Module 5: Large Language Models

The Engine Behind Generative AI

B.Tech Mechanical Engineering (7th Sem)

Guided by:

Prof. Dinesh Kumar
Assistant Professor

School of Engineering
D.Y. Patil International University (DYPIU)

October 29, 2025

Course: Product Design and Development

Module 5: Agenda

1. Core Concepts
2. Best Practices (Placeholder)
3. Applications (LLM)
4. Conclusion

Topic 1 Goes Here

Understanding the "Brain" of Generative AI

First, What is a Language Model (LM)?

- A Language Model is a system that understands and generates human language.
- At its core, it's a **statistical tool** that predicts the **probability** of a sequence of words.
- **Simple Example:** Your phone's keyboard auto-complete.

Example: Keyboard Prediction

If you type: "The quick brown fox jumps..."
The Language Model predicts the most probable next words:

- "over" (High probability)
- "into" (Medium probability)
- "apple" (Very low probability)

- A **Large** Language Model (LLM) is just a Language Model built at an *enormous* scale.

What is a Large Language Model (LLM)?

- An LLM is a massive (Deep Learning) Transformer model trained on a gigantic dataset of text and code (e.g., a large portion of the internet).
- **Core Function:** At its heart, an LLM is a *next-token predictor*.

How it "Thinks" (Simplified Analogy)

- It's like the **most powerful auto-complete** in the world.
- When you give it a prompt, it statistically predicts the most likely "next piece of text" (a token) based on all the text it has ever read.
- It generates a response token by token, feeding its own output back as the new input.

Example: Token-by-Token Generation

Prompt: "The primary material for a high-pressure turbine blade is..."
LLM Predicts: 1. "a" (most likely token) 2. "super" (most likely token after "a") 3. "alloy" (most likely token after "super") 4. "..." (e.g., "such as Inconel 718")

- "Intelligence" and "reasoning" are *emergent properties* of this core prediction task.

How are LLMs Trained? (The Basics)

Phase 1: Pre-training

- **What it is:** The model is "pre-trained" on a massive, general dataset (e.g., all of Wikipedia, all of GitHub, terabytes of books).
- **The Task:** "Masked Language Modeling" or "Next Token Prediction."
- **Example Task:** The model is given the sentence "The... brown fox jumps... the lazy dog" and must learn to fill in "quick" and "over."
- **Result:** A **Foundation Model**. It "understands" language, grammar, facts, and reasoning, but isn't built for any specific task.
- **Cost:** $10M - $1B+ (Billions)

Phase 2: Fine-Tuning

- **What it is:** The Foundation Model is then "fine-tuned" on a smaller, high-quality dataset of questions and answers.
- **Example Task:** Given thousands of examples of "Prompt $\rightarrow$ Good Answer" pairs, the model learns to be a helpful, instruction-following assistant.
- This is also where techniques like **RLHF** (Reinforcement Learning from Human Feedback) are used to make the model safer and less toxic.
- **Result:** A **Chatbot** (like ChatGPT, Claude).

Core Concept 1: Tokens

- LLMs don't "see" words. They see **Tokens**.
- A token is a common chunk of text. It can be a whole word, or just a part of a word.

Example: Tokenization

- The word `"engineer"` might be one token: `["engineer"]`
- The word `"engineering"` might be two tokens: `["engineer", "ing"]`
- The word `"hydrostatic"` might be three tokens: `["hydro", "stat", "ic"]`
- The phrase `"P-401A"` might be four tokens: `["P", "-", "401", "A"]`

Why This Matters to Engineers (Two Reasons)

1. **Cost:** Most LLM APIs (which we'll cover) charge you **per token**, not per word. Complex technical jargon and part numbers use more tokens.
2. **Context Window:** The AI's "memory" is measured in tokens, not words.

- **Rule of Thumb:** 100 tokens $\approx$ 75 words.

Core Concept 2: Embeddings (The "Math" of Words)

- The LLM cannot do math on a "token." It must first convert each token into a list of numbers called a **Vector Embedding**.
- **Analogy:** An embedding is like a "coordinate" for a word's *meaning* in a giant, multi-dimensional space.
- Words with similar meanings are "plotted" close together in this space.

The "King - Man + Woman = Queen" Analogy

This is the classic example:

- The vector for "King" is mathematically similar to the vector for "Man."
- The vector for "Queen" is mathematically similar to the vector for "Woman."
- The *relationship* (vector) between "King" and "Man" is almost identical to the *relationship* (vector) between "Queen" and "Woman."
- $\mathit{Vector}(\text{"}\mathit{King}\text{"}) - \mathit{Vector}(\text{"}\mathit{Man}\text{"}) + \mathit{Vector}(\text{"}\mathit{Woman}\text{"}) \approx \mathit{Vector}(\text{"}\mathit{Queen}\text{"})$

Engineering Example

In an LLM trained on engineering texts:
$\mathit{Vector}(\text{"}\mathit{Steel}\text{"}) - \mathit{Vector}(\text{"}\mathit{Stiffness}\text{"}) \approx \mathit{Vector}(\text{"}\mathit{Aluminum}\text{"}) - \mathit{Vector}(\text{"}\mathit{Stiffness}\text{"})$ (The relationship is similar)
$\mathit{Vector}(\text{"}\mathit{Bolt}\text{"}) \approx \mathit{Vector}(\text{"}\mathit{Screw}\text{"}) \approx \mathit{Vector}(\text{"}\mathit{Fastener}\text{"})$ (These would be "close" to each other)

Core Concept 3: The Transformer (Attention)

- The "T" in GPT (Generative Pre-trained **Transformer**).
- The Transformer architecture was a breakthrough (2017) that allowed models to get *really* big and "smart."
- **Key Innovation:** The **"Attention Mechanism."**
- "Attention" allows the model to weigh the importance of different words in a sentence, no matter how far apart they are.

Example: Why Attention is Critical

Consider this sentence:
*"The **bracket** that holds the main engine sensor must not fail, as **it** is a safety-critical component."*

- A human knows **"it"** refers to the **"bracket"**, not the "engine" or the "sensor."
- The **Attention Mechanism** allows the LLM to learn this connection. When the model is processing the word **"it"**, "Attention" tells it to pay high *attention* to the word **"bracket"**, even though it's far away.

- This is what allows an LLM to understand context, nuance, and long-range dependencies in your prompt.

Core Concept 4: Context Window (The AI's "Memory")

- A **Context Window** is the **maximum number of tokens** an LLM can "remember" or "see" at one time (both input + output).
- **Analogy:** It's the size of the AI's "whiteboard" or "short-term memory."
- Anything outside this window is forgotten.

Example: Context Window Sizes

- **GPT-3.5-Turbo:** 4,096 tokens (or 16k) $\approx$ 3,000 words
- **GPT-4-Turbo:** 128,000 tokens $\approx$ 100,000 words (a whole book)
- **Google Gemini 1.5 Pro:** 1,000,000 tokens $\approx$ 750,000 words (an entire library)

Why This is CRITICAL for Engineers

- You **cannot** paste your entire 100-page "Turbine Design Specification" PDF into a model with a 4k context window and ask a question about Chapter 9.
- The model will only "see" the first $\tilde{1}5$ pages.
- **You must choose a model whose context window fits your task.**

Core Concept 5: Parameters

- **What are they?** A "parameter" is a variable (a weight) inside the neural network that was "learned" during training.
- **Analogy:** Think of them as the **synapses in the brain**.
- The number of parameters is a rough measure of the **size, complexity, and "knowledge"** of a model.

Smaller Models

- 7 Billion Parameters (e.g., Llama 3 8B)
- **Pros:** Very fast, can run on a local machine (or even a phone).
- **Cons:** Less "smart," worse at complex reasoning.
- **Good for:** Simple tasks like classification, summarization.

Larger Models

- 175 Billion (GPT-3) to 1.7 Trillion+ (GPT-4)
- **Pros:** Extremely capable, can perform complex multi-step reasoning.
- **Cons:** Very slow, very expensive, requires massive data centers.
- **Good for:** Difficult problems (e.g., design critique, complex code).

- **Key Takeaway:** Bigger isn't always better. You must trade off "capability" vs. "speed and cost."

Core Concept 6: Giving LLMs Your Data

The Two Ways to Make an LLM "Smart" about Your Domain

Option 1: Fine-Tuning

- **What it is:** You take a pre-trained model (like Llama 3) and *continue the training process* using thousands of your own documents (e.g., all your FEA reports).
- **Analogy:** Sending a general doctor to a specialist medical school.
- **Result:** The model's *internal parameters (weights) are updated.* It "learns" your style and jargon.

Option 2: RAG (Retrieval-Augmented Generation)

- **What it is:** You "stuff" the prompt with relevant data *at the time of the query*.
- **Analogy:** Giving a doctor the patient's chart *before* they answer a question.
- **Result:** The model's *parameters are NOT updated.* It just uses the "context" you provide.

RAG is the most important pattern for engineering.

For 95% of business cases, RAG is the better, faster, and more practical solution.

RAG Deep Dive: How it Works (Conceptual)

This is the most important concept for practical applications.

The Problem

You Ask: "What is the recommended bolt torque for part P-401A?"
Public LLM (e.g., ChatGPT): "I'm sorry, I don't have access to your internal part documents. P-401A is not a public part."

The RAG Solution (A 3-Step Process)

- **Step 1: Retrieve (The "R")**
- Your RAG system takes your question ("...torque for P-401A?")
- It searches your *private company database* (e.g., Sharepoint, PLM) for relevant documents.
- It **finds** "P-401A_Spec_Sheet.pdf" and **extracts** the text: *"...the M12 bolts on P-401A must be torqued to 85 N-m."*
- **Step 2: Augment (The "A")**
- The system "stuffs" the prompt with this new context.
- It builds a *new, hidden prompt* to send to the LLM.
- **Step 3: Generate (The "G")**
- The LLM receives the augmented prompt (see next slide) and generates the answer.

RAG Deep Dive: The "Augmented Prompt"

The Hidden, Augmented Prompt Sent to the LLM

"You are a helpful engineering assistant. Use **only** the provided context below to answer the user's question. Do not use any other knowledge. If the answer is not in the context, say so.
CONTEXT: — *Document: "P-401A_Spec_Sheet.pdf"*
Title: P-401A Mounting Bracket Spec
...the M12 bolts on P-401A must be torqued to 85 N-m... —
USER'S QUESTION: "What is the recommended bolt torque for part P-401A?"

LLM's Final, Grounded Answer

"Based on the document *P-401A_Spec_Sheet.pdf*, the recommended torque for the M12 bolts on part P-401A is 85 N-m."

- **This solves the "Hallucination" problem!** The AI is forced to use *your* data.
- **This solves the "Data Privacy" problem!** Your documents are never sent to the LLM vendor for training.

RAG Deep Dive: How does "Retrieve" work?

- How does the system "find" the right document so fast?
- It uses a special database called a **Vector Database**.
- **Setup (Done once):**
  1. You take all 10,000 of your company documents.
  2. You split them into small chunks (e.g., by paragraph).
  3. You use an **Embedding Model** (see Core Concept 2!) to turn every chunk into a vector (a list of numbers).
  4. You store all these vectors in the Vector Database.

The "Retrieve" Step (Done at query time)

1. **User asks:** "What is the torque for P-401A?"
2. The system turns the *question itself* into an embedding vector.
3. It then searches the database: "Find me the 5 text chunks whose vectors are **mathematically closest** to my question's vector."
4. This is how it finds the *semantically relevant* chunk about "M12 bolts... torqued to 85 N-m," even if the user didn't use the exact words.

RAG vs. Fine-Tuning: When to Use Which?

Use RAG When...

- You need to "teach" the AI about **facts, figures, and knowledge**.
- The knowledge is **new or changes often** (e.g., new part specs, daily reports).
- You need to **cite your sources** (e.g., "According to doc P-401A...").
- You need to prevent factual hallucinations.
- **Example:** Building an "Internal Engineering Standards" chatbot.

Use Fine-Tuning When...

- You need to "teach" the AI a new **style, tone, or format**.
- You have thousands of examples of a specific task.
- The underlying "knowledge" doesn't change much.
- You are teaching a **skill**, not a **fact**.
- **Example:** Training an AI to *write in your company's formal report style*, or to convert your lab notes into a specific JSON format.

- **Advanced Use:** You can combine them! You can *fine-tune* a model to be good at writing reports, and then use *RAG* to feed it the facts for a specific report.

Topic 2 Goes Here

Moving from simple prompts to advanced instructions.

Recap: The 5 Elements of a Good Prompt

From Module 1 - This is your foundation.

1. Role (Persona)

"Act as a..." Sets the context and expertise.

2. Task (The Verb)

Be specific. "Compare," "Analyze," "Summarize," "Critique," "Generate."

3. Context (The "Why")

Provide the necessary background.

4. Format (The Output)

Tell the AI *how* you want the answer.

5. Constraints (The Rules)

Set the boundaries.

We will now break these down with new, detailed examples.

Best Practice 1: Persona Prompting (Role)

- Giving the AI a "Role" or "Persona" is the fastest way to improve the quality of its output. It primes the model to use the correct vocabulary and tone.

Vague Prompt

"Tell me about using aluminum for a bike."
Result: A generic, Wikipedia-style answer.

Good Persona Prompt

"Act as a **materials scientist** specializing in lightweight alloys. Explain the trade-offs between 6061-T6 and 7075-T6 aluminum for a competitive racing bicycle frame, focusing on fatigue life and weldability."
Result: A detailed, technical, and accurate comparison.

Best Practice 2: Specific Task (The Verb)

- Never use vague verbs like "Tell me about..." or "Do..."
- Use precise, actionable verbs.

Vague Task

"I need to know about bearings."

Specific Task

"**Compare and contrast** deep-groove ball bearings, angular contact bearings, and tapered roller bearings. **Analyze** them for a high-speed spindle application (15,000 RPM) with moderate axial and radial loads."

Best Practice 3: Provide Context (with Delimiters)

- Don’t make the AI guess. Give it all the background it needs.
- Use **delimiters** (like ```, """, `context`,) to clearly separate your instructions from the data/context you are providing.

Good Prompt with Delimiters

"You are a test engineer. Your task is to summarize a test report.
Use the following lab notebook entry, which is delimited by triple backticks, to write a formal summary.
```
Jan 10, 2025 - Test Rig 4
Part: P-401A Bracket, SN-003
Test: Cyclic load, 5kN to 50kN.
Result: Failed at 88,450 cycles.
Note: Fracture started at the main fillet, as predicted by FEA.
```
Write a 3-paragraph summary for the final project report. Start with the objective, describe the result, and state the conclusion."

Best Practice 4: Specify the Format

- Don't accept a wall of text. Tell the AI *exactly* how to structure its answer.
- This is crucial for using the output in other programs (e.g., Excel, Python).

Vague Format

"What are the pros and cons of CNC vs 3D printing for this part?"
Result: A long, hard-to-read paragraph.

Specific Format

"Generate a comparison of CNC milling and DMLS (3D Printing) for a titanium prototype part. **Format the output as a Markdown table** with the following columns: - Technology - Lead Time - Material Waste - Typical Cost per Part - Max Part Size"

Best Practice 5: Use Constraints (The Rules)

- It's often just as important to tell the AI what **not** to do.
- This helps constrain the output and avoid common, annoying AI behaviors (like "fluff").

Prompt with Negative Constraints

"You are an AI assistant helping a senior engineer.
Task: Compare the manufacturability of Ti-6Al-4V using 5-axis CNC vs. DMLS.
Format: A bulleted list.
Constraints: - Do not discuss cost. - Do not use overly conversational language (e.g., "That's a great question!"). - Do not define what Ti-6Al-4V is. Assume I am an expert. - Keep the comparison to 5 bullet points per method."

Result

A concise, professional, and dense summary, without the "fluff" that AIs usually add. This is perfect for pasting into a report.

Advanced Practice 1: Chain-of-Thought (CoT)

- For any problem involving math, logic, or multi-step reasoning, an LLM will often "rush" to the answer and get it wrong.
- **The Fix:** Add the simple magic phrase: **"Let's think step by step."**

Bad Prompt

"A 2m steel beam (E=200 GPa) has a rectangular cross-section (b=50mm, h=100mm). If a 10kN point load is applied at the center, what is the max deflection?
Answer: 13.3 mm"
(This answer is probably wrong. The AI guessed.)

Good Prompt (with CoT)

"A 2m steel beam (E=200 GPa) has a rectangular cross-section (b=50mm, h=100mm). If a 10kN point load is applied at the center, what is the max deflection?
Let's think step by step."
Answer: "Step 1: The formula for deflection of a simply supported beam with a center point load is $\delta = \frac{PL^3}{48EI}$.
Step 2: Calculate I for the rectangle: $I = \frac{bh^3}{12}$...
...
Step 6: The final deflection is 6.67 mm."
*(This is correct, and you can **check the work**.)*

Advanced Practice 2: Few-Shot Learning (Provide Examples)

- When you need a *very specific* output format (like for code or data extraction), "telling" the AI is not as good as "showing" it.
- **Few-Shot Learning** means giving 1-3 full examples of the (Input → Output) you want.

Few-Shot Prompt (Data Extraction)

"I need to extract part numbers and materials from unstructured text. Follow the examples.
Text: "The main housing (P/N 401-A) is ABS plastic." **JSON:** {"part": "401-A", "material": "ABS"}
Text: "Use SS 316 for the main shaft (P/N G-102-B)." **JSON:** {"part": "G-102-B", "material": "SS 316"}
Text: "For the mounting plate (P/N H-770), we are switching to Aluminum 6061." **JSON:**"

LLM's Guaranteed Output

```
{"part": "H-770", "material": "Aluminum 6061"}
```

- This is *much* more reliable than trying to describe the rules.

Advanced Practice 3: Iterative Refinement

- Your **first prompt will almost always be bad**. This is normal!
- Prompt engineering is an **iterative process** of refining your prompt based on the AI's output.

Prompt V1

"Analyze the test data."
Output V1: "The data shows some numbers. What analysis do you want?" *(Useless)*

Prompt V2

"I have tensile test data. Find the ultimate tensile strength."
Output V2: "The UTS is 450 MPa. It also shows a yield strength of 270 MPa." *(Better, but missed things)*

Prompt V3 (Good)

"Act as a materials scientist. I'm providing tensile test data (Stress in MPa, Strain in mm/mm) inside triple backticks.
```
[...data...]
```
Perform the following: 1. Calculate the Ultimate Tensile Strength (UTS). 2. Calculate the 0.2% offset Yield Strength. 3. Calculate the Modulus of Elasticity from the linear portion. 4. Summarize the findings in a table."

Advanced Practice 4: Critique & Refinement

- Use the AI as a "sparring partner." Ask it to critique your own work to find blind spots.
- This forces the AI into an analytical, critical mode instead of an agreeable, generative one.

Critique Prompt

"Act as a senior design engineer with 20 years of experience in automotive design.
I am designing a new suspension A-arm. I have decided to make it from cast Aluminum (A356) to save weight.
Your Task: Critically analyze this decision. List 5 potential **disadvantages** or **risks** of using A356 casting for this application compared to a traditional forged steel component. Focus on durability, failure modes, and manufacturing complexity."

Result

The AI will now give you a valuable list of concerns:

- 1. Porosity in casting leading to fatigue cracks...
- 2. Lower impact strength compared to forged steel...
- 3. Complexity of heat treatment (T6)...
- ...etc.

Advanced Practice 5: Prompting for Code (Python/MATLAB)

- LLMs are incredibly powerful code generators. You can use them to automate your data analysis and scripting tasks.
- Be *extremely specific* about libraries, variable names, and input/output formats.

Code Generation Prompt

"Write a **Python** script.

1. Use the **'pandas'** library to read a CSV file named **'tensile**$_{d}$*ata.csv*'.2.*TheCSVhastwocolumns* :

'Strain (mm/mm)'*and***'Stress (MPa)'**.3.*Usethe***'matplotlib'***librarytocreateascatterplotofStressvs.Strain*.4.*LabeltheX* − *axis*" *Strain*(*mm*/*mm*)" *andtheY* − *axis*" *Stress*(*MPa*)" .5.*Titletheplot*" *Stress* − *StrainCurve*" .6.*SavetheplotasaPNGfilenamed*

Result

A complete, working Python script that does exactly what you asked, saving you 20 minutes of looking up syntax.

Prompt Engineering: Key Takeaways

- Prompting is an **iterative skill**, not a one-shot command.
- **Be specific.** The AI is not a mind-reader.
- Use the 5 Elements: **Role, Task, Context, Format, Constraints**.
- For complex tasks, give **examples** (Few-Shot) or ask the AI to **think step-by-step** (CoT).
- For structured data, be *extremely* specific about formats (e.g., Table, JSON, CSV).
- For code, be *extremely* specific about libraries and variable names.

Your Final Year Project

Start using these techniques **today** for your project work, report writing, and data analysis. You will save dozens of hours.

Topic 3 Goes Here

How to build LLMs into your own tools and workflows.

What is an API?

Application Programming Interface

- An API is a **messenger** that lets two pieces of software talk to each other.
- You don't need to know *how* the LLM works (the Trillions of parameters). You just need to know *how to ask it a question* in a format it understands.

The "Restaurant" Analogy

- **You (Your App):** The customer.
- **The LLM (e.g., GPT-4):** The kitchen (a complex system).
- **The API:** The **waiter**.

1. You (your app) don't go into the kitchen.
2. You give a structured **request** (your prompt) to the waiter (the API).
3. The waiter (API) takes it to the kitchen (LLM).
4. The waiter (API) brings back the **response** (the generated text).

How an API Call Works (Conceptual)

Your Python Script → The LLM

1. Your Python Script (or App)

You write a few lines of code (using a library like ‘openai‘ or ‘google.generativeai‘)

```
client = OpenAI()
response = client.chat.completions.create(
          model="gpt-4o",
          messages=[
               {"role": "user",
               "content": "What is...?"}
          ]
)
```

2. The API Request (JSON)

Your code sends a **request** over the internet. This request is just text, formatted in **JSON**.

```
{
          "model": "gpt-4o",
          "messages": [
               {"role": "user",
               "content": "What is...?"}
          ]
}
```

↓ (OpenAI / Google / Anthropic Servers Process Request) ↓

3. The API Response (JSON)

The server sends back a **response**, also in JSON.

```
{
          "id": "chatcmpl-...",
          "choices": [
               {"message": {
                    "role": "assistant",
                    "content": "The answer is..."}
          ]}
}
```

What is an API Key? (And Why You MUST Protect It)

- An API Key is a unique, secret password (like `sk-a...b...c...123...`).
- It proves to the API provider (like OpenAI) that the request is **from you**.
- This is how they **bill your account** for the tokens you use.

CRITICAL WARNING

- **NEVER, EVER** share your API key.
- **NEVER, EVER** paste your API key into your code (like a Python script) and upload it to a public place like **GitHub**.
- Bots are constantly scanning GitHub for leaked keys. If your key is found, an attacker will use it to spend **thousands of dollars** on your account in minutes.
- **How to use it safely:** Store it as an "Environment Variable" or in a secure "secrets" file that is *never* committed to your code repository.

Understanding API Costs: You Pay Per Token

- You are billed for **both** the tokens you **send** (your prompt) and the tokens you **receive** (the AI's answer).
- Models have different prices. Generally: **Smarter model = More expensive.**

Example Pricing (Hypothetical)

- **Model A (Fast, "Dumb"):**
  - $0.450 per 1 million input tokens
  - $1.50 per 1 million output tokens
- **Model B (Smart, "Genius"):**
  - $5.00 per 1 million input tokens
  - $15.00 per 1 million output tokens

Engineering Implication

- **RAG is expensive!** Your "context" (the documents you "stuff" in the prompt) counts as **input tokens**.
- A 100-page document is *much* more expensive to process than a 1-page document.
- This is why good "Retrieval" (Step 1 of RAG) is so important – you only want to retrieve the *single most relevant paragraph*, not the whole book.

The API Landscape: Key Players

Company	Key Models (via API)	Primary Characteristic
OpenAI	GPT-4o, GPT-4 Turbo, GPT-3.5	The "market leader." High performance, very popular, feature-rich. (Closed Source)
Google	Gemini 1.5 Pro, Gemini 1.0	Excellent performance, *massive* context windows (1M tokens). Deeply integrated with Google Cloud. (Closed Source)
Anthropic	Claude 3 (Opus, Sonnet, Haiku)	"Safety-first" approach. Excellent reasoning and writing, strong on ethics. Good cost/performance balance. (Closed Source)
Meta	Llama 3	**Open Source.** Not an API itself, but you can download and run this model on *your own servers* (on-premise) for free.
Hugging Face	(Thousands of models)	The "GitHub for AI." A platform where you can access and host thousands of different models, including open source ones.

Key Decision: Open vs. Closed Source Models

Closed Source (e.g., GPT-4, Claude 3)

Analogy: Using a public cloud service (like Google Drive).

- **Pros:**
- Easiest to use (just an API call).
- Always have the latest, most powerful "genius" model.
- No hardware or maintenance.
- **Cons:**
- Pay per token (can get expensive).
- Your data (the prompt) leaves your network. (It's secure, but for "Top Secret" data, this is a non-starter).
- Relies on their servers being online.

Open Source (e.g., Meta's Llama 3)

Analogy: Hosting your own private server in your building.

- **Pros:**
- **Total Privacy.** Data never leaves your server. This is the *only* option for defense, IP-sensitive R&D, etc.
- **No per-token cost.** (You pay for the hardware).
- You can fine-tune it to be a deep, deep expert.
- **Cons:**
- **Requires hardware.** You must buy and maintain very expensive GPU servers ($$$+).
- Requires IT and ML expertise.
- Models are *good*, but usually 6-12 months

ME Application 1: Internal Knowledge Chatbot

- **The Problem:** Your company's knowledge (test reports, CAD standards, best practices, previous project "lessons learned") is locked in thousands of PDFs and web pages. It's impossible for a new engineer to find anything.

The Solution (using RAG + API)

1. You build a simple web app (a chat box).
2. You "index" all your company documents into a **Vector Database**.
3. When an engineer asks a question:
   - Your app **retrieves** the relevant documents from the database.
   - Your app **augments** the prompt.
   - Your app **calls the LLM API** (e.g., GPT-4) with the augmented prompt.
   - The app displays the answer.

Result: "Chat with your Engineering Standards"

Your new-hire engineer can now ask: "What is our company standard for fillet radius on a cast aluminum part?" and get an instant, correct answer with a source link.

ME Application 2: Automated Report Generation

- **The Problem:** You just ran 50 tensile tests. You have a giant CSV file of raw data. You now have to spend 4 hours writing the "methods" and "results" sections of the report.

The Solution (using API + Prompting)

- You write a **Python script** that:
  1. Loads your CSV file (e.g., using ‘pandas‘).
  2. Calculates basic stats (mean, std dev, max) on the data.
  3. Creates a **master prompt** that includes:
     - A template for the report ("Introduction," "Methods," etc.)
     - The calculated stats.
     - A few examples of the raw data.
     - A good prompt (e.g., "Act as a materials scientist...").
  4. The script **calls the LLM API** with this giant prompt.
  5. The API returns the fully-written *first draft* of your report.

Result: 4 Hours of Work → 5 Minutes

The AI writes 90% of the report. You, the engineer, just need to **review, edit, and validate** it before signing off.

ME Application 3: CAD & CAE Co-Pilots (The Future)

- **The Problem:** Complex CAD/CAE software (like SolidWorks, ANSYS) has thousands of commands and a steep learning curve.

The Solution (API Integration)

- CAD companies are building LLM APIs *directly into their software*.
- This creates a "co-pilot" that understands natural language.
- **You (the engineer):** "Create a 5mm fillet on all edges of this bracket. Now run a static structural analysis with a 500N load on the top face and a fixed support on the bolt holes."
- **The AI Co-Pilot:**
  1. It parses your text.
  2. It translates "create a fillet" into the actual software *API commands* (e.g., ‘swModel.FeatureManager.InsertFillet(...)‘).
  3. It translates "run analysis" into the 20 steps required to set up the FEA.

Result: Text-to-CAD / Text-to-Simulation

This automates the "clicking" and lets you focus on the **engineering intent**. Your job becomes describing *what* you want, not clicking the 50 buttons to get it.

ME Application 4: Summarizing Technical Papers

- **The Problem:** For your Final Year Project, you have 20 research papers to read on "vibration damping in composite materials." This will take days.

The Solution (using a model with a Large Context Window)

- You use a model like Gemini 1.5 Pro (1M token window).
- You upload **all 20 papers at once**.

The Prompt

"You are a research assistant. I have uploaded 20 papers on vibration damping.
Your task is to analyze all of them and generate a literature review summary.
1. What are the 5 most common damping techniques mentioned? 2. Which materials are most frequently cited as being effective? 3. Generate a table comparing the damping coefficients found in papers [3], [7], and [12]. 4. What is a key research gap that multiple papers mention?"

Result: Days of Research $\rightarrow$ One Hour

The AI acts as a research assistant to synthesize knowledge across a huge volume of text instantly.

ME Application 5: Generating FMEA Drafts

- **The Problem:** Creating a Failure Modes and Effects Analysis (FMEA) is tedious, manual, and relies on brainstorming all possible failures.

The Solution (using a "Brainstorming" prompt)

- You feed the AI a description of your system.

The Prompt

"Act as a reliability engineer. I am designing a hydraulic actuator for aircraft landing gear.
The system consists of: - A 3000 psi hydraulic cylinder - A main piston with two elastomer O-rings - A steel piston rod - Hydraulic fluid (MIL-H-5606)
Task: Brainstorm a list of potential **failure modes** for this system. For each failure mode, list a potential **cause** and a potential **effect** on the landing gear.
Format as a 3-column table."

Result: A Solid First Draft

The AI will instantly generate a table with 10-15 rows (e.g., "Failure: Internal Leak", "Cause: O-ring wear/extrusion", "Effect: Slow/failed gear extension..."). This provides a **starting point** for your team to review and refine.

ME Application 6: Root Cause Analysis (RCA)

- **The Problem:** A part failed in the field. You have the failure report, and you need to brainstorm *why*.

The Solution (using a "Critique" prompt)

- You provide the "facts" of the failure.

The Prompt

"Act as a failure analysis expert. A P-401A bracket (cast A356-T6 aluminum) failed in the field after 2 years of service.
Observed Facts: 1. The machine it was on experiences high-frequency vibration (200-300 Hz). 2. The failure was a fatigue crack that initiated at a sharp 90-degree internal corner. 3. There was no sign of corrosion or overload (no plastic deformation).
Task: Generate a "5 Whys" analysis to determine the potential root cause. Also, list 3 distinct hypotheses for the root cause."

Result: Structured Brainstorming

The AI will provide a structured analysis:

- H1: **Design Flaw:** The sharp internal corner acted as a stress concentrator...
- H2: **Manufacturing Flaw:** Casting porosity at the corner...
- H3: **Analysis Flaw:** The original FEA did not correctly model the 200Hz vibration...

ME Application 7: Supply Chain Analysis

- **The Problem:** Your primary supplier for M12 Titanium bolts (Grade 5) just had a factory fire. You need to find alternatives *fast*.

The Solution (using a "Search" prompt)

- Using an LLM connected to the internet (or an internal purchasing database via RAG).

The Prompt

"Act as a supply chain specialist for a high-performance engineering firm.
I need to find alternative suppliers for: '**M12 x 40mm, Grade 5 Titanium (Ti-6Al-4V) Socket Head Cap Screw**'.
Task: 1. List 5 potential suppliers in North America or Europe. 2. For each supplier, provide a link to the product page. 3. Check if any of them mention **AS9100 or ISO 13485 certification**.
Format this as a table."

Result: Instant Market Research

The AI will perform hours of Google searching and data extraction in seconds, giving you an actionable list to start calling.

ME Application 8: Data Extraction from PDFs

- **The Problem:** You have 50 material data sheets (PDFs) from different suppliers, all in different formats. You need to get the "Yield Strength" and "Elastic Modulus" from all of them into an Excel sheet.

The Solution (using a script + "Few-Shot" prompt)

- You write a Python script that: 1. Loops through all 50 PDF files. 2. Extracts the raw text from each PDF. 3. For each text, calls the LLM API using a "Few-Shot" prompt (like Slide 25) to find and format the data.

The (simplified) Prompt (inside the loop)

"Extract the Yield Strength (MPa) and Elastic Modulus (GPa) from the text.
[Example 1...] [Example 2...]
Text: ``` [Raw text from PDF] ```
JSON:"

Result: Automated Data Entry

The script runs and, in 10 minutes, produces a structured JSON or CSV file with all the data, saving you a full day of mind-numbing copy-pasting.

Summary of ME Applications

Application	Key Benefit (What it Automates)
1. Knowledge Chatbot (RAG)	"Finding things" - Instantly query internal standards, reports, and procedures.
2. Report Generation	"Writing" - Create first drafts of test reports from raw data/notes.
3. CAD/CAE Co-Pilot	"Clicking" - Automate repetitive CAD modeling and simulation setup.
4. Literature Summary	"Reading" - Synthesize and compare dozens of research papers at once.
5. FMEA Generation	"Brainstorming" - Create a first draft of failure modes for a new design.
6. Root Cause Analysis	"Analyzing" - Generate structured hypotheses for a field failure.
7. Supply Chain Search	"Googling" - Perform rapid market research for alternative parts/suppliers.
8. Data Extraction (PDF)	"Copy-Pasting" - Pull structured data (like material properties) from unstructured text.

The Engineer as the "Conductor"

Module 5: Key Takeaways (1/2)

1. **LLMs are Next-Token Predictors.** Their "intelligence" is an emergent property of predicting text at a massive scale.
2. **Tokens, Context Window, and Parameters** are the key concepts that define a model's capabilities, memory, and cost.
3. **Embeddings** are the "math of meaning," allowing computers to find semantically similar concepts.
4. **RAG (Retrieval-Augmented Generation)** is the *single most important pattern* for engineering. It allows LLMs to use your private, up-to-date data securely.
5. **Vector Databases** are the high-speed search engines that make RAG possible by finding the "closest" vector (meaning).

Module 5: Key Takeaways (2/2)

6. **RAG vs. Fine-Tuning:** Use RAG to "teach facts" (like a patient chart). Use Fine-Tuning to "teach skills" (like a medical specialty).
7. **Prompt Engineering is an Iterative Skill.** Start simple, and refine. Use CoT, Few-Shot, and delimiters to get precision.
8. **APIs are the "Waiters."** They are the simple messengers that let your apps (Python, CAD) use the power of a complex LLM, opening the door for powerful automation.
9. **Protect Your Keys!** An API key is a password for your money. Never, ever post it to GitHub.

The Future: What's Next?

1. Multimodality (It's already here)

- Models that understand text, **images, audio, and video** all at once.
- **ME Example:** You upload a *photo* of a failed part and ask, "Based on this image, what is the likely failure mode? Is this brittle fracture or ductile?"

2. On-Device LLMs

- Smaller, efficient models (like Llama 3 8B) that run directly on your **laptop or phone**.
- **ME Example:** An AR headset for a factory technician that "sees" a machine and, using a local LLM, provides repair instructions without needing Wi-Fi.

3. AI Agents

- LLMs that can **take action**. They can "use tools" (like browse the web, run code, or use an API).
- **ME Example:** "Design a bracket for this load case." The Agent runs a generative design (Tool 1), then runs an FEA (Tool 2), then writes the report (Tool 3) - all from one prompt.

The Engineer's New Role

- Your job is **not** "prompt engineer."
- Your job is **Mechanical Engineer**.
- The LLM is a new, powerful tool, just like SolidWorks or ANSYS.
- An LLM automates **text, code, and knowledge tasks**, just as CAD automated **drawing tasks**.

The Engineer as the "Orchestra Conductor"

- You don't need to play every instrument.
- You have a "Generative Design" player, an "FEA" player, and now an "LLM" player.
- Your job is to be the **domain expert** who knows **what** problem to solve, **which tool** to use, and **how to validate** the final result.

The AI Hallucinates.

It will confidently invent:

- **Material properties**
- **Engineering standards**
- **Equations**
- **Part numbers**

Your Job is Not to Trust. Your Job is to VERIFY.

An AI's output is a **hypothesis**, not a fact. It's a "first draft" for you, the engineer, to **check, validate, and approve**. You are always the human in the loop.

Module 5: Summary

- We learned what LLMs are (next-token predictors) and their core concepts (tokens, context windows, embeddings).
- We learned the **critical difference** between teaching an AI facts (RAG) and teaching it skills (Fine-Tuning).
- We learned **advanced prompt engineering** techniques (CoT, Few-Shot, Critique, Code) to get precise, expert-level answers.
- We learned how **APIs** work, how they are priced (per token), and why you MUST protect your API keys.
- We explored 8 concrete **ME applications** you can build today, from "chat with your data" (RAG) to automating FMEA and data extraction.

Questions?

Prof. Dinesh Kumar
dinesh.kumar@dypiu.ac.in

Course: Product Design and Development