Building Generative AI from Scratch: A Technical Deep Dive

Training of LLM's :

Generative AI (GenAI) has revolutionized various industries, from creative content generation to advanced data augmentation. But how exactly was Generative AI built from scratch? In this blog, we will explore the fundamental building blocks of GenAI, tracing its evolution from basic neural networks to state-of-the-art transformer architectures.

Pretraining on Large Text Corpora:

The model was trained on a diverse dataset containing books, articles, websites, and other publicly available text sources. Using a process called causal language modeling, the model learned to predict the next word in a sentence, enabling it to generate coherent text.

Now what is Causal Language Modelling:

▶ Causal Language Modeling (CLM) trains a model to predict the next token in a sequence, using only past tokens. This respects the temporal order, ensuring the model doesn't "peek" into future tokens.

Example:
Prompt: "The patient was diagnosed with"
Prediction: "cancer"

⚙!! How It Works in LLMs:
Autoregressive Modeling: Models like GPT predict tokens one at a time, feeding outputs back into the model for the next prediction.
Masked Self-Attention: Uses masked attention so each token can only attend to previous tokens.
Training Objective: Optimizes for next-token prediction loss, typically using cross-entropy.
?? Causal vs. Masked Language Modeling:

Characteristic	Causal LM(GPT)	Masked LM(BERT)
Directionality	Left-to-right	Bidirectional
Training Objective	Predict next token	Predict masked tokens
Strengths	Text generation, dialogue systems	Understanding tasks (NER, classification)
Weakness	Limited to past context during generation	Not ideal for generation

Supervised Fine-Tuning:

Once pretraining was completed, the model was fine-tuned using human-annotated datasets. This step helped refine its ability to generate accurate and contextually relevant responses.

What is Supervised Fine-Tuning?:

▶ Supervised Fine-Tuning (SFT) is a key step in training Large Language Models (LLMs) after the initial pretraining phase.

⚙!! How It Works in LLMs:
?? Pretraining: The model learns general language patterns from vast datasets (e.g., books, web pages) using self-supervised tasks like causal language modeling.
?? Supervised Fine-Tuning: The pretrained model is fine-tuned on curated, task-specific data where both inputs and expected outputs are provided.
▶ Example tasks:
Question-Answering: Given a question, generate a correct answer.
Summarization: Turn a long document into a concise summary.
NER/Classification: Identify named entities or classify documents.
?? Loss Function: The model uses a supervised loss function, typically cross-entropy, to minimize the difference between its outputs and the ground-truth annotations.

Benefits of Supervised Fine-Tuning:
--> Contextual Relevance: Improves the model's ability to generate accurate, on-topic responses.
--> Domain Adaptation: Helps the model specialize in certain areas (e.g., healthcare, legal) by fine-tuning on domain-specific datasets.
--> Performance Boost: Enhances the model's results on specific downstream tasks compared to its general pretrained version.

Reinforcement Learning from Human Feedback (RLHF):

To further improve response quality, OpenAI employed Reinforcement Learning from Human Feedback (RLHF). This process involved:

Human Labeling: Human annotators ranked multiple responses from the model.
Reward Model: A secondary model learned from human preferences to rate responses.
Optimization: Using Proximal Policy Optimization (PPO), the model was fine-tuned to prioritize higher rated responses.

How it works?:

▶ Human Labeling

⚙Humans are shown multiple model responses to a prompt.
They rank these from best to worst based on helpfulness, relevance, and safety.
?? This creates high-quality training data for human preferences.

▶ Reward Model Training

⚙A secondary model is trained on the human rankings.
It learns to assign a score to a new model response — higher scores for responses similar to highly ranked ones.
?? This reward model acts like a critic that judges response quality

▶ Policy Optimization (PPO)

⚙The original language model is fine-tuned using the reward model's scores.
An algorithm called Proximal Policy Optimization (PPO) updates the model to generate responses that receive higher rewards.
?? This helps the model gradually produce more helpful, honest, and harmless replies.

Sample Text Training Through All Stages:

▶ Pretraining Stage – Learning Language Basics

⚙!!Raw text example: "Transformers have changed AI by introducing self-attention mechanisms."

The model is trained using next-token prediction on large, diverse text corpora. It sees incomplete sentences and learns to guess the most probable next word based on context.

--> Training Instance: Input: "Transformers have" Expected Output: "changed"

This process is repeated billions of times across various sequences.
?? The model gradually learns grammar, facts, associations, and how language flows — but not necessarily how to follow instructions or give useful answers yet.

▶ Supervised Fine-Tuning Stage – Teaching Task-Specific Behavior

After pretraining, the model is fine-tuned on curated, human-annotated input-output pairs. These examples are designed to teach it how to respond helpfully to specific prompts.

⚙!!Example Prompt: "Explain transformers in AI."
Target Output: "Transformers are deep learning models that use self-attention to process entire sequences efficiently."

Here, annotators manually craft high-quality answers.
?? This phase helps the model align with user expectations for clarity, informativeness, and format. It learns how to answer questions, summarize, translate, and more — moving beyond simple next-word guessing.

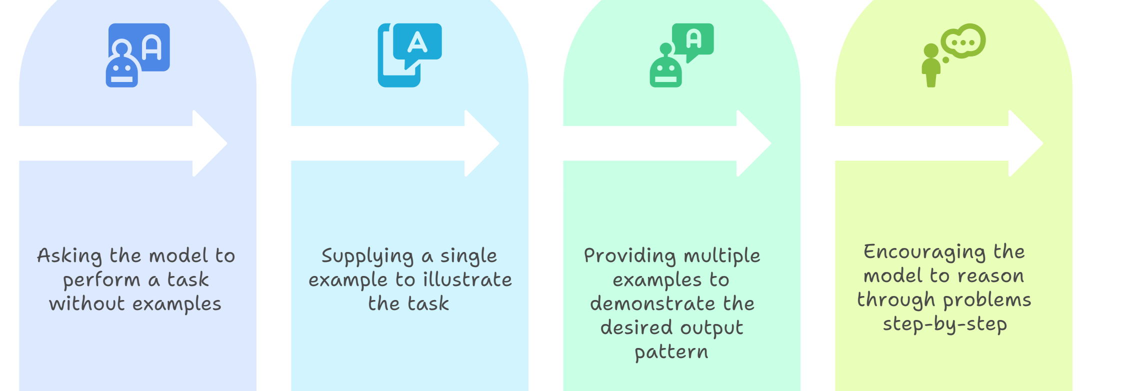
▶ RLHF Stage – Aligning with Human Judgment

The model now generates multiple responses for a single input. These outputs are then ranked by human reviewers based on helpfulness, accuracy, and safety.
⚙!!Example Prompt: "Explain transformers in AI."
Sample Model Responses:
"Transformers are great for AI."
"Transformers revolutionized AI by enabling parallel processing of sequences."
"Transformers use self-attention to improve deep learning architectures."

Human Ranking:
Best: Response 2 (clear, accurate, insightful)
Worst: Response 1 (too vague)

A reward model is trained to predict which answers humans would rate highly. Then, the main model is fine-tuned with Proximal Policy Optimization (PPO) to maximize these reward scores.
?? This stage encourages the model to prioritize helpful, honest, and harmless responses, even in ambiguous or nuanced cases.

Prompt Engineering Techniques



Performance Efficient Fine Tuning:

Training a full model is resource-intensive. PEFT methods enable the efficient use of memory and computation by modifying only a subset of the model's parameters, significantly reducing the required memory for training.

What is LoRA (Low-Rank Adaptation)??:

LoRA is a parameter-efficient fine-tuning (PEFT) technique that enables adaptation of large pre-trained language models (LLMs) without updating all model parameters.

Key Idea
LoRA introduces trainable low-rank matrices into specific parts of the model (typically attention layers), while keeping the original weights frozen. This significantly reduces memory usage and computational requirements

▶ How LoRA Works

⚙!!Decomposition of Weight Update:
Instead of directly updating a full weight matrix W, LoRA approximates the update as:

$W = W + \Delta W = W + A.B$
$A \in R^{d \times r}$
$B \in R^{r \times k}$

▶ Frozen Base Model:
W (original weights) remain unchanged.
Only A and B are trained.

☑ Benefits of LoRA

Fewer Trainable Parameters → Reduces training time & overfitting
Lower Memory Use → Can fine-tune on consumer-grade GPUs
Scalable Fine-Tuning → Suitable for very large models (e.g., LLaMA, GPT)

What is QLoRA (Quantized Low-Rank Adaptation)??:

QLoRA takes LoRA a step further by introducing quantization of the base model weights—enabling even more memory-efficient fine-tuning.

☑ What Makes QLoRA Unique?
Combines 4-bit quantization with low-rank adapters
Enables fine-tuning very large models (e.g., 65B) on a single GPU

Mathematical Intuition:

LoRA

▶ Problem Setup

⚙!!Goal: Fine-tune using low-rank update: $\Delta W = BA$

W = [[10, 20], [30, 40]]

We freeze W and learn low-rank matrices A and B:
Rank r = 1
A = [[1, 2]] , B = [[3],
[4]]

Compute the low-rank update:
ΔW = B × A = [3, 4] × [1,
2]
= [3, 6],
= [4, 8]
W' = W + ΔW = [
[10+3, 20+6] = [13, 26],
[30+4, 40+8] = [34, 48]

Only 4 parameters (in A and B) are trained instead of 4 in W — but the same shape update is achieved!

QLoRA

▶ Problem Setup

⚙!!Goal: Fine-tune using low-rank update: $\Delta W = BA$

W = [[10, 20], [30, 40]]

Quantize W to lower precision (e.g., int4):
W_q = quantize(W)
→ compressed version stored in 4-bit form

During inference/training:

W' = dequant(W_q) + B × A
W_q is only dequantized temporarily.

W_q is only dequantized temporarily.
You still add the same ΔW = B × A as before.
Saves memory by storing W_q in int4, but still gets performance boost from LoRA.

Evaluation Metrics:

Evaluating text generation models involves metrics that assess both fluency and relevance of the generated content.

Comparison of Text Evaluation Metrics

Metric Name	Definition	Use Cases	Strengths	Limitations
Perplexity	Measures prediction of word sequence.	Language modeling, next-token prediction	Lower value indicates better model.	Only for known probability distributions.
BLEU	Overlap between generated and reference texts.	Machine Translation, Text Summarization	Fast, easy to compute, widely adopted.	Ignores synonyms, penalizes diverse outputs.
ROUGE	Recall of reference text in generated text.	Summarization, dialogue systems	Emphasizes coverage of reference information.	Sensitive to rewording or paraphrasing.
METEOR	Aligns sentences using exact match, stems, synonyms.	Machine Translation, Chatbots	Better correlation with human judgment.	More computationally expensive to calculate.
BERTScore	Compares embeddings using BERT models.	Summarization, QA, Dialogue.	Captures semantic similarity, robust to paraphrasing.	Slower, depends on embedding model choice.

Image Generation Metrics Comparison

Metric	Inception Score	Fréchet Inception Distance
Definition	Quality and diversity evaluation	Distribution comparison
Formula	expπ(Ex\	FID= | | μr−μg | | 2+Tr(Σr+Σg−2(ΣrΣg)1/2)
Interpretation	High score means confident/diverse	Lower score means closer to real
Use Cases	GAN image evaluation	GAN comparison
Strengths	None	Measures quality and diversity
Limitations	Doesn't compare to real; gameable	Sensitive to noise; computationally expensive

Sample Calculation of Evaluation Metrics using toy data:

☑ Reference and Candidate

Let's assume: Reference sentence: R = `"the cat is on the mat"` , Candidate sentence: C = `"the the the cat sat on the mat"`

▶ 1. Perplexity

Perplexity = e^(−(1/N) ∑log P(wi)) for i=1 to n

Suppose model predicts probabilities:
P("the") = 0.4 , P("cat") = 0.2
P("sat") = 0.1 , P("on") = 0.1
P("mat") = 0.2

Then, for candidate:
C = "the the the cat sat on the mat" → 8 words

log P:
log(0.4) ≈ −0.916 (×3)
log(0.2) ≈ −1.609
log(0.1) ≈ −2.302 (×2)
log(0.2) ≈ −1.609

Sum = −0.916×3 − 1.609 − 2.302×2 − 1.609
= −2.748 − 1.609 − 4.604 − 1.609
= −10.570

Perplexity = e^(−(1/8) · (−10.570)) = e^(1.321) ≈ **3.75**

▶ 2. BLEU Score (up to bigrams)

BLEU = BP · exp(∑ wn · log pn)

* Modified unigram precision p1:
Unigrams in reference: {"the", "cat", "is", "on", "the", "mat"}
Unigrams in candidate: {"the"×3, "cat", "sat", "on", "the", "mat"}
Count clipped: "the" = 2 (max in reference), rest once each
Matched = {"the"×2, "cat", "on", "mat"} → 5 matches out of 8
→ p1 = 5/8

* Modified bigram precision p2:
Bigrams in reference: "the cat", "cat is", "is on", "on the", "the mat"
Bigrams in candidate:
"the the", "the the", "the cat", "cat sat", "sat on", "on the", "the mat"
Clipped match: "the cat", "on the", "the mat" → 3 matches out of 7
→ p2 = 3/7

* Weights: w1 = w2 = 0.5

* BP = 1 (candidate length ≥ reference length)

log p1 = ln(5/8) ≈ −0.470
log p2 = ln(3/7) ≈ −0.847
BLEU = 1 · exp(0.5·(−0.470) + 0.5·(−0.847))
= exp(−0.659) ≈ **0.517**

▶ 3. ROUGE-L (Longest Common Subsequence)

Reference: the cat is on the mat
Candidate: the the the cat sat on the mat
LCS = "the cat on the mat" → length = 5
Reference length = 6
Candidate length = 8

* Recall = LCS/Reference = 5/6 ≈ **0.833**
* Precision = LCS/Candidate = 5/8 = **0.625**

F1 = 2·(P·R)/(P + R) = 2·(0.625·0.833)/(0.625 + 0.833)
≈ 2·0.5208/1.458 ≈ **0.714**

→ **ROUGE-L F1 ≈ 0.714**

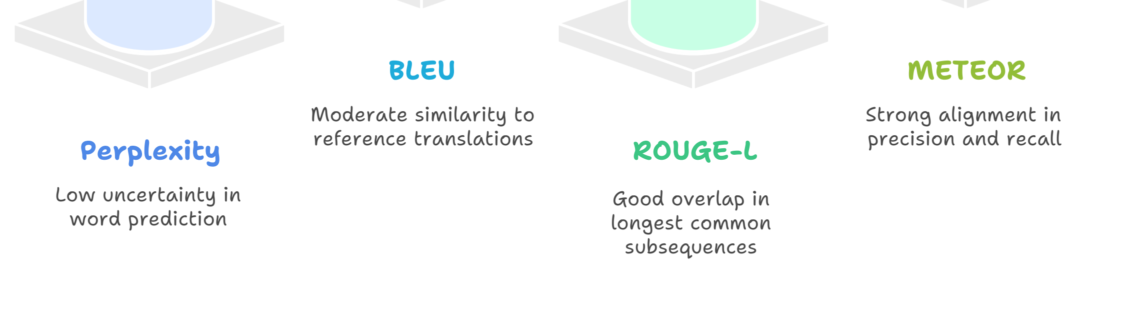
▶ 4. ?? **METEOR**

Steps:

* Matching unigrams = {"the"×2, "cat", "on", "mat"} → 5
* Precision P = 5/8 = 0.625
* Recall R = 5/6 ≈ 0.833
* F_mean = (10·P·R)/(R + 9·P)
= (10·0.625·0.833) / (0.833 + 5.625)
= 5.208 / 6.458 ≈ **0.806**

→ **METEOR ≈ 0.806**

Model Performance Metrics in NLP Tasks



Retrieval-Augmented Generation (RAG):

"Retrieval-Augmented Generation (RAG) is an AI technique that enhances large language models (LLMs) by retrieving external knowledge before generating responses. Instead of relying solely on a model's internal memory, RAG dynamically searches a knowledge base, extracts relevant information, and integrates it into the generation process. This improves accuracy, reduces hallucinations, and enables responses based on the latest available information."

Key Components Breakdown:

Data Preprocessing: Chunking & Vectorization:

☑ Before retrieval can happen, raw text data (e.g., documents, PDFs, or web pages) must be processed into smaller, searchable units called chunks.

Chunking: "Long documents are broken into smaller, meaningful sections (e.g., paragraphs or fixed-size text segments). This ensures that retrieval returns the most relevant context rather than entire documents."
Vectorization: "Each chunk is converted into an embedding (a numerical representation) using an embedding model (e.g., OpenAI's text-embedding models, BERT, or SBERT). These embeddings capture semantic meaning and are stored in a vector database for efficient similarity search."

☑ Why Chunking & Vectorization?

Prevents retrieving excessive or irrelevant text
Improves the efficiency of search queries
Helps the model focus on specific, relevant pieces of information

Retrieval Phase:

"When a user asks a question, their query is also converted into an embedding using the same embedding model. This embedding is then compared against stored document embeddings in a vector database (e.g., FAISS, Pinecone, ChromaDB) to find the most relevant chunks."

☑ Uses similarity search to find relevant text chunks
☑ Supports keyword-based (BM25) and dense vector search
☑ Retrieves only the most relevant knowledge for augmentation

Augmentation & Fusion Phase:

"The retrieved chunks are combined with the original query before being fed into a language model. This step ensures that the model generates responses grounded in factual, retrieved knowledge."

☑ Reduces hallucinations by forcing the model to use real data
☑ Fusion strategies include simple concatenation or attention-based integration
☑ Helps the model answer knowledge-intensive queries more accurately

Generation Phase:

"Finally, the augmented query is processed by a generative model (e.g., GPT-4, LLaMA, or T5), which uses both its pre-trained knowledge and the retrieved chunks to generate a final response."

☑ Uses retrieved information + model's internal knowledge
☑ Ensures responses are relevant, fact-based, and context-aware
☑ Improves interpretability and trustworthiness of AI-generated answers

Why Use RAG?

"Traditional LLMs are static—they cannot update their knowledge without expensive fine-tuning. RAG solves this by dynamically fetching external knowledge, allowing models to stay up to date without retraining."

▶ Key Benefits:

☑ More accurate responses with real-world knowledge
☑ Reduces hallucinations (misleading or false information)
☑ Lower computational cost compared to full fine-tuning
☑ Can be easily updated by refreshing the knowledge base

Possible Extensions & Optimizations

☑ Hybrid Search: Combining keyword search (BM25) with dense vector search for better retrieval
☑ Multi-Document Fusion: Retrieving multiple relevant chunks and merging insights
☑ Memory-Augmented RAG: Keeping track of previous conversations for contextual continuity
☑ RAG + Agents: Using agentic workflows where retrieved knowledge helps in multi-step reasoning

Handling Follow-up Questions

Q: How does RAG compare to fine-tuning?
"Fine-tuning updates a model's internal parameters, making it expensive and requiring periodic retraining. RAG, on the other hand, retrieves external knowledge dynamically, allowing real-time updates without modifying the model itself."

? Q: What challenges exist in RAG implementation?
"Challenges include retrieval latency, noisy data affecting response quality, and maintaining an efficient knowledge base. These can be mitigated with optimized chunking strategies, ranking algorithms, and indexing improvements."

? Q: How would you deploy RAG in production?
"I'd use a vector database like FAISS or Pinecone to store embeddings, integrate an API-based LLM for generation, and optimize retrieval latency using caching and ranking techniques."