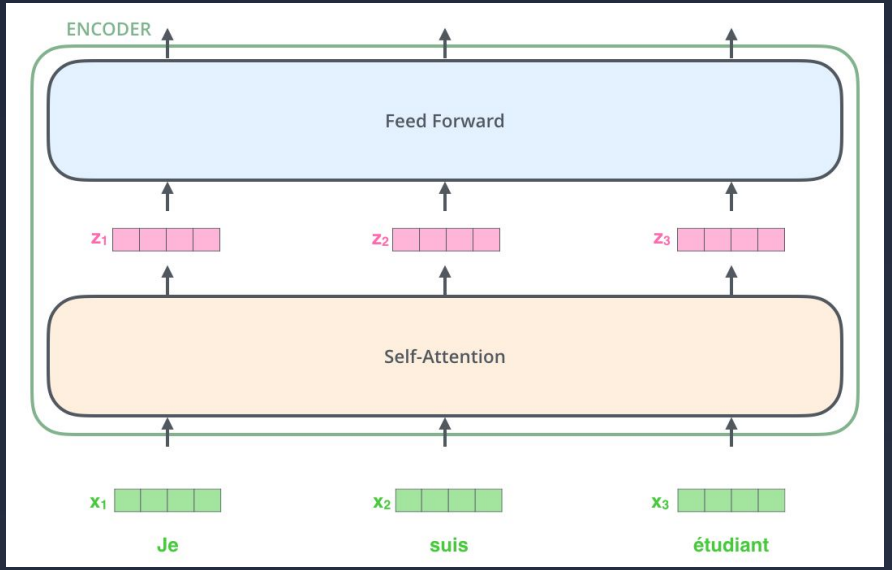
1. Multi-head Attention (with padding mask)
2. Point-wise Feed Forward Networks

Each sublayer is followed by:

- A **residual connection** that adds the input back to the output.
- **Layer normalization** to stabilize training.

The formula: `LayerNorm(x + Sublayer(x))` ensures better gradient flow and avoids vanishing gradients.

This process is repeated across **N encoder layers** in the Transformer.



Decoder Layer

The encoder processes the input and produces attention vectors **K** and **V**. These are passed to each decoder layer's encoder-decoder attention block to help focus on relevant parts of the input sequence.

Each decoder layer contains:

1. Masked Multi-head Attention (with look-ahead and padding mask)
2. Multi-head Attention (with padding mask):

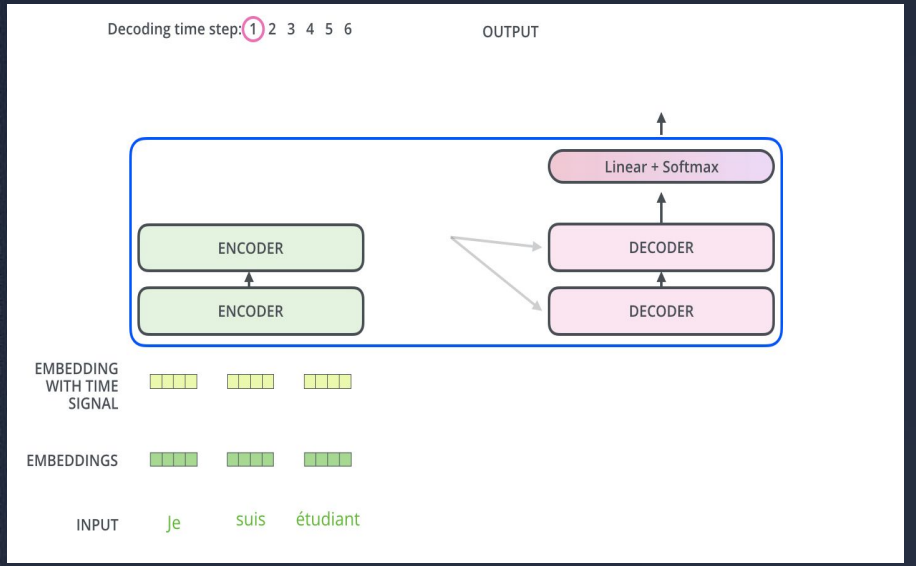
- **Q** (Query) receives output from the first attention block
- **K** and **V** come from the encoder output

3. Point-wise Feed Forward Network

Each sublayer has a **residual connection** and **layer normalization**:

```
LayerNorm(x + Sublayer(x))
```

There are **N decoder layers**. The model predicts the next word by attending to both the encoder output and its own prior output.



Decoding Phase

After finishing the encoding phase, we begin the decoding phase, where each step outputs a token (e.g., a word in the translation).

The steps continue until a special `<end>` token is generated. The output of each step is passed into the next decoder layer recursively.

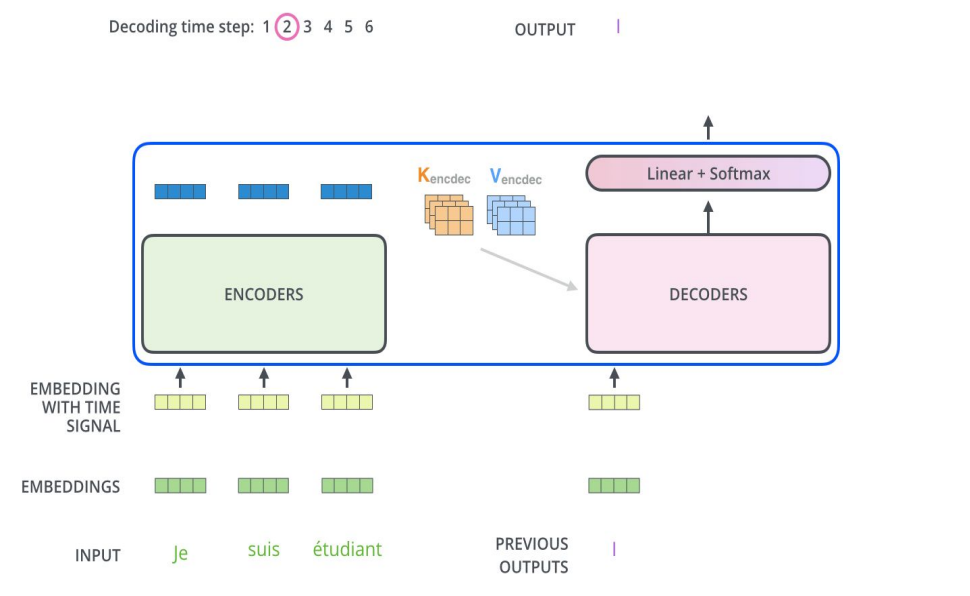
Like the encoder, decoder inputs are also embedded and added with positional encodings to signal token order.

The self-attention layers in the decoder differ from those in the encoder:

• They are masked to prevent the model from peeking into future tokens — ensuring only past and present positions are visible.

The Encoder-Decoder Attention layer:

• Uses queries from the decoder's prior layer.
• Keys and values come from the encoder's output.
• Helps align decoding with relevant encoded content.



Final Linear & Softmax Layer

After decoding, we get a vector of floats for each time step. But how do we turn this into a word?

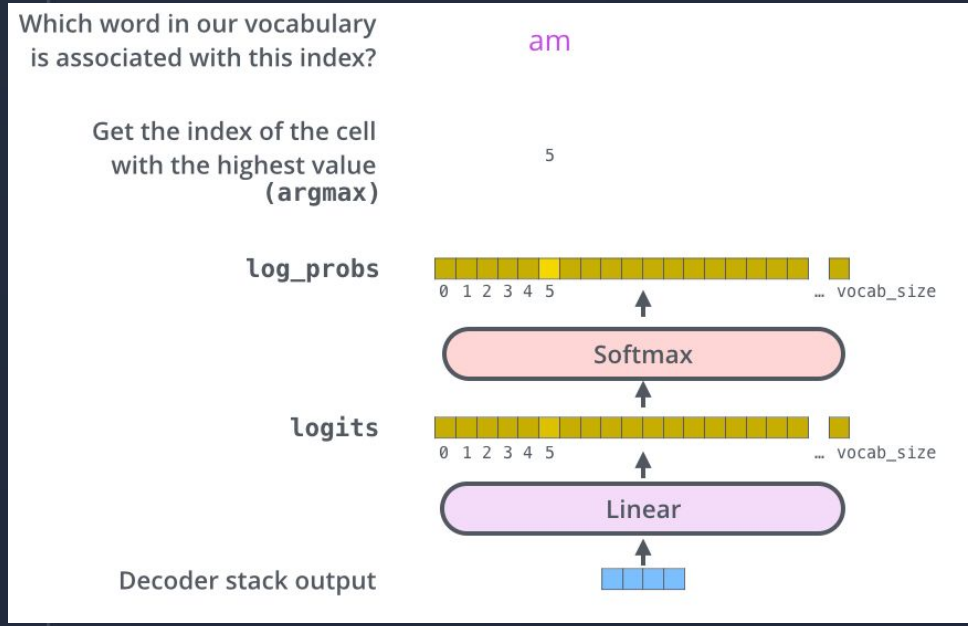
✅ Linear Layer: Projects the decoder's output to a high-dimensional vector—the size of the output vocabulary (e.g., 10,000 words).

➤ This produces a logits vector, where each cell represents the score for a potential word.

✅ Softmax Layer: Converts these logits into probabilities (summing to 1).

➤ The word with the highest probability is selected as the output at this step.

📌 This process repeats for each output token until the model emits an `<end>` token.



Encoder, Decoder & Encoder-Decoder Models

Different transformer training strategies depend on architecture:

🧠 Decoder-Only Models

- Pre-trained using causal language modeling.
- Example: Input "`the cat sat on`" → predict "`the`".
- Trained by shifting target sequences — often used in autoregressive LLMs like GPT.

🧩 Encoder-Only Models

- Trained using corrupted inputs, like **masked language modeling** (MLM).
- Example: Input "`The [MASK] sat on the mat`" → predict "`cat`".
- Used in models like **BERT** for understanding and representation tasks.

🔁 Encoder-Decoder Models

- Trained on sequence-to-sequence tasks like translation, summarization, and QA.
- Example: Input "`Le chat est assis sur le tapis`" → Output "`The cat sat on the mat`".
- Can also be trained unsupervised by splitting articles into two parts (input + target).
- Used in models like T5, original Transformer, and many multilingual models.

Data Preparation

Cleaning

• Apply filtering, deduplication, and normalization to improve data quality.

Tokenization

• Convert text into tokens using methods like Byte-Pair Encoding (BPE) and Unigram tokenization.
• Generates a **vocabulary**—a unique set of tokens that the model will understand.

Splitting

• Divide data into **training** and **test** sets.
• Training data is used to teach the model, while the test data is used for evaluation.

Training and Loss Function

Training Loop

- Batches of input sequences are sampled from a training dataset.
- Each input has a corresponding target sequence, often derived from the input in unsupervised setups.

Forward Pass

- Inputs are passed through the transformer to generate predicted outputs.

Loss Calculation

- The difference between predicted and target sequences is calculated using a loss function, commonly cross-entropy loss.

Backpropagation

- Gradients of the loss are computed.
- An optimizer updates the transformer’s parameters using these gradients.

Convergence

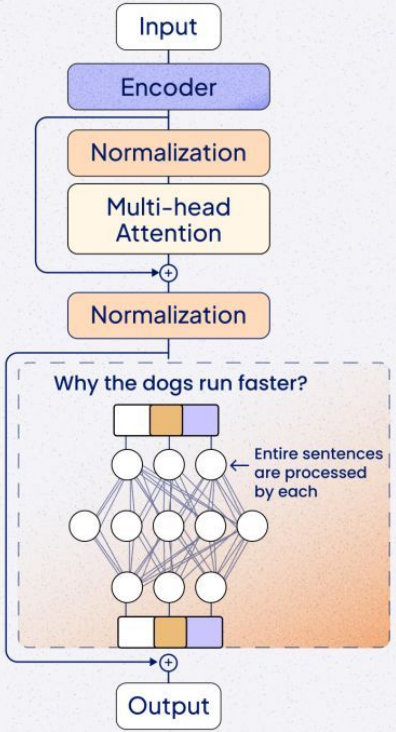
- The process is repeated until performance stabilizes or a token limit is reached.

MoE(Mixture of Experts) Architecture Explained

@rakeshgohel01

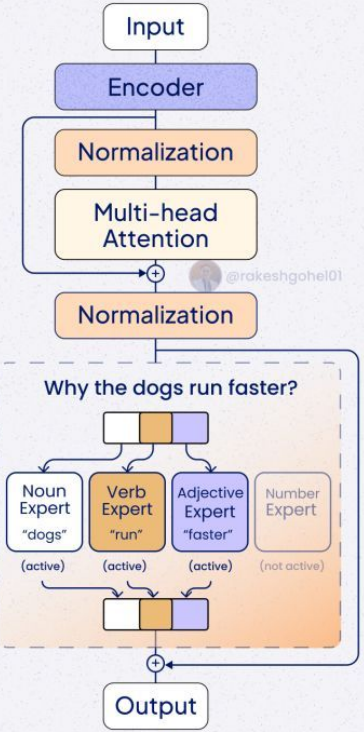
Dense Architecture

Used by: Gemin 2.5 Pro, Gemma 3

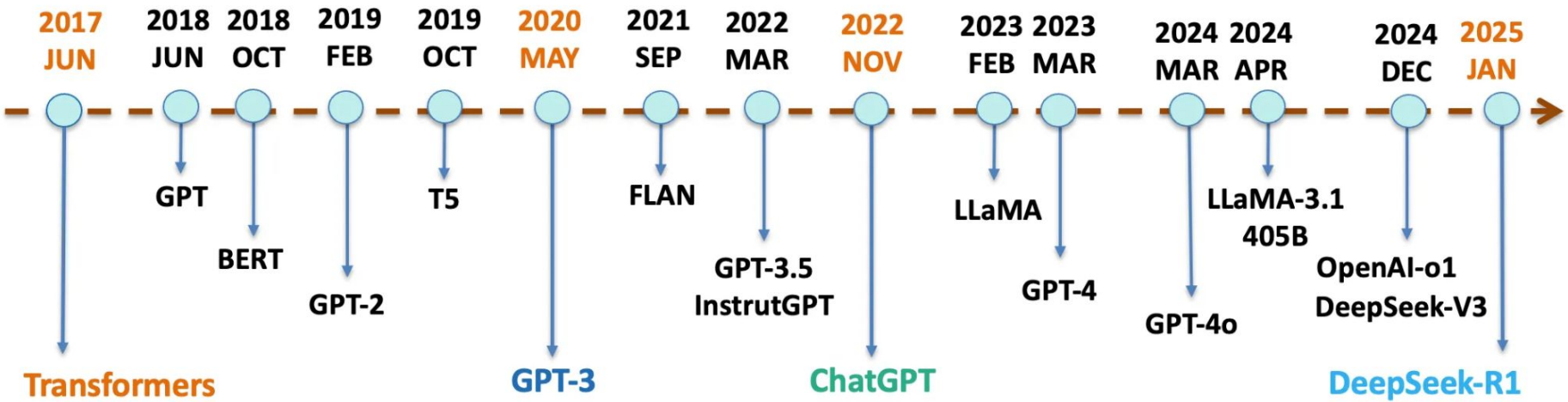


MoE Architecture

Used in: Llama 4, R1 and V3



A Brief History of LLMs



How to use LLMs

Exercise Coding
How LLM works?

https://github.com/Ntin0709/GEN-AI/blob/main/Excercise_Notebook.ipynb

https://tinyurl.com/ASgenai

Prompt Engineering

From Basics to Advanced Applications: A Practical Guide to Mastering AI Communication

Learn how to effectively craft prompts that unlock the full potential of AI models for real-world applications.

What is Prompt Engineering?

Prompt engineering is the process of designing, refining, and optimizing inputs to AI systems to elicit desired outputs, behaviors, or capabilities from the underlying language models.

The Key Interface

Prompts are the primary way we communicate with and control AI models

Natural Language

Leverages human language to guide AI reasoning and response patterns

Technical Skill

Combines understanding of AI capabilities with effective communication

Unlocks Potential

Well-crafted prompts dramatically enhance AI performance in real-world tasks

Core Components of Effective Prompts

Clear Instructions

Explicitly state what you want the AI to do

```
"Write a concise summary of the following text in 3 bullet points."
```

Context Setting

Provide necessary background information or constraints

```
"You are an expert in renewable energy advising a policy maker..."
```

Examples (Few-Shot Learning)

Show the AI what good outputs look like

```
"Input: cloudy → Output: The weather is cloudy today..."
```

Output Format Specification

Define how you want the response structured

```
"Format your response as a markdown table with columns for Name, Price, and Rating."
```

Basic Prompt Techniques

Role Prompting

Assign a specific role to the AI to frame its perspective and expertise.

```
"Act as an experienced cybersecurity expert and explain how to create a secure password policy for a mid-size company."
```

Step-by-Step Instructions

Break down complex requests into sequential steps for more controlled outputs.

```
"First, summarize the key points of this article. Then, identify potential biases. Finally, suggest three follow-up questions."
```

Zero-Shot Prompting

Request a task without providing examples, relying on the model's pre-trained knowledge.

```
"Classify this customer feedback as positive, negative, or neutral: 'The product arrived on time but was missing parts.'"
```

Few-Shot Learning

Provide examples of desired input-output pairs to guide the model's response pattern.

```
"Input: It's sunny outside.
Output: The weather is nice today.

Input: There's a thunderstorm.
Output: ..."
```

Intermediate Techniques

Chain-of-Thought Prompting

Instruct the AI to break down complex reasoning into logical steps, improving accuracy on difficult tasks.

```
"Think step-by-step about how to solve this math problem: If a store has 75 items and sells 30% on Monday, then 25% of the remainder on Tuesday, how many items are left?"
```

System & User Messages

Separate instructions into system context and user queries for more consistent interactions.

```
"System: You are a financial advisor who specializes in retirement planning.

User: I'm 35 and want to retire by 55. What should I do?"
```

Constrained Generation

Set specific limitations on the response format, length, or content to control outputs.

```
"Write a product description for a smart speaker in exactly 50 words. Include keywords 'voice assistant', 'smart home', and 'easy setup'."
```

Self-Consistency

Generate multiple solutions to a problem and then select the most consistent or common answer.

```
"Solve this problem 3 different ways, then determine which answer appears most frequently: What is the next number in the sequence 2, 5, 10, 17, 26, ...?"
```

Advanced Techniques

ReAct (Reasoning + Acting)

Combines reasoning and action steps, allowing the AI to think and act iteratively to solve complex problems.

```
"To answer this question, think about what information you need, then search for that information, and finally formulate your answer based on what you find."
```

Tree of Thoughts

Explore multiple reasoning paths in parallel, evaluating each branch before selecting the most promising solution.

```
"Consider three different approaches to this problem. For each approach, think through the implications, challenges, and likely outcomes before recommending the best solution."
```

Retrieval-Augmented Generation

Enhance responses by retrieving and incorporating external knowledge from specific sources.

```
"Here is information from our company handbook: [handbook text]. Based specifically on this information, answer the following question about our vacation policy."
```

Meta-Prompting

Create prompts that instruct the AI to generate better prompts for specific tasks, leveraging the model's own capabilities.

```
"Design the optimal prompt that would help a language model generate a comprehensive business plan for a subscription box service. Then execute that prompt."
```

Prompt Design Best Practices

Be Specific and Explicit

Avoid ambiguity by clearly articulating your requirements. The more specific your instructions, the better the results.

Start Simple, Then Iterate

Begin with a basic prompt and refine it based on results. Progressive improvements often lead to better outcomes than complex first attempts.

Set Constraints

Define boundaries for length, format, tone, and content. Constraints help channel the AI's capabilities toward your specific goals.

Provide Context

Include relevant background information. Context helps the AI understand the domain and respond appropriately to your query.

Define the Audience

Specify who the response is for. Tailoring to an audience (e.g., "explain to a 10-year-old" or "for technical experts") improves relevance.

Use Prompt Templates

Create reusable structures for common tasks. Templates ensure consistency and save time when working on similar projects.

💡 Pro Tip: Document your successful prompts and organize them by use case. Building a prompt library accelerates future work and helps identify patterns that produce quality results.

Common Pitfalls & Solutions

Vague Instructions

Ambiguous requests like "Write content about AI" produce generic, unfocused responses that rarely meet expectations.

Solution

Be specific: "Write a 250-word explanation of how AI is transforming healthcare, focusing on diagnostic applications."

Information Overload

Extremely long, complex prompts with excessive details can confuse the model and dilute the main request.

Solution

Break down complex prompts into sequential steps or multiple focused interactions to maintain clarity.

Hallucinations & Fabrication

AI models can generate plausible-sounding but incorrect information when prompted beyond their knowledge limits.

Solution

Ask the model to cite sources or state when it's uncertain. Verify critical information with external reliable sources.

Prompt Leakage

Including sensitive data in prompts can risk exposing information, as prompts may be logged or analyzed.

Solution

Sanitize prompts to remove personally identifiable information (PII) and sensitive data. Use placeholder data where possible.

Real-World Applications

Healthcare

Medical professionals use prompt engineering to extract insights from patient data and research literature.

> "Analyze these patient symptoms and lab results, then suggest possible diagnoses with their likelihood, ordered by probability."

Finance

Financial analysts create prompts that help interpret market trends and generate investment insights.

> "Summarize the key points from this quarterly report and identify potential risks not explicitly mentioned that investors should consider."

Education

Educators design prompts to generate personalized learning materials and assess student understanding.

> "Create a 5-question quiz on photosynthesis for 8th-grade students, including questions that test both recall and application of concepts."

Software Development

Developers craft prompts to generate code, debug errors, and document software systems.

> "Explain what this function does, identify potential bugs, and suggest optimizations: [code snippet]"

E-commerce

Marketers engineer prompts to create product descriptions, analyze customer reviews, and generate personalized recommendations.

> "Write five different product descriptions for this coffee maker, each targeting a different customer persona: busy parent, coffee enthusiast, etc."

Customer Service

Support teams develop prompts for chatbots that accurately answer queries and escalate complex issues.

> "You are a customer service bot for a telecom company. Address customer concerns empathetically, and flag for human assistance if the query involves account changes."

Measuring & Optimizing Prompt Performance

Relevance

How accurately does the response address the query's intent? Look for direct answers to questions and appropriate handling of the request.

Accuracy

Factual correctness and reliability of the information. Cross-check against trusted sources, especially for specialized domains.

Usability

How easily can users apply the response? Includes clarity, organization, completeness, and appropriate level of detail.

Prompt Optimization Workflow

1. **Establish Baseline**
   Create an initial prompt and measure its performance against your key metrics.
2. **Identify Improvements**
   Analyze where the response falls short and what elements of the prompt might be causing issues.
3. **A/B Testing**
   Create variants that address different aspects (phrasing, structure, examples, etc.) and test them systematically.
4. **Iterate & Refine**
   Implement improvements from winning variants, then repeat the process to continually enhance results.
5. **Document & Standardize**
   Create templates from successful patterns and share knowledge across teams.

The Future of Prompt Engineering

Automated Prompt Optimization

AI-powered tools that automatically generate, test, and refine prompts based on user goals, reducing the manual effort of prompt engineering.

High Impact Potential

Multi-Modal Prompting

Integrated prompting across text, images, audio, and video, enabling more natural human-AI interactions and complex, cross-modal reasoning.

Rapidly Emerging

Domain-Specific Prompt Libraries

Specialized collections of optimized prompts for specific industries, standardizing best practices and reducing redundant engineering efforts.

Growing Adoption

Collaborative Prompt Development

Platforms for teams to collaboratively design, test, and iterate on prompts, with version control and performance tracking capabilities.

Enterprise Focus

> *"As models continue to evolve, prompt engineering will shift from writing specific instructions to designing strategic interactions that guide AI reasoning processes across complex tasks and domains."*

Conclusion & Key Takeaways

Key Takeaways

- Prompt engineering is a critical skill for effectively leveraging AI models in real-world applications.
- Structure matters: Clear instructions, context, examples, and formatting specifications dramatically improve results.
- Advanced techniques like chain-of-thought and ReAct can solve complex reasoning tasks previously impossible with basic prompting.
- Iteration and testing are essential—rarely will your first prompt be your best prompt.
- Domain knowledge enhances prompt quality—the best prompts combine AI expertise with subject matter expertise.

Resources for Learning

Prompt Engineering Guide
Comprehensive open-source guide maintained by DAIR.AI

OpenAI Cookbook
Official examples and techniques for working with OpenAI models

Learn Prompting
Free, community-driven course on prompt engineering techniques

Prompt Engineering Community
Forums and Discord servers dedicated to sharing prompt techniques

Getting Started Today

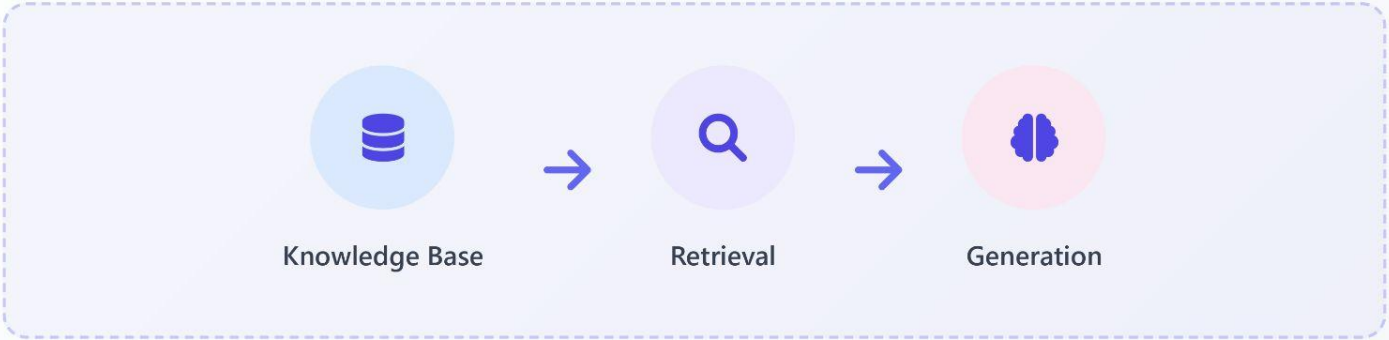
Begin with simple use cases in your domain, document your results, iterate on your prompts, and gradually integrate more advanced techniques as you gain experience.

CODING EXERCISE

https://github.com/Ntin0709/GEN-AI/blob/main/1_Prompting.ipynb

Retrieval-Augmented Generation (RAG)

From Basics to Advanced Applications



A foundation for building knowledge-grounded AI applications

What is Retrieval-Augmented Generation?

RAG is a technique that combines the power of large language models with the ability to retrieve information from external knowledge sources before generating a response.

The Problem RAG Solves

- LLMs have limited, static knowledge from training
- No awareness of recent information
- Can't access domain-specific knowledge
- Prone to hallucinations when uncertain

Key Benefits

Up-to-date
Access to latest information

Grounded
Responses based on facts

Reduced Hallucinations
More accurate responses

Adaptable
Works with any knowledge source

"RAG connects language models to external knowledge, enabling them to generate responses grounded in the most relevant, up-to-date information."

How RAG Works: Architecture Overview

Key Components

Knowledge Base
Document collection, vector database, external APIs

Embedding Model
Converts text to vector representations

Retriever
Semantic search and similarity matching

Language Model
Generates coherent, contextual responses

How It's Different

Traditional LLM

Relies solely on internal parametric knowledge learned during training

- ⚠ Knowledge cutoff
- ⚠ No citation sources
- ⚠ Hallucination prone

RAG Model

Dynamically retrieves external knowledge when needed

- ✓ Up-to-date information
- ✓ Citable sources
- ✓ Factually grounded

RAG = Parametric Knowledge (LLM) + Non-parametric Knowledge (Retrieval)

Component 1: Document Loading & Data Preparation

What is Document Loading?

The process of ingesting, parsing, and transforming various data sources into a format suitable for embedding and retrieval in a RAG system.

Why It Matters

- Quality of retrieval directly depends on document preparation
- Proper chunking improves semantic relevance
- Efficient storage impacts system performance

Common Data Sources

Text Files
TXT, CSV, JSON, PDF, Markdown, HTML

Databases
SQL, NoSQL, Vector DBs, Graph DBs

Web Data
Web pages, APIs, RSS feeds, Wikis

Conversations
Chat logs, emails, message histories

Code & Docs
Source code, documentation, technical specs

Enterprise
Internal wikis, knowledge bases, reports

Document Processing Pipeline

1. **Loading & Parsing** → Extract raw text from various file formats
2. **Text Cleaning** → Remove noise, normalize formatting, handle special characters
3. **Text Splitting** → Chunk documents with appropriate size and overlap
4. **Metadata Extraction** → Add source info, timestamps, categories, and other attributes

Key Considerations

Chunking Strategy
Find optimal chunk size: too large = irrelevant info, too small = lost context

Metadata Enrichment
Add context that improves filtering and retrieval precision

Data Freshness
Establish update mechanisms for maintaining current information

Text Chunking Strategies

What is Text Chunking?

The process of splitting documents into smaller, manageable pieces (chunks) for efficient embedding and retrieval in RAG systems.

Why Chunking Matters:

- Defines the context window for retrieval
- Impacts semantic coherence of retrieved content
- Affects storage requirements and retrieval speed
- Determines quality of context provided to LLM

Chunk Size Considerations

Too Small

- Loss of contextual information
- Increased storage overhead
- More embedding computations

Too Large

- Diluted semantic meaning
- Retrieves irrelevant information
- Context length limitations

Best Practice: "Goldilocks Zone"

Find the optimal chunk size that balances semantic integrity with retrieval precision. Typically between 256-1024 tokens depending on content type.

Common Chunking Strategies

Fixed-Size Chunking

Split text into equal-sized chunks (by characters, words, or tokens)

Simple to implement · May break semantic boundaries

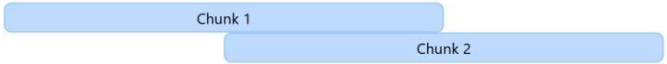
Semantic Chunking

Split text based on semantic boundaries (paragraphs, sections)

Preserves meaning units · Variable chunk sizes

Overlapping Chunks

Create chunks with overlapping content to preserve cross-boundary context



Preserves cross-boundary information · Storage redundancy

Hierarchical Chunking

Create multi-level chunks (document → section → paragraph)

Multi-level context retrieval · More complex implementation

Component 2: Text Embeddings

What are Embeddings?

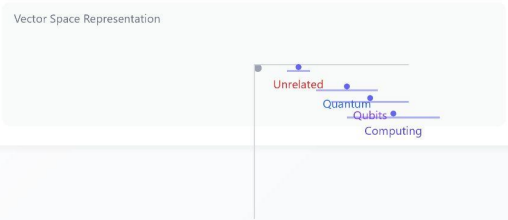
Embeddings are dense vector representations of text that capture semantic meaning in a high-dimensional space. They convert words, sentences, or documents into numerical vectors where similar meanings have similar vector representations.

Key Properties:

- Dimensionality: Typically 384-1536 dimensions
- Semantic Similarity: Measured by cosine similarity or dot product
- Context Awareness: Capture meaning based on surrounding words

From Text to Vectors: How It Works

"Quantum computing uses qubits" → [0.32, -0.71, 0.44, ...]



Popular Embedding Models

OpenAI

text-embedding-3-small/large

Dimensions: 1536

- High quality, industry standard

Sentence Transformers

all-MiniLM-L6-v2

Dimensions: 384

- Lightweight, open-source

Cohere

embed-english-v3.0

Dimensions: 1024

- Specialized for RAG systems

BGE

bge-large-en-v1.5

Dimensions: 1024

- Strong MTEB benchmark score

Embedding Best Practices

Model Selection
Choose models optimized for retrieval tasks, not just general embeddings

Chunk-Level Embeddings
Embed at appropriate semantic units to match query granularity

Cache Management
Cache embeddings to reduce API costs and latency

Dimension Tradeoffs
Higher dimensions = better accuracy but more storage/compute

Real-World Consideration

The choice of embedding model is one of the most influential factors for RAG performance. Benchmarks like MTEB (Massive Text Embedding Benchmark) can help evaluate and select the best model for your use case.

Pro Tip: Test multiple embedding models on your specific dataset. The best general-purpose model might not be the best for your domain-specific content.

Vector Databases: Storing and Retrieving Embeddings

What is a Vector Database?

A specialized database designed to efficiently store, index, and query high-dimensional vector embeddings based on similarity rather than exact matching.

Why Vector DBs Matter for RAG:

- Enable approximate nearest neighbor (ANN) search
- Scale to millions or billions of vectors
- Support metadata filtering with vector search
- Offer low-latency for real-time applications

Vector Search Algorithms

HNSW
Hierarchical Navigable Small World - Graph-based approach with multiple layers

IVF
Inverted File Index - Clusters vectors for faster search

PQ
Product Quantization - Compresses vectors to save memory

Tradeoff: Accuracy vs. Speed vs. Memory Usage

Popular Vector Database Solutions

Pinecone

Fully Managed | Serverless

Pure vector database optimized for production-grade vector search with minimal setup.

- Easy to use
- Scales automatically

Milvus

Open Source | Cloud/Self-host

Scalable vector database with multiple similarity metrics and index types.

- Highly configurable
- Active community

Weaviate

Open Source | GraphQL API

Vector search engine with semantic schema and real-time vectorization.

- Multi-modal support
- Built-in vectorization

Qdrant

Open Source | Rust-based

Vector similarity search engine with rich filtering and payload storage.

- Advanced filtering
- High performance

Feature Comparison

Database	Setup Ease	Performance	Scalability	Filtering	Metadata
Pinecone	●●●●●	●●●●○	●●●●●	●●●○○	●●●●○
Milvus	●●○○○	●●●●●	●●●●●	●●●●○	●●●●○
Weaviate	●●●○○	●●●●○	●●●●○	●●●●●	●●●●●
Qdrant	●●●●○	●●●●●	●●●●○	●●●●●	●●●●●

Selection Criteria

Scale & Performance
Consider current and future data volume needs

Deployment Model
Cloud-managed vs. self-hosted options

Filtering Capabilities
Support for metadata/hybrid search

Component 3: Retrieval Strategies

What is Retrieval in RAG?

The process of finding the most relevant documents or context from a knowledge base in response to a query, to provide as context for the language model.

Goal
Find the most semantically relevant information to augment LLM's knowledge

Challenge
Balance between precision, recall, and computational efficiency

Similarity Search Methods

Cosine Similarity

Measures the cosine of the angle between vectors - focuses on direction, not magnitude.

```
similarity = cos(θ) = A·B / (||A||·||B||)
```

Dot Product

Multiplies corresponding elements and sums the results - considers both direction and magnitude.

```
similarity = A·B = ∑(A_i × B_i)
```

Euclidean Distance

Calculates the "straight-line" distance between vectors in the embedding space.

```
distance = √∑(A_i - B_i)²
```

Advanced Retrieval Techniques

Dense Retrieval

Basic RAG

Standard vector similarity search directly from embeddings

Use when: You need a simple, effective approach for semantic search

Hybrid Search

Advanced

Combines semantic (vector) and keyword (BM25) search results

Use when: Dealing with technical content or specific terminology

Metadata Filtering

Targeted

Pre-filters document set based on categorical or numerical attributes

Use when: Working with diverse document sources or time-sensitive data

Re-ranking

Advanced

Uses a separate model to re-score initially retrieved documents

Use when: Precision is critical and additional compute is acceptable

Query Processing

Query Expansion

Generating multiple variations of the original query to increase retrieval coverage

✓ Improves recall ✗ May reduce precision

Query Decomposition

Breaking complex queries into simpler sub-queries for more targeted retrieval

✓ Handles complex queries ✗ Increases latency

Component 4: Generation with Retrieved Context

How Retrieved Context Enhances LLMs

The generation phase combines retrieved documents with the original query to create a prompt for the LLM, enabling it to produce responses grounded in the retrieved knowledge.

Benefits of Context-Augmented Generation:

- Access to information outside model's training data
- Reduced hallucination with factual grounding
- More specific, detailed, and accurate responses
- Source attribution and verifiability

Prompt Engineering for RAG

Basic RAG Prompt Template

```
Answer the question based ONLY on the following context:
[Retrieved Documents]
Question: [User Query]
Answer:
```

Enhanced RAG Prompt Template

```
You are a helpful assistant. Answer the user's question
based on the provided context.
If the answer cannot be determined from the context, say "I
don't know" instead of making up information.
Context: [Retrieved Documents]
Question: [User Query]
Answer:
```

Citation-Based RAG Prompt

```
Answer the question using only the provided sources. For
each part of your answer, indicate which source(s) support
that information using [1], [2], etc.
Sources:
[1] [Document 1 with metadata]
[2] [Document 2 with metadata]
Question: [User Query]
Answer with citations:
```

Context Injection Techniques

Direct Context Insertion

Simply insert relevant documents into the prompt template.

✓ Simple to implement
✗ Limited by context window

Relevance Filtering

Use similarity scores to include only the most relevant content.

✓ Focuses on key information
✗ May miss relevant context

Document Compression

Summarize or extract key information from retrieved documents.

✓ Fits more content in context
✗ Potential information loss

Multi-Step Reasoning

Use an intermediate step to analyze context before final generation.

✓ Better reasoning with context
✗ Increased latency and cost

Token Management Strategies

Context Truncation
Prioritize most relevant chunks when context exceeds limits

Chunk Re-ranking
Use an additional model to score and select best chunks

Hybrid Approaches
Use model with larger context for complex queries

LangChain RAG Implementation Example

```python
from langchain.chat_models import ChatOpenAI
from langchain.schema import HumanMessage, SystemMessage
from langchain.prompts import ChatPromptTemplate

# 1. Retrieve relevant documents
docs = retriever.get_relevant_documents(query)

# 2. Format the documents into a string
context = "\n\n".join([doc.page_content for doc in docs])

# 3. Create a prompt template
prompt_template = ChatPromptTemplate.from_messages([
    SystemMessage(content="You are a helpful assistant that answers based or
    ("human", "Context: {context}\n\nQuestion: {question}\n\nAnswer:")
])

# 4. Format the prompt with our query and context
prompt = prompt_template.format(context=context, question=query)

# 5. Generate the answer using the LLM
llm = ChatOpenAI(model_name="gpt-4")
response = llm.invoke(prompt)
```

Advanced RAG Architectures

Beyond Basic RAG

Advanced RAG architectures extend the basic retrieval-augmented generation approach to solve complex challenges, improve accuracy, and reduce hallucinations.

Multi-Query RAG

Popular

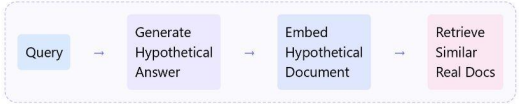
Generates multiple variations of the original query to perform separate retrieval operations, retrieving a broader set of relevant documents.

Use when: Handling complex questions that may require multiple perspectives or aspects

HyDE (Hypothetical Document Embeddings)

Efficient

Uses LLM to generate a hypothetical document that answers the query, then embeds and uses it to retrieve similar real documents.

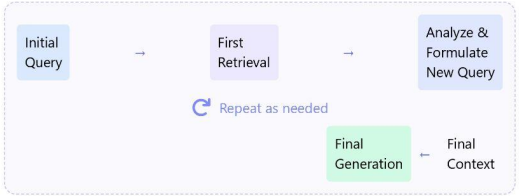


Use when: Dealing with low recall in semantic search or natural language queries

Recursive Retrieval RAG

Advanced

Uses multiple rounds of retrieval with LLM analysis in between to progressively refine the information gathered before final generation.

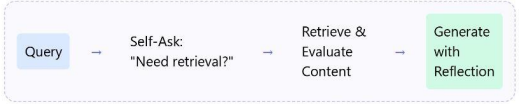


Use when: Solving complex tasks that require multi-step reasoning or exploration

Self-RAG: Self-Reflective Retrieval

Powerful

The model determines when to retrieve, critiques retrieved content, and decides whether to use the information in the response.



Use when: Highest quality responses are needed and model calibration is important

Architecture Comparison

Approach	Complexity	Latency	Accuracy	LLM Calls
Basic RAG	Low	Low	Moderate	1
Multi-Query	Moderate	Moderate	High	2+
HyDE	Moderate	Moderate	High	2
Recursive	High	High	Very High	3+
Self-RAG	Very High	High	Highest	Multiple

Evaluating RAG Systems

Why Evaluation Matters

Proper evaluation of RAG systems ensures they meet user needs, provide accurate information, and justify implementation costs.

Evaluation Challenges:

- Multiple system components to assess
- Balancing quantitative and qualitative metrics
- Domain-specific evaluation requirements
- End-to-end performance vs. component-level

Key RAG Benchmarks

RAGAS

Open-source framework for evaluating RAG pipelines along multiple dimensions.

Faithfulness | Context Relevance | Answer Relevance

KILT

Knowledge-Intensive Language Tasks benchmark for evaluating retrieval and generation.

Fact Checking | Entity Linking | Question Answering

LangChain Evaluation

Tools for custom evaluation of LLM chains and RAG pipelines.

Correctness | Reasoning | Helpfulness

RAG Evaluation Dimensions

Retrieval Quality

How well the system finds relevant documents from the knowledge base.

Precision & Recall
Measures relevance and comprehensiveness of retrieved documents

Mean Reciprocal Rank (MRR)
Evaluates rank position of first relevant document

Context Relevance
How well retrieved content matches information needs

Generation Quality

How well the LLM generates responses using retrieved context.

Faithfulness
Factual alignment with retrieved information

Answer Relevance
Direct addressing of the user's question

Hallucination Rate
Frequency of factually incorrect information

Evaluation Methodologies

Methodology	Description	Advantages	Challenges
Human Evaluation	Manual assessment by subject matter experts	High accuracy, nuanced feedback	Time-consuming, expensive, subjective
LLM-based Evaluation	Using LLMs to assess system outputs	Scalable, consistent, cost-effective	May inherit LLM biases, limited understanding
Benchmark Testing	Testing against standard datasets with known answers	Standardized, reproducible, comparable	May not reflect real-world usage
A/B Testing	Comparing different RAG configurations in production	Real-world performance metrics, user feedback	Complex setup, potential user experience impact

Example Evaluation Pipeline

1. **Define Test Set** — Create diverse query-answer pairs
2. **Run RAG System** — Log retrieval & generation outputs
3. **Compute Metrics** — Apply evaluation frameworks
4. **Analyze & Iterate** — Improve based on findings

RAG Optimization Techniques

Why Optimize Your RAG System?

Optimization improves accuracy, reduces latency, lowers costs, and enhances user experience with RAG applications.

Retrieval Optimization

Query Rewriting

Transform user queries to be more effective for semantic search by expanding, focusing, or reformulating them.

Original: "AI risks"

Rewritten: "What are the primary safety concerns, ethical challenges, and existential risks associated with advanced artificial intelligence systems?"

Improvement Impact: High

Hybrid Search Tuning

Optimize the weighting between semantic (vector) and keyword (BM25) search methods based on query types.

Factual Queries	Conceptual	Technical
70/30 BM25/Vector	30/70 BM25/Vector	50/50 Balance

Improvement Impact: Medium-High

Context Processing Optimization

Document Compression

Summarize or extract key information from retrieved documents to maximize context window utilization.

- Use LLMs to extract only relevant portions of retrieved documents
- Apply hierarchical compression: title → summary → details
- Implement adaptive compression based on document relevance score

Improvement Impact: High

Context Re-ranking

Apply a secondary model to reorder retrieved documents based on more nuanced relevance criteria.

Cross-encoders like `ms-marco-MiniLM-L-12-v2` can analyze query-document pairs to produce more accurate relevance scores than initial retrieval.

Improvement Impact: Medium-High

Generation Optimization

Prompt Engineering

Refine prompt templates to better guide the LLM in using retrieved context correctly.

Chain of Thought

Guide LLM to reason step-by-step through retrieved information

Context Weighting

Indicate confidence levels for different retrieved pieces

Improvement Impact: Very High

Scaling RAG Systems for Production

Production Scaling Challenges

Moving RAG from prototype to production introduces engineering challenges that must be addressed for reliable, efficient, and cost-effective systems.

- High query volume
- Growing data size
- Cost optimization
- Latency requirements

Infrastructure Scaling Strategies

Horizontal Scaling

Deploy multiple RAG service instances behind a load balancer to distribute query processing.

Implementation Options:

- Kubernetes-orchestrated auto-scaling
- Replicated serverless functions
- Multi-region deployment for global reach

Vector Database Scaling

Scale vector storage and retrieval to handle billions of embeddings with sub-second query times.

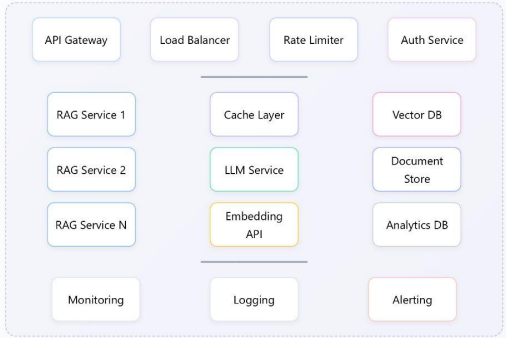
Scaling Options:

- Sharding by namespace/collection
- Distributed index structures
- Read replicas for high availability

Scale Metrics:

- QPS (queries per second)
- Vector count capacity
- Latency under load
- Cost per million vectors

Production Architecture Example



Performance Optimization Techniques

Caching Strategies

Implement multi-level caching to reduce computation and API calls.

Query Results Cache
Store complete responses for identical or similar user queries

Embedding Cache
Cache vector embeddings to avoid redundant encoding

Document Cache
Cache retrieved documents to reduce database load

Batch Processing

Process data in batches for better throughput and resource utilization.

Batch Embeddings
Process multiple texts in single embedding API call

Async Processing
Use queue systems for document ingestion and indexing

Scheduled Updates
Batch-update knowledge base during off-peak hours

Production Monitoring Essentials

Performance Metrics

- Latency (p50, p90, p99)
- Throughput (QPS)
- Error rates
- Resource utilization

Cost Metrics

- API call costs
- Storage costs
- Compute utilization
- Cost per query

Quality Metrics

- Retrieval precision
- Answer correctness
- Hallucination rate
- User feedback

Health Monitoring

- Service availability
- API dependencies
- Database connectivity
- Rate limit alerts

Multi-Modal RAG: Beyond Text

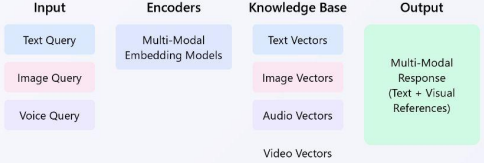
What is Multi-Modal RAG?

Multi-modal RAG extends traditional RAG beyond text to incorporate and retrieve from diverse data types such as images, audio, video, and structured data.

Key Capabilities:

- Process and understand multiple data formats
- Cross-modal retrieval (e.g., find images from text queries)
- Generate responses that reference or include different modalities

Multi-Modal RAG Architecture



Cross-modal retrieval allows finding relevant content across different modalities

Popular Multi-Modal Models

CLIP/BLIP
Text-image understanding models from OpenAI/Salesforce

LLaVA/GPT-4V
Vision-language models that process images and text

ImageBind
Meta's model for binding 6 modalities in one embedding space

Whisper/AudioLDM
Models for audio understanding and generation

Multi-Modal Capabilities

Visual Document Understanding

Extract and index information from visual documents like PDFs, diagrams, charts, and tables.

Applications:
Technical documentation | Financial reports | Scientific papers

Image-Based Knowledge Retrieval

Search and retrieve relevant images or information based on visual queries.

Applications:
Medical imaging analysis | Product visual search | Art and design research

Audio and Video Retrieval

Index and search audio and video content by transcriptions, visual elements, or sound patterns.

Applications:
Meeting intelligence | Educational video search | Multimedia archives

Implementation Challenges

Embedding Alignment
Different modalities may use different embedding spaces, requiring alignment techniques

Storage Requirements
Multi-modal data requires substantially more storage and specialized indexes

Computational Intensity
Processing images and video requires significantly more computing power

Relevance Evaluation
Determining relevance across modalities is more complex than text-only systems

Solution Approach: Use specialized multi-modal foundation models combined with modality-specific preprocessing and joint embedding spaces.

Common RAG Applications & Use Cases

Enterprise Knowledge

Access and utilize company knowledge across internal documents, wikis, and databases.

- Internal docs
- Company policies
- Knowledge graphs

Example: Acme Inc.
Created an internal assistant that answers employee questions using 15 years of company documentation and policies.

Healthcare

Support medical professionals with up-to-date research and patient-specific information.

- Medical research
- Clinical guidelines
- Patient records

Example: MediAssist
Helps doctors quickly retrieve relevant case studies and treatment guidelines for rare conditions based on specific patient symptoms.

Education

Create personalized learning experiences with access to accurate educational content.

- Textbooks
- Academic papers
- Course materials

Example: StudyBuddy
Tutoring platform that answers student questions by retrieving information from specific assigned textbooks and lecture notes.

Customer Support

Provide accurate, context-aware responses to customer inquiries using product knowledge.

- Product manuals
- Support tickets
- FAQs

Example: SupportAI
Reduced support ticket volume by 40% by answering customer questions using product documentation and previous support interactions.

Finance

Access financial data and reports to support investment and analysis tasks.

- Earnings reports
- Market data
- SEC filings

Example: FinAnalyst
Helps investment analysts quickly find and compare relevant financial metrics across thousands of quarterly reports and presentations.

Software Development

Assist developers with codebase-specific information and guidance.

- Source code
- Documentation
- Issue trackers

Example: CodeCopilot
Helps developers understand large codebases by retrieving context-specific documentation and code examples from project repositories.

Ethical Considerations in RAG Systems

Why Ethics Matter in RAG

RAG systems inherit ethical challenges from both the AI models they use and the knowledge sources they retrieve from. Responsible deployment requires addressing these concerns.

> *"The quality, accuracy, and fairness of RAG systems are directly dependent on the knowledge sources they access, making the curation of these sources an ethical responsibility."*

Hallucinations

Even with retrieved context, LLMs can still generate inaccurate or fabricated information.

Mitigation Strategies:
- Citation requirements in prompts
- Fact-checking final outputs
- Confidence scores for answers

Bias

RAG systems can amplify biases present in both the LLM and the knowledge sources.

Mitigation Strategies:
- Diverse knowledge sources
- Regular bias audits
- Balanced representation

Privacy & Security

RAG systems must handle sensitive information responsibly and maintain data security.

Privacy Concerns:
- PII in training/retrieval data
- Unauthorized data access
- Data leakage via prompts

Security Measures:
- Access controls
- Data encryption
- PII detection/redaction
- Audit trails

Key Ethical Challenges

Copyright & Attribution

RAG systems use and reproduce content that may be protected by copyright laws.

Challenges:
- Proper attribution of retrieved information
- Fair use vs. copyright infringement
- Managing licensed content

Best Practices:
- Cite sources in responses
- Track content provenance
- Obtain proper licenses for commercial use

Transparency & Explainability

Users should understand where information comes from and how the system works.

Implementation Methods:
- Clear source attribution in responses
- Confidence scores for retrieved information
- Explainable retrieval processes
- Documentation on system limitations

Responsible RAG Development Practices

Knowledge Source Curation

Carefully select, validate, and update knowledge sources to ensure accuracy and balance

Retrieval Oversight

Monitor retrieval patterns to detect and correct bias in information selection

Response Guardrails

Implement safety measures to prevent harmful outputs even with potentially harmful retrieved content

Continuous Evaluation

Regularly test and assess system behavior with diverse queries and scenarios

Human-in-the-loop verification | Regular bias audits | Transparency documentation | User feedback channels | Ethical review boards

RAG Development Tools & Frameworks

Building RAG Systems

A variety of specialized tools and frameworks have emerged to simplify the development, deployment, and management of RAG applications.

Key Framework Features:

- Data connectors & document loaders
- Text processing & chunking utilities
- Embedding model integrations
- Vector storage abstractions
- LLM integrations & chaining

Framework Selection Guide

How to Choose the Right Framework

1. **Consider your use case complexity**
   Simple RAG vs multi-stage pipelines vs agent-based systems
2. **Evaluate integration needs**
   LLM providers, vector databases, and data source connectors
3. **Assess deployment requirements**
   Cloud, on-premises, edge, or hybrid environments
4. **Consider community & support**
   Documentation, examples, active development, and community size

Popular RAG Frameworks

LangChain

Most widely adopted RAG framework

A comprehensive framework for building LLM applications with chain-based architecture.

- Chain of Thought
- Agents
- Memory

Language Support: Python, JavaScript/TypeScript

LlamaIndex

Data framework for LLM applications

Specializes in data ingestion, indexing, and structured data handling for LLMs.

- Query Engines
- Data Connectors
- Structured Data

Language Support: Python, TypeScript (Beta)

Haystack

Modular NLP framework

End-to-end framework with strong focus on production-ready search systems.

- Pipeline Design
- Evaluation
- Hybrid Search

Language Support: Python

LangFlow

Visual pipeline builder

UI-based tool for building and visualizing LangChain flows without coding.

- Visual Editor
- Low Code
- Code Export

Based on: LangChain (Python)

Additional Tools & Utilities

Evaluation Tools

Tools for measuring RAG performance

- RAGAS
- TruLens
- DeepEval

Deployment Platforms

Specialized hosting for RAG applications

- Vercel AI SDK
- Modal
- LangServe

Experimental Tools

Emerging tools for advanced RAG systems

- DSPy
- Instructor
- CrewAI

Conclusion & Further Resources

Key Takeaways

1. RAG combines the strengths of LLMs with external knowledge retrieval to create more accurate, up-to-date, and factual AI systems
2. The quality of your retrieval pipeline (document preparation, embeddings, vector store) directly impacts the quality of generated responses
3. Advanced RAG techniques (query rewriting, multi-step retrieval, etc.) can significantly improve system performance
4. RAG is a rapidly evolving field with applications across industries — stay current with new research and techniques

What's Next?

Build Your Own RAG System

Apply what you've learned to create a custom RAG solution for your specific use case

Experiment with Techniques

Test different embedding models, chunking strategies, and retrieval methods

Join the Community

Participate in open-source projects and forums focused on RAG development

Keep Learning

Stay up-to-date with the latest research papers and framework updates

Resources for Further Learning

Educational Resources

- → LangChain RAG Documentation
- → DeepLearning.AI RAG Course
- → Pinecone Learning Center: RAG Guide

Research Papers

- → "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" (Lewis et al.)
- → "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection" (Asai et al.)
- → "RARR: Researching and Revising What Language Models Say" (Gao et al.)

GitHub Repositories

- → LangChain RAG Examples
- → LlamaIndex (GPT Index) - Advanced RAG
- → RAGAS - Evaluation Framework for RAG Systems

CODING EXERCISE

https://github.com/Ntin0709/GEN-AI/blob/main/Simple_RAG.ipynb
https://github.com/Ntin0709/GEN-AI/blob/main/RAG_with_Gemma%2C_Langchain_and_HuggingFace.ipynb

LLM Guardrails

Ensuring Safe and Reliable AI Interactions

Guardrails are programmable safety controls that monitor and guide a user's interaction with Large Language Model applications, ensuring they operate within ethical, legal, and technical boundaries.

What are LLM Guardrails?

- Pre-defined rules and filters designed to protect LLM applications from vulnerabilities
- Small programs that validate and correct the model's outputs
- Control mechanisms that ensure outputs align with application requirements

Why Guardrails Matter

- Prevent harmful, biased, or inappropriate content generation
- Protect against jailbreaks, prompt injections, and data leakage
- Ensure compliance with regulations and ethical standards

Types of Guardrails & Implementation Techniques

Five Primary Types of Guardrails

Input Rails

Filter and validate user inputs before they reach the LLM; can reject harmful prompts or mask sensitive data

Dialog Rails

Guide conversation flow, enforce specific dialog paths, and determine when to engage specific responses

Retrieval Rails

Validate information retrieved from external sources in RAG applications to prevent hallucinations

Execution Rails

Monitor and control tool/action calls made by LLMs to external systems or APIs

Output Rails

Filter and validate LLM-generated content before it reaches the user, ensuring safe responses

Implementation Techniques

Rule-Based Computation

Using predefined rules, regex patterns, and statistical methods to filter content based on explicit criteria

LLM-Based Metrics

Leveraging metrics derived from LLM evaluations to score and validate content against safety thresholds

LLM Judges

Using separate LLMs to evaluate and judge the output of the main model for compliance with guidelines

Prompt Engineering

Crafting specialized prompts with instructions that encourage the LLM to stay within safety boundaries

Best Practice: Multi-Layer Approach

Combine multiple guardrail types and techniques to create comprehensive protection against various vulnerabilities and edge cases.

Popular Guardrail Libraries & Implementations

Guardrails AI

Python | Open Source

A Python framework that helps build reliable AI applications by performing validation and structured data generation.

- Input/Output risk detection and mitigation
- Structured data generation from LLMs
- Hub of pre-built validators

NVIDIA NeMo Guardrails

Python | NVIDIA | Colang

An open-source toolkit for adding programmable guardrails to LLM-based conversational systems.

- Five rail types: input, dialog, retrieval, execution, output
- Colang language for dialog flow control
- Built-in jailbreak detection, moderation, fact-checking

LangChain Integration

Python | JavaScript | Framework

Integration options to add guardrails to LangChain-based applications and workflows.

- GuardrailsOutputParser for output validation
- LCEL compatibility for complex chains
- LangGraph integration for stateful guardrails

Implementation Best Practices

- Layer multiple guardrails for comprehensive protection
- Test guardrails with diverse, challenging inputs
- Continuously monitor and refine guardrail performance

Agentic AI

Practical Applications and Real-World Impact

Autonomous Agents

Real-World Systems

Intelligent Decision-Making

Integrated Frameworks

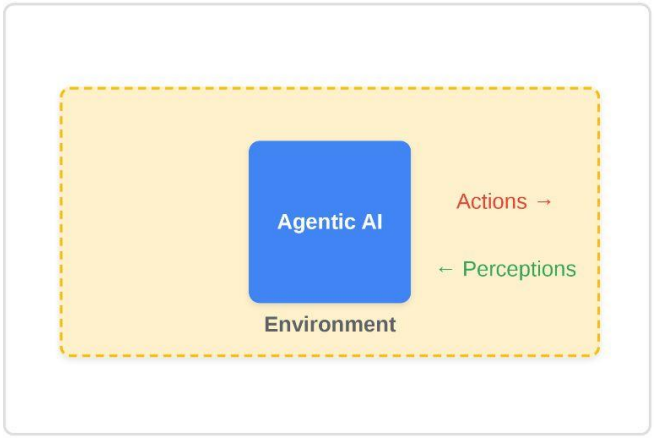
What is Agentic AI?

Definition:

Agentic AI refers to AI systems that can autonomously perceive their environment, make decisions, and take actions to achieve specified goals without constant human instruction or intervention.

Key Characteristics:

- Goal-oriented: Works toward specific objectives
- Autonomous: Can operate independently
- Adaptive: Learns from feedback and experience
- Persistent: Continues working until goals are met



Agentic AI	Rule-based Systems
Adapts to new situations and learns from experience	Follows fixed rules without learning or adaptation
Can handle uncertainty and incomplete information	Requires complete information and clear conditions
Makes decisions based on complex goals	Makes decisions based on predefined if-then logic

Evolution & Workflow of Agentic AI

Understand

Perceive environment, interpret data, and analyze context to establish situational awareness

Goal-Set

Establish objectives and success criteria based on understanding and user requirements

Execute

Take actions, make API calls, and interact with systems to accomplish established goals

Learn

Evaluate outcomes, update knowledge, and refine strategies for future interactions

Real-World Workflow Example: Customer Service Agent

Understand
Analyze customer inquiry about billing issue from chat history

Goal-Set
Identify billing error and determine steps to resolve issue

Execute
Query billing system, issue credit, update account information

Learn
Note resolution path for similar future issues, update knowledge base

Agentic vs. Generative AI

Agentic AI

- Takes autonomous actions to achieve goals
- Maintains state and memory between interactions
- Uses tools and accesses external systems
- Plans and executes multi-step processes
- Adapts strategy based on feedback

Examples:

AutoGPT

Autonomous agent that can perform tasks like research and content creation

Shopping Assistant

Finds products, compares prices, and places orders on behalf of users

Generative AI

- Creates content based on patterns in training data
- Responds to prompts without autonomous action
- Typically stateless between interactions
- Works within context window limitations
- Relies on human direction for task sequencing

Examples:

ChatGPT

Conversational model that generates text responses to prompts

DALL-E

Creates images from text descriptions

Benefits of Agentic AI

Autonomy

- Reduces human intervention in complex tasks
- Makes decisions based on changing conditions
- Handles unexpected scenarios adaptively
- Operates continuously without supervision

Case Study: Healthcare Monitoring

An agentic AI system monitors patient vital signs, adjusts treatment parameters, and alerts medical staff only when human intervention is required, reducing alert fatigue by 68%.

Scalability

- Handles increasing workloads without proportional resource increase
- Manages multiple tasks simultaneously
- Distributes work across multiple instances
- Maintains performance as complexity increases

Case Study: Customer Support

A major e-commerce platform implemented agentic AI for customer support, handling 250,000 inquiries daily with 92% resolution rate without increasing staff, reducing cost-per-resolution by 74%.

67%

Average reduction in human intervention needed for routine tasks

3.5x

Average productivity improvement in workflow automation

24/7

Operational capability without human supervision

85%

Companies reporting significant ROI from agentic AI deployment

Use Cases & Frameworks

LangChain

Framework for developing applications powered by language models with modular components and predefined chains for common tasks.

Key Features:

- Chains for sequencing LLM operations
- Memory for context retention
- Document loaders and retrievers
- Tool/API integration capabilities

CrewAI

Framework focused on creating teams of AI agents that collaborate to solve complex tasks with role specialization and process management.

Key Features:

- Role-based agent specialization
- Inter-agent communication
- Task delegation and coordination
- Process orchestration for complex workflows

AutoGPT

Experimental application that creates fully autonomous agents capable of setting their own tasks to achieve high-level goals.

Key Features:

- Goal-oriented autonomous planning
- Long-term memory management
- Internet browsing capabilities
- Self-evaluation and correction

Real-World Applications

Financial Research Assistant

Using LangChain to create an agent that analyzes financial reports, extracts key metrics, and generates investment insights by integrating with financial databases and APIs.

Travel Planning Crew

Implementing CrewAI to build a team of specialized agents that handle different aspects of travel planning: lodging, transportation, activities, and budget management.

Autonomous Code Generation

Using AutoGPT to develop entire software applications from high-level requirements, handling everything from architecture design to implementation and testing.

Risks and Best Practices

Key Risks

Goal Misalignment

Agents may interpret goals literally or find unexpected paths to achieve objectives, leading to unintended consequences.

Security Vulnerabilities

Autonomous agents with system access create new attack surfaces and potential for exploitation by malicious actors.

Privacy Concerns

Agents that access, process, and act upon sensitive data may create privacy risks and compliance issues.

Transparency Issues

Complex agent decision-making processes can be difficult to understand, explain, or audit.

Best Practices

Clear Goal Specification

Define agent objectives with precise constraints, boundaries, and success criteria to prevent goal misinterpretation.

Human Oversight

Implement human-in-the-loop processes for critical decisions and regular review of agent performance and actions.

Fail-Safe Mechanisms

Design emergency shutdowns, authorization limits, and rollback capabilities for when agents operate outside parameters.

Testing & Validation

Thoroughly test agents in sandboxed environments with adversarial scenarios before deployment in production.

Implementation Framework for Responsible Agentic AI

1. **Assess Need & Risk**
   Evaluate use case suitability for agentic approach and perform comprehensive risk assessment
2. **Start Constrained**
   Begin with limited permissions and gradually increase autonomy as reliability is proven
3. **Monitor & Log**
   Implement comprehensive monitoring systems and maintain detailed audit trails
4. **Review & Improve**
   Regular performance reviews and iterative improvement based on observed behavior

Conclusion & Future Directions

Key Takeaways

- Agentic AI creates autonomous systems that understand, set goals, execute, and learn
- Key benefits include reduced human intervention and scalable task management
- Frameworks like LangChain, CrewAI, and AutoGPT enable practical implementations
- Responsible implementation requires clear goals, oversight, and fail-safes

Future Directions

- Multi-agent collaboration systems for complex problem-solving
- Specialized domain experts with deep industry knowledge
- Enhanced tool creation and manipulation capabilities
- Standardized ethical frameworks and regulatory compliance

Resources for Further Learning

LangChain Documentation
python.langchain.com

CrewAI GitHub
github.com/joaomdmoura/crewAI

AutoGPT Repository
github.com/Significant-Gravitas/AutoGPT

Agentic AI Newsletter
aispotlight.beehiiv.com

LLM Agents Course
learn.deeplearning.ai

Research Papers
arxiv.org/search/cs?query=agentic+AI

Coding Exercise

https://github.com/Ntin0709/GEN-AI/blob/main/code_agents.ipynb
https://github.com/Ntin0709/GEN-AI/blob/main/Building_an_agent_with_LangGraph.ipynb

Fine-Tuning Large Language Models

Techniques and Best Practices

Enhancing model performance for specialized tasks

What is Fine-Tuning?

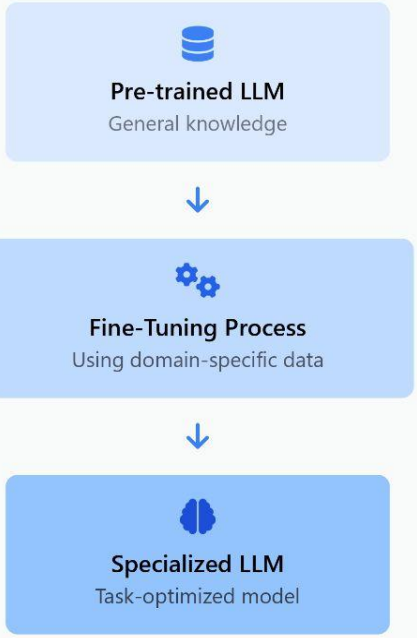
Fine-tuning is the process of taking a pre-trained language model and further training it on a domain-specific dataset to adapt it for specialized tasks.

Specialized Knowledge
Adapts general models for specific domains or tasks

Parameter Adjustment
Updates model weights while preserving core capabilities

Resource Efficiency
More efficient than training from scratch

Fine-Tuning Process



When and Why We Need to Fine-Tune

Key Reasons

Domain Specialization
Adapt models to industry-specific terminology and knowledge

Brand Voice Alignment
Ensure consistent tone and style matching your organization

Reduced Hallucinations
Improve factual accuracy for critical applications

Enhanced Performance
Achieve better results on specific tasks than generic models

When to Fine-Tune

Specialized Industries
Medical, legal, or technical domains with unique terminology

Multi-language Support
When needing stronger capabilities in specific languages

Proprietary Knowledge
Incorporating internal documents and company data

Specific Task Optimization
When a general model underperforms on targeted tasks

Pre-training vs Fine-tuning

Pre-training

Building foundational knowledge from massive datasets

Fine-tuning

Adapting the foundation for specific tasks

Aspect	Pre-training	Fine-tuning
Purpose	Learn general language understanding	Specialize for specific tasks/domains
Data	Vast, diverse, unlabeled text	Smaller, domain-specific, often labeled
Resources	High computational cost (weeks/months)	Lower computational cost (hours/days)
Learning	Self-supervised on general patterns	Supervised on specific tasks
Frequency	Once per model generation	Multiple times for different applications
Output	General-purpose foundation model	Task-optimized specialized model

Types of Fine-Tuning Techniques

Supervised Fine-Tuning (SFT)

Training a model on labeled examples to optimize for specific task performance. Uses high-quality prompt-response pairs to teach the model desired behaviors.

Reinforcement Learning from Human Feedback (RLHF)

Uses human preferences to further refine model outputs. Aligns model behavior with human values through reward modeling.

Parameter-Efficient Fine-Tuning (PEFT)

Updates only a small subset of model parameters, reducing computational costs while maintaining performance. Includes LoRA and QLoRA techniques.

Instruction Fine-Tuning

Training the model to follow specific instructions by using examples that demonstrate how it should respond to different prompts and requests.

Multi-task Fine-Tuning

Training on multiple tasks simultaneously to improve general capabilities and prevent catastrophic forgetting of previously learned skills.

Domain-Adaptive Fine-Tuning

Specializes the model for particular domains (medical, legal, etc.) by training on domain-specific corpora before task-specific training.

Supervised Fine-Tuning (SFT)

Supervised Fine-Tuning is a process that updates a pre-trained model using labeled data for specific tasks through direct training on examples.

The SFT Process

1. Prepare Dataset
Create high-quality prompt-response pairs that demonstrate desired model behavior

2. Train the Model
Update model weights using supervised learning on labeled examples

3. Evaluate Performance
Test the model's performance on validation data to ensure quality

4. Deploy Specialized Model
Implement the fine-tuned model for your specific application

Advantages

Targeted Optimization
Direct improvement on specific tasks using relevant examples

Resource Efficiency
Requires fewer computational resources than pre-training from scratch

Maintains General Knowledge
Preserves foundational capabilities while adding specialized skills

Limitations

Data Quality Dependency
Highly sensitive to the quality and diversity of training examples

Potential for Overfitting
Can become too specialized if training data is too narrow

Catastrophic Forgetting
Risk of losing previously learned capabilities during targeted training

Reinforcement Learning from Human Feedback (RLHF)

RLHF refines models through iterative training using human preference data, aligning AI behavior with human values and expectations.

RLHF Process

1. Initial SFT Model

Start with a model already fine-tuned using supervised learning

2. Create Comparison Data

Generate multiple responses to prompts and have humans rank them by quality

3. Train Reward Model

Build a model that predicts human preferences based on the comparison data

4. Optimize Policy with RL

Use reinforcement learning to optimize the model based on the reward function

SFT vs. RLHF

SFT (Supervised Fine-Tuning)
Direct training on labeled examples of desired outputs

VS

RLHF
Training based on human preferences between different outputs

Benefits of RLHF

- **Better Alignment**
  More closely matches human values and preferences
- **Reduced Harmful Outputs**
  Minimizes unsafe or inappropriate responses
- **Nuanced Learning**
  Captures subtle human preferences that are hard to specify
- **Continuous Improvement**
  Can be iteratively refined as more feedback is collected

Low-Rank Adaptation (LoRA)

LoRA is a parameter-efficient technique that fine-tunes LLMs by adding small trainable rank decomposition matrices to pre-trained weights, drastically reducing memory requirements.

How LoRA Works

Instead of updating all parameters:

LoRA freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer, greatly reducing the number of trainable parameters.

Mathematical Representation

$W + \Delta W = W + BA$, where W is frozen, and B and A are low-rank trainable matrices

Benefits of LoRA

Massive Memory Reduction

Can reduce trainable parameters by 10,000× compared to full fine-tuning while maintaining performance

Faster Training

Less memory usage means faster backpropagation and reduced training time

Model Adaptability

LoRA adapters can be swapped without changing the base model

Comparison with Full Fine-tuning

Memory Usage	~10% of full fine-tuning	
Training Speed	2-4× faster	
Performance	Comparable to full fine-tuning	

QLoRA: Quantized Low-Rank Adaptation

QLoRA extends LoRA by applying quantization to further reduce memory requirements, enabling fine-tuning of larger models on limited hardware.

How QLoRA Works

Quantization Process

Original Parameters (32-bit)			→	Quantized Parameters (4-bit)		
3.14159	2.71828	1.61803		3.14	2.72	1.62
0.57721	1.41421	0.69314		0.58	1.41	0.69

QLoRA quantizes the base model to 4-bit precision (NF4 format), significantly reducing memory usage while maintaining stability during fine-tuning.

Key Innovations

- 4-bit NormalFloat (NF4) quantization
- Double quantization to further reduce memory
- Paged optimizers to manage GPU memory

Hardware Impact

Enable fine-tuning of 65B+ parameter models on a single consumer GPU (e.g., RTX 4090 with 24GB VRAM)

QLoRA vs LoRA

Memory Efficiency
QLoRA uses up to 4x less memory than standard LoRA for the same model size

Speed Trade-off
LoRA is approximately 66% faster than QLoRA in training speed

Cost Efficiency
LoRA is up to 40% more cost-efficient for smaller models

Model Size Capability
QLoRA enables fine-tuning much larger models on limited hardware

When to Choose QLoRA

Ideal For

- Very large models (30B+ params)
- Limited GPU memory
- Single GPU environments
- Research experimentation

Consider LoRA When

- Training speed is critical
- Smaller models (< 13B params)
- Sufficient GPU memory available
- Production deployments

Real-life Applications of Fine-Tuning

Healthcare

Fine-tuned LLMs assist in medical report generation, symptom analysis, and specialized knowledge retrieval for rare conditions.

87% diagnostic accuracy

Legal

Models trained on legal documents assist with contract analysis, case research, and legal brief preparation with precise terminology.

70% time reduction in research

Customer Service

Domain-specific chatbots fine-tuned on company data provide consistent support aligned with brand voice and policies.

35% higher satisfaction rates

Financial Analysis

Models fine-tuned on financial reports and market data generate insights, summarize earnings calls, and predict market trends.

3x faster analysis workflows

Education

Personalized tutoring systems fine-tuned on curriculum content and teaching methodologies provide adaptive learning assistance.

24% improvement in test scores

Content Creation

Publishers fine-tune models to match their style guides, helping writers create consistent, on-brand content more efficiently.

50% faster content production

Scientific Research

Domain-specific models assist researchers by summarizing papers, suggesting experimental designs, and generating hypotheses.

42% more novel research pathways

Multilingual Support

Fine-tuned for specific language pairs and domains, providing more accurate and culturally appropriate translations.

65% reduced cultural errors

Code Assistants

Fine-tuned on company codebases to suggest code that follows internal standards and uses preferred libraries and patterns.

32% fewer bugs in production

Best Practices for Fine-Tuning LLMs

Data Preparation

Clean & Diverse Data

Remove errors, duplicates, and biased examples. Ensure broad coverage across your target domain to prevent overfitting.

Model Selection & Setup

Right-size Your Model

Choose a base model that balances performance with computational requirements. Smaller isn't always worse for specialized tasks.

Training Process

Monitor Metrics

Track loss, validation performance, and compute utilization. Use early stopping to prevent overfitting on your training data.

Evaluation

Multi-faceted Testing

Evaluate on diverse test cases, including edge cases and examples not seen during training. Test for both accuracy and safety.

Resource Optimization

Mixed Precision Training

Use 16-bit or mixed precision when possible to reduce memory usage and speed up training without significant quality loss.

Deployment

Quantization

Apply post-training quantization (8-bit, 4-bit) to reduce model size and inference costs while maintaining acceptable quality.

Key Takeaways

Fine-tuning Transforms LLMs

Adapts general-purpose models into specialized tools with domain expertise and aligned behaviors

Multiple Techniques Available

SFT, RLHF, LoRA, and QLoRA each offer unique advantages for different requirements and resources

Resource Efficiency Matters

PEFT methods like LoRA and QLoRA democratize fine-tuning by reducing computational requirements

Quality Data is Critical

The quality and relevance of training data directly impact fine-tuning success

Future of Fine-Tuning

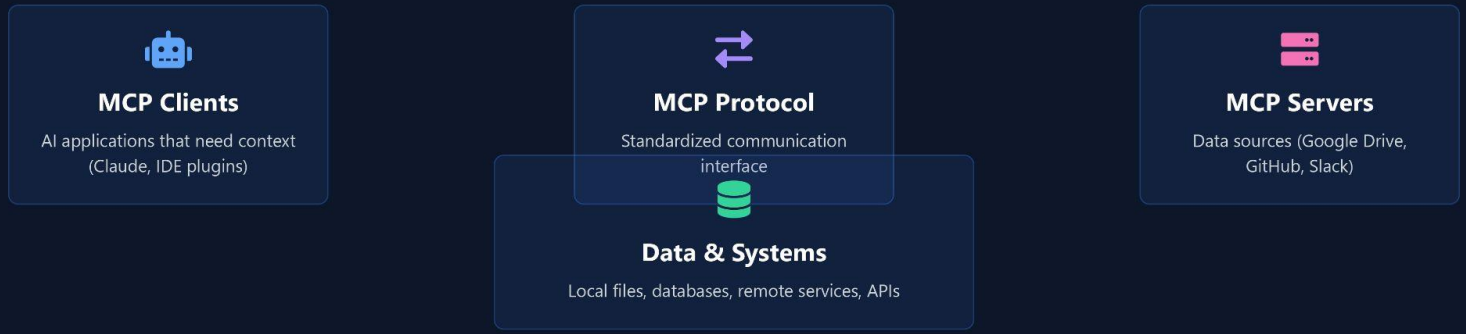
As models grow larger, parameter-efficient methods will become even more important. We'll likely see more specialized techniques balancing performance, efficiency, and alignment with human values. Automated fine-tuning processes may help democratize LLM customization further.

Model Context Protocol (MCP)

A Universal Standard for AI Connectivity

MCP is an open protocol that standardizes how applications provide context to Large Language Models.

Think of MCP like a USB-C port for AI applications — providing a standardized way to connect AI models to different data sources and tools.



Why MCP Matters

- Replaces fragmented integrations with a single protocol
- Connects AI systems with existing data sources
- Improves AI responses with relevant context

How MCP Works & Benefits

Key Components

Client API
AI tools (Claude, IDEs) that connect to MCP servers

Protocol
Standardized communication format with queries, resources, and capabilities

Server API
Data providers that expose resources through MCP endpoints

```
// Example MCP Connection
client.connect("github-server")
  .queryRepos({ owner: "anthropic" })
  .getFiles({ path: "/examples" })
```

Pre-built MCP Servers

GitHub
Access repos, code, issues and PRs

Slack
Connect to channels, messages

Google Drive
Access documents and files

Postgres
Query database content

Workflow Example

1. User asks Claude "Summarize our recent GitHub PR discussions"
2. Claude connects to GitHub MCP server via client
3. MCP server fetches PR data and returns to Claude
4. Claude provides a summarized response with context

Security
Data remains within your infrastructure. MCP servers control what can be accessed and by whom.

Interoperability
Connect to multiple data sources with a single protocol. Switch between AI providers easily.

Open Source
Community-driven ecosystem with SDKs for Python, JavaScript, C#, and other languages.

New Developments in Generative AI

Agentic AI

AI systems that autonomously take actions on behalf of users. They can proactively anticipate needs and execute complex workflows across applications.

Self-directed | Task orchestration

Multimodal AI

Systems trained on diverse data types (text, images, audio, video) creating more holistic and human-like cognitive experiences.

Cross-modal reasoning | Rich inputs/outputs

Retrieval-Augmented Generation

Enhances AI outputs by retrieving knowledge from external data sources before generating responses, improving factual accuracy.

Fact-based | Knowledge integration

Sentimental AI

Systems that analyze and interpret human emotions from text, speech, and visual inputs, enabling more empathetic interactions.

Emotion analysis | Personalization

Quantum AI

Combines quantum computing with AI to solve complex problems more efficiently than classical computers, opening new frontiers.

Exponential speedup | Advanced optimization

Explainable AI

Models designed to provide transparency into their decision-making processes, building trust and enabling better oversight.

Transparent decisions | Ethical compliance

The Future of AI in 2025

"In 2025, AI will evolve from a tool for work and home to an integral part of both. AI-powered agents will do more with greater autonomy and help simplify your life at home and on the job."

6 AI Trends for 2025
Microsoft Research

- More capable AI models
- AI companions for everyday life
- Resource-efficient infrastructure

Reference Links

Transformer Visualization - https://bbycroft.net/llm

Transformer Blog by Jay Alamar - https://jalammar.github.io/illustrated-transformer/

Google Gemini API Key - https://aistudio.google.com/app/apikey

Synth ID - https://deepmind.google/technologies/synthid/

Notebooklm by google - https://notebooklm.google/

Chroma db - https://www.trychroma.com/

Llama index - https://docs.llamaindex.ai/en/stable/

Understand Transformer from Andrej Karpathy - https://www.youtube.com/@AndrejKarpathy/videos

Run models locally by ollama - https://ollama.com/

Hugging Face model - https://huggingface.co/google/gemma-3-1b-it

Gen AI courses - https://www.deeplearning.ai/courses/

Andrew Ng courses - https://www.andrewng.org/courses/

FOR ANY DOUBTS YOU CAN WRITE TO ntech0709@gmail.com