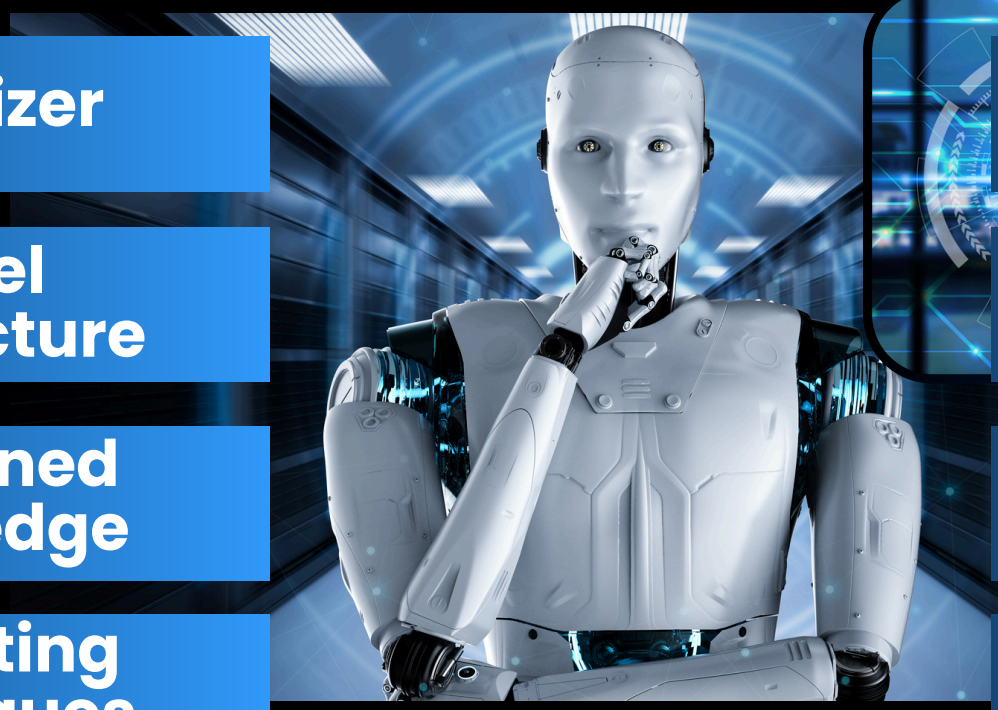
Naresh Edagotti
@Statfusionai

CORE COMPONENTS OF LARGE LANGUAGE MODELS YOU SHOULD KNOW



- Tokenizer
- Model Architecture
- Pretrained Knowledge
- Prompting Techniques
- Context Window
- Tool Use / External APIs
- Retrieval (RAG)
- Self-Reflection Techniques

What Makes an LLM?

Large Language Models The Building Blocks

LLMs are complex systems built from 8 essential components

- Each component serves a specific purpose in the AI pipeline
- Understanding these components helps in better LLM implementation
- From raw text processing to intelligent responses

Key Insight

- LLMs aren't just "black boxes" - they're engineered systems with distinct, understandable parts working together.

Tokenizer

From Text to Numbers

What it does:

- Converts human text into numerical tokens the model can understand
- Breaks down words, subwords, or characters into standardized units
- Handles vocabulary mapping and encoding/decoding

Key Technologies:

- Byte Pair Encoding (BPE) - Most common approach
- SentencePiece - Google's tokenization method
- WordPiece - BERT's tokenization strategy

Why it matters:

- Determines how efficiently the model processes language
- Affects model performance across different languages
- Critical for handling out-of-vocabulary words

Example

"Hello world!" → [15496, 995, 0] (token IDs)

Model Architecture

Neural Network Design

Core Architecture Types:

- **Transformer Architecture -** The foundation of modern LLMs
- **Encoder-Decoder vs. Decoder-** Only models
- **Attention Mechanisms -** The key breakthrough

Key Components:

- **Multi-Head Attention -** Parallel processing of relationships
- **Feed-Forward Networks -** Non-linear transformations
- **Layer Normalization -** Stabilizes training
- **Positional Encoding -** Understands sequence order

Popular Architectures:

- **GPT (Generative Pre-trained Transformer) -** Decoder-only
- **BERT (Bidirectional Encoder) -** Encoder-only
- **T5 (Text-to-Text Transfer) -** Full encoder-decoder

Scale Matters

From millions to trillions of parameters

Pretrained Knowledge (Weights)

What the Model "Knows"

What are Weights?

- Numerical parameters learned during training
- Encode patterns, relationships, and knowledge from training data
- Billions to trillions of interconnected values

Training Process:

- Pretraining - Learning from massive text datasets
- Fine-tuning - Adapting to specific tasks
- Reinforcement Learning from Human Feedback (RLHF) - Alignment

Knowledge Storage:

- Factual information embedded in weight matrices
- Language patterns and grammatical rules
- Reasoning capabilities and common sense

Challenge

Knowledge is distributed and not easily interpretable

Prompting Techniques

How We Communicate with LLMs

Core Prompting Strategies:

- Zero-shot - Direct task instruction without examples
- Few-shot - Providing examples within the prompt
- Chain-of-Thought - Step-by-step reasoning prompts

Advanced Techniques:

- Role-based prompting - "You are an expert in..."
- Template-based prompting - Structured input formats
- Instruction tuning - Training on instruction-following tasks

Best Practices:

- Be specific and clear in instructions
- Use examples for complex tasks
- Iterate and refine prompts based on outputs

Impact

Good prompting can dramatically improve model performance

Context Window

Memory Limitations

What is Context Window?

- The maximum amount of text the model can process at once
- Measured in tokens (typically 2K to 2M+ tokens)
- Both input and output count toward this limit

Context Window Sizes:

- GPT-3.5: 4K tokens (~3,000 words)
- GPT-4: 8K-128K tokens
- Claude: Up to 200K tokens
- Newer models: 1M+ tokens

Challenges:

- Information beyond context window is "forgotten"
- Longer contexts require more computational resources
- Managing context efficiently is crucial for performance

Solutions

Sliding windows, summarization, and context compression

Tool Use / External APIs

Expanding LLM Capabilities

What is Tool Use?

- LLMs calling external functions and APIs
- Bridging the gap between language understanding and action
- Enabling real-world task completion

Common Tools:

- Calculators - Mathematical computations
- Web Search - Real-time information access
- Database Queries - Data retrieval and analysis
- Code Execution - Running and testing code
- API Integrations - Third-party service access

Implementation:

- Function calling frameworks
- Agent-based architectures
- Structured output formats (JSON, XML)

Benefits

- **Transforms LLMs from text processors to capable agents**

Retrieval (RAG)

Retrieval-Augmented Generation

What is RAG?

- Combines LLM generation with external knowledge retrieval
- Searches relevant documents before generating responses
- Provides up-to-date and domain-specific information

RAG Pipeline:

1. Query Processing - Understanding user intent
2. Document Retrieval - Finding relevant information
3. Context Integration - Combining retrieved content with prompt
4. Response Generation - Creating informed answers

Key Components:

- Vector Databases - Semantic search capabilities
- Embedding Models - Converting text to searchable vectors
- Chunking Strategies - Optimal document segmentation

Advantages

Reduces hallucination, enables knowledge updates, domain expertise

Self-Reflection Techniques

Making LLMs Think About Their Thinking

What is Self-Reflection?

- LLMs evaluating and improving their own outputs
- Metacognitive approaches to enhance reasoning
- Quality assurance and error correction mechanisms

Key Techniques:

- Self-Correction - Reviewing and revising responses
- Chain-of-Verification - Fact-checking generated content
- Constitutional AI - Following predefined principles
- Multi-step reasoning - Breaking down complex problems

Implementation Methods:

- Prompt-based reflection ("Check your work")
- Multi-agent systems (different roles for generation and evaluation)
- Iterative refinement processes

Benefits

Improved accuracy, reduced hallucination, better reasoning quality

Putting It All Together

The Complete LLM System

Integration Flow:

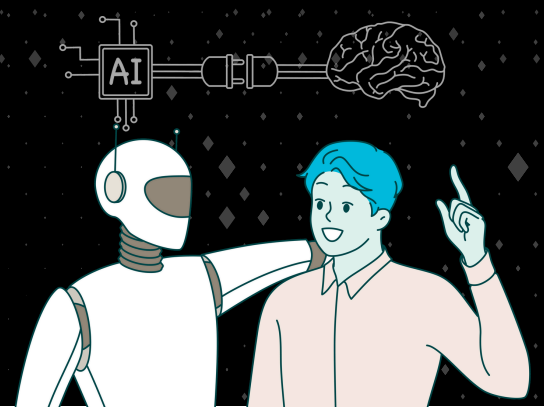
1. **Input** → Tokenizer → Model Architecture
2. **Processing** → Pretrained Knowledge + Context Window
3. **Enhancement** → Prompting + Tool Use + RAG
4. **Quality** → Self-Reflection → **Output**

Key Takeaways:

- LLMs are engineered systems with distinct components
- Each component addresses specific challenges and capabilities
- Understanding these components enables better implementation and optimization
- The future lies in improving integration between components

For AI Professionals:

- Master each component for better LLM deployment
- Consider component interactions in system design
- Stay updated on advances in each area



LIKE THIS CONTENT?

FOLLOW FOR MORE!

NARESH EDAGOTTI

LIKE REPOST SAVE