ChatGPT

Overview of LLM Architectures and Models

Large language models (LLMs) like GPT, LLaMA, BLOOM, Mistral, etc., are all based on the Transformer architecture [1]. At a high level, a Transformer processes tokenized text by mapping each token to a learned embedding vector, adding positional encoding, and then applying multiple layers of **multi-head self-attention** and feed-forward networks [1] 【68†】. In multi-head attention, the model projects the hidden states into multiple "heads" of queries (Q), keys (K), and values (V), computing attention as

$$\mathrm{Attention}(Q,K,V)=\mathrm{softmax}\Big(\frac{QK^\top}{\sqrt{d_k}}\Big)\,V$$

[1]. The output is then combined and passed through residual ("Add & Norm") and feed-forward layers. Figure 68† shows a schematic of the Transformer: encoders on the left ingest input tokens, while decoders (for autoregressive LLMs) generate output tokens one by one【68†】. All modern LLMs are decoder-only (GPT-style) or encoder–decoder (T5-style) variants of this basic transformer.

Transformer architecture (from Vaswani et al. 2017): Inputs are token embeddings with positional encodings. Each Transformer block has multi-head self-attention and feed-forward sublayers with residual connections. The decoder (right) attends to encoder outputs plus previous outputs when generating text [1] 【68†】.

Each model varies in depth (layers), width (hidden size), number of attention heads, vocabulary, and training data. For example, OpenAI's GPT-3 (175B) uses 96 layers, hidden size 12,288, and 96 attention heads [2]. Meta's LLaMA2-7B has 32 layers, hidden size 4096, and 32 heads [3] [4]. EleutherAI's GPT-J-6B uses 28 layers, hidden size 4096, and 16 heads [5]. Mistral-7B also has 32 layers and hidden size 4096 (with 32 heads) [6]. All these models use a context window (sequence length) on the order of thousands of tokens (e.g. 2048 for GPT-3/GPT-J [2] [5]; 4096 for LLaMA2 [7]; 8192 for Mistral-7B [6]). Tokenization is typically Byte-Pair Encoding (BPE) or variants (e.g. GPT/BLOOM use byte-level BPE [8]). The embedding matrix (size = `vocab_size × hidden_size`) maps each token ID to a continuous vector; for instance, GPT-3's embedding is 12,288-dimensional [2].

The total number of parameters can be computed from these dimensions. For a decoder-only Transformer with hidden size E, vocab size V, context length P, and L layers, one can show the parameter count is:

$$C = E(V+P) + L(12E^2 + 13E) + 2E,$$

which for GPT-2 (768×50257 vocab, 12 layers) gives ~124M [9]. In general, parameters = embedding params ($E\times V$) plus positional ($E\times P$), plus each layer's W_Q,K,V and W_O ($\approx12E^2$) and MLP weights ($\approx2\times4E^2$) [9]. This scaling explains why very large models have billions of params.

Popular Open-Source and Proprietary LLMs

Below are key examples of widely-used LLMs and chatbots, focusing on open-source models (with run-local availability) and some notable proprietary ones:

- **OpenAI GPT series**: *GPT-2* (up to 1.5B, 1024-token context), *GPT-3* (175B, 2048 context) [2], *GPT-3.5* (175B with RLHF/instruction-tuning), and *GPT-4* (undisclosed >170B with even longer context). These are decoder-only models trained on massive corpora (GPT-3: ~300B tokens, web text, books) with next-token prediction. GPT-3.5/4 were further fine-tuned with **Reinforcement Learning from Human Feedback (RLHF)** to optimize for helpfulness, making them suitable as chatbots (e.g. ChatGPT). The models use standard sinusoidal or learned positional encodings (GPT-2/3 use learned fixed embeddings) and byte-level BPE tokenization [2] [8].

- **Meta LLaMA series**: *LLaMA-1* (7B–65B, 2048 context, RoPE positional encodings) and *LLaMA-2* (7B, 13B, 70B; 4096 context) [7]. These decoder-only models (hidden size 4096 for 7B) use rotary position embeddings (RoPE) [10] and were trained on 2 trillion tokens of public data [7]. Meta also released "instruction-tuned" LLaMA-2-chat models aligned by RLHF. LLaMA models are source-available for research (Meta's license). Community finetunes include Alpaca-7B and Vicuna-7/13B (Stanford/others) which apply supervised finetuning on user-shared instruction datasets.

- **EleutherAI Models**: *GPT-Neo/J/NeoX* (1.3B, 2.7B, 6B, 20B) – open, trained on "The Pile" (825GB). GPT-J-6B is a popular 6B model (28 layers, rotary embeddings, 2048 context) [5]. *Pythia* (by EleutherAI) provides multiple checkpoints (70M–12B) with known training details for research. These models use similar architectures to GPT-3 but often with parallelized QKV computations and rotary embeddings (as in GPT-J) [11].

- **BigScience BLOOM**: An open-access 176B multilingual model (decoder-only, 46 languages) [12]. It introduced ALiBi positional biases and applies layer-norm on embeddings [12]. BLOOM uses a 250K-token BPE vocabulary to cover many languages [8]. Smaller BLOOM variants (0.6B–7.1B) exist too.

- **Falcon (TII)**: *Falcon-7B* and *Falcon-40B* are open decoder models (40B trained on RefinedWeb) with 2048 or higher context. They achieved strong language and code performance. (Falcon 40B uses 4096 context by default, and can be extended to 10k [13].)

- **Mistral**: *Mistral-7B* is a 7.3B open model (Apache 2.0 license) released late 2023. It uses hidden size 4096, 32 heads, **Grouped Query Attention (GQA)** and a **sliding-window attention** spanning 4096 tokens per layer [14] [6]. This gives an effective context up to 8192 tokens [6] while reducing cost. Mistral outperforms much larger models on many benchmarks, and an instruction-tuned variant (Mistral-7B Instruct) rivals 13B chat models [14] [6].

- **Specialized/chatbot models**: For chat applications, many base LLMs are finetuned or chained. *GPT-3.5/4* (OpenAI) and *Claude* (Anthropic, proprietary) are fine-tuned with RLHF. Open models like LLaMA or Falcon can be instruction-tuned (e.g. Vicuna, Alpaca). Frameworks like **LangChain** allow combining LLM calls, retrieval, and memory. LangChain "chains" LLM steps (e.g. document retrieval → question answering) in sequence. **LangGraph** is a newer LangChain library for stateful, multiagent workflows (graph of actions/nodes) [15]. LangGraph enables persistent context and loops, useful for complex assistants; LangChain is suited for linear pipelines [15].

In general, these LLMs share the same core architecture (Transformers) but differ in scale, training data, positional encoding (absolute vs rotary vs ALiBi), and special techniques (sparsity, mixture-of-experts, etc.). Fine-tuning methods vary: some models are **instruction-tuned** by supervised learning on Q&A datasets (e.g. Flan-T5, Alpaca) or by RLHF (ChatGPT). Others offer adapters (LoRA) for efficient fine-tuning. Open-source toolkits like Hugging Face Transformers provide implementations for most of these models.

Tokenization, Embeddings, and Token Generation

LLMs operate on tokens (subwords) rather than raw characters. Input text is first **tokenized** (often via Byte-Pair Encoding or SentencePiece) into a sequence of token IDs. An embedding layer then maps each ID to a learned dense vector (size = *hidden_dim*). For example, GPT-3 has a vocabulary of 50,257 BPE tokens and maps each to a 12,288-dimensional vector [2]. A positional embedding (sinusoidal, RoPE, or ALiBi) is added to encode token order.

These embedding vectors are processed through the Transformer layers. After the final layer, the model applies a linear projection (often tying weights with the input embeddings) to produce **logits** over the vocabulary. A softmax then yields a probability distribution: for hidden state h_t at time t,

$$P(\text{token} = i \,|\, \text{context}) = \frac{\exp(W_i^\top h_t)}{\sum_j \exp(W_j^\top h_t)},$$

where W is the output weight matrix. The model is trained to maximize the likelihood of the next token (cross-entropy loss). At generation time, the highest-probability token or a sampled token (possibly using temperature, top-k or nucleus sampling) is chosen to produce text. For example, top-k sampling restricts to the k most probable tokens to reduce randomness [2].

For **RAG (Retrieval-Augmented Generation)**, embeddings play a key role. A user query is encoded into a vector (via a sentence/embedding model) and compared with a vector index of documents (via cosine similarity) [16]. Relevant documents are retrieved and concatenated into the prompt for the LLM, giving it factual context. In practice, one uses libraries (e.g. LangChain) to compute document embeddings and query embeddings with models like OpenAI's or sentence-transformers. The retrieved text guides generation, mitigating hallucinations [17] [16].

Training, Fine-Tuning, and Inference

Most LLMs are initially **pretrained** on vast text corpora (unsupervised next-token prediction). For instance, OpenAI's GPT-3 was trained on ~300B tokens of web text and books. After pretraining, **fine-tuning** adapts the model: - *Supervised fine-tuning (SFT)* on curated datasets (e.g. Q&A pairs) for specific tasks. - *RLHF*, where human feedback on outputs is used to train a reward model and further optimize the LLM (as with GPT-3.5/GPT-4 chat models). - *LoRA/QLoRA* techniques train low-rank adapters to reduce compute.

Fine-tuning is done with toolkits like Hugging Face Transformers, DeepSpeed, or off-the-shelf services. For example, Alpaca and Vicuna were created by fine-tuning LLaMA on instruction-following data. LangChain isn't used to train models directly, but it can orchestrate inference and retrieval (chain of LLM calls). LangGraph's stateful approach can maintain context over a conversation.

During inference, models often use **KV caching**: past keys/values from attention are cached so generation of each new token is efficient (as explained for Mistral [18]). Sliding-window or rotary techniques (like ALiBi in BLOOM) allow long context by decaying attention or overlapping windows [12] [19].

Mathematical Details

Key mathematical components include: - **Self-Attention**: Given input tokens' embeddings in matrix X, we compute $Q=XW_Q,\; K=XW_K,\; V=XW_V$, then

$$\mathrm{Attention}(X) = \mathrm{softmax}\big(\tfrac{QK^\top}{\sqrt{d_k}}\big)V,$$

where $\sqrt{d_k}$ normalizes dot products [1]. Multi-head simply does this in parallel and concatenates results. - **Feed-Forward Network (FFN)**: A two-layer MLP in each block: $\text{FFN}(x) = \text{GeLU}(xW_1+b_1)W_2 + b_2$, often with hidden size 4× input size. - **LayerNorm** (or RMSNorm as in LLaMA) is applied around sublayers for stability. - **Embedding/Output Projection**: The input embedding matrix W_e is size $V\times E$. The output logit matrix W_o is often tied to $W_e^\top$.

The **parameter count** scales as noted above [9]. For example, GPT-2 (124M params) with $E=768$, $V=50257$, $P=1024$, $L=12$ fits the formula $C=E(V+P)+L(12E^2+13E)+2E$ [9]. Larger models simply have much larger E,L,V. The feed-forward layers (with $4E^2$ weights each) often dominate the count for big models.

Model Comparison

Model	Params	Layers×Heads	Hidden Dim	Context	Positional Embedding	Vocab/ Tokenization	Training Data	Notab Use / Licens
GPT-3	175B	96×96	12288	2048	Learned absolute	50K BPE (byte-level)	~300B tokens (Common Crawl, etc)	Close (Open API)
GPT-2	1.5B	36×16	1600	1024	Learned absolute	50K BPE (byte-level)	WebText (~8M docs)	Open licens
GPT-J 6B	6B	28×16	4096	2048	Rotary (RoPE)	50K BPE	The Pile (800B tokens)	Open (Apac 2.0) [5

Model	Params	Layers×Heads	Hidden Dim	Context	Positional Embedding	Vocab/ Tokenization	Training Data	Notab Use / Licens
GPT-NeoX 20B	20B	44×32	6144	2048	Rotary	50K BPE	The Pile	Open
BLOOM 176B	176B	70×140	14080	2048	ALiBi	250K BPE (multilingual) [8]	ROOTS corpus (46 lang, 46B tokens)	Open licens [12]
LLaMA-2 7B	7B	32×32	4096	4096	Rotary	32K SPM (English)	2T tokens (mix of sources) [7]	Sourc availa (Meta licens [20]
LLaMA-2 70B	70B	80×128	8192	4096	Rotary	32K SPM	2T tokens	Sourc availa (Meta licens [7]
Mistral 7B	7.3B	32×32 (GQA)	4096	8192*	Rotary	32K SPM (mostly English)	1.3T tokens (public web)	Open (Apac 2.0) [1 [6]
Falcon 40B	40B	72×128	8192	2048*	RoPE	65K BPE	RefinedWeb (~1T tokens)	Open (MPL-
Pythia 12B	12B	24×32	4096	2048	Rotary	50K BPE	Pile (fewer tokens)	Open BY 4.0
T5 (XXL)	11B	24×16 (enc) + 24×16 (dec)	4096	5120*	Absolute sin/cos	32K SPM (SentencePiece)	C4 dataset (750GB)	Open (Apac 2.0)
CodeLlama 7B	7B	32×32	4096	4096	Rotary	32K SPM	Code + text (source data)	Open (Apac 2.0)
ChatGLM-6B	6B	28×32	4096	2048	Rotary	~1K Chinese BPE/tokenizer	Chinese+web corpora	Open (Apac 2.0)
Claude 3 Opus	100B*	? (Mixture?)	?	~9000	Unknown	Unknown	Custom (web, books)	Propri (Anthr

Model	Params	Layers×Heads	Hidden Dim	Context	Positional Embedding	Vocab/ Tokenization	Training Data	Notab Use / Licens
GPT-4	>170B*	?	?	8192	Likely Rotary	~50K BPE	Custom (web, code, etc)	Propri (Open

Estimated or approximate; many proprietary models don't disclose full specs. "" indicates context extension (e.g. Mistral uses sliding-window to effectively 8k even if layers attend 4k at a time). "GQA" = Grouped-Query Attention (used in large LLaMA2/Falcon), "SPM" = SentencePiece model.

This table highlights differences: model size, structure, context window, tokenization, training data volume, and openness. For example, **LLaMA-2 7B** (32×32 blocks, hidden 4096) was trained on 2T tokens [7], while **GPT-3** (96×96, hidden 12288) used ~300B tokens. **BLOOM** (176B) is notable for multilingual training and ALiBi, whereas **Falcon/Mistral** achieve strong performance with leaner sizes (by using GQA or sparse attention). Fine-tuned variants (Vicuna, Alpaca, Dolly, etc.) are based on these families, and specialized models like CodeLlama focus on code.

Key Takeaways for Tuning and Comparison

- **Hyperparameters**: Key model hyperparameters include layer count L, hidden size E, number of heads H (usually $H=E/64$ or similar), feed-forward size (often $4E$), and vocab size V. These determine model capacity (param count) as in the formula [9]. Training hyperparameters (batch size, learning rate, total tokens, LR schedule) crucially affect convergence but are largely opaque for closed models. For open models, published info includes global batch (LLaMA2 used 4M-token batch) and token budgets [7].
- **Fine-tuning vs Chains**: Fine-tuning changes the model weights to target tasks (e.g. instruction following via RLHF or SFT). Alternatively, frameworks like LangChain/LangGraph use the base LLM as-is but manage prompt engineering, retrieval, and multi-step logic. LangChain excels for linear pipelines (document splitting, prompt chain) while LangGraph excels for stateful agents with branching workflows [15].
- **Embeddings & Retrieval**: LLMs produce contextual embeddings (last-layer hidden states) which can be pooled as sentence embeddings. For RAG, one typically uses separate dense embedding models (like `text-embedding-*`) to index documents [21]. Query and doc embeddings are matched via cosine similarity (or other vector math) to find relevant context [16]. Vector databases like FAISS, Pinecone store these vectors for fast retrieval.
- **Token Generation**: The softmax output provides token probabilities [1]. Decoding methods include greedy (argmax), temperature scaling, top-k or nucleus (top-p) sampling. Controlling these sampling parameters affects randomness versus determinism in generation. Understanding how logit adjustment (e.g. biasing or logits manipulation) works is key to steering models.

This comprehensive comparison and architectural analysis should guide model selection and tuning. By understanding each model's size, context limits, and special features (GQA, MoE, attention type), one can predict computational needs and performance trade-offs. It also highlights how to tailor training (finetune vs retrieval chains) and inference settings (sampling parameters, attention mechanisms) for specific domains or tasks.

Sources: Model facts are drawn from official docs and papers [5] [7] [12] [6] and expert reviews [2] [14]. Key formulas and concepts come from standard transformer literature [1] [9] and industry blogs (NVIDIA, AWS) on RAG [17] [16].

---

[1] What are self-attention models?. In the early days of the NLP, wherever... | by Rahul Meka | Medium
https://medium.com/@mekarahul/what-are-self-attention-models-69fb59f6b5f8

[2] The GPT-3 Architecture, on a Napkin
https://dugas.ch/artificial_curiosity/GPT_architecture.html

[3] [4] [10] Introduction to Llama2 : Part-1 Architectural Analysis | by Utsavtiwari | Medium
https://medium.com/@utsavtiwari9936/introduction-to-llama2-part-1-architectural-analysis-3e335e7b1104

[5] [11] GPT-J - Wikipedia
https://en.wikipedia.org/wiki/GPT-J

[6] [18] [19] Mistral 7B Explained!. Mistral 7B is one of the most discussed... | by Pranjal Khadka | Towards AI
https://pub.towardsai.net/mistral-7b-explained-53720dceb81e?gi=15a960f75aec

[7] [20] meta-llama/Llama-2-7b · Hugging Face
https://huggingface.co/meta-llama/Llama-2-7b

[8] [12] Review — BLOOM: A 176B-Parameter Open-Access Multilingual Language Model | by Sik-Ho Tsang | Medium
https://sh-tsang.medium.com/review-bloom-a-176b-parameter-open-access-multilingual-language-model-bcff847f8a22

[9] Transformer Math (Part 1) - Counting Model Parameters
https://michaelwornow.net/2024/01/18/counting-params-in-transformer

[13] Extend the context length of Falcon40B to 10k | by Chen Wu | Medium
https://medium.com/@chenwuperth/extend-the-context-length-of-falcon40b-to-10k-85d81d32146f

[14] Mistral 7B | Mistral AI
https://mistral.ai/news/announcing-mistral-7b

[15] ⚙LangChain vs. LangGraph: A Comparative Analysis | by Tahir | Medium
https://medium.com/@tahirbalarabe2/%EF%B8%8Flangchain-vs-langgraph-a-comparative-analysis-ce7749a80d9c

[16] [21] What is RAG? - Retrieval-Augmented Generation AI Explained - AWS
https://aws.amazon.com/what-is/retrieval-augmented-generation/

[17] What Is Retrieval-Augmented Generation aka RAG | NVIDIA Blogs
https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/