Beyond GPT: Understanding the Advancements and Challenges in Large Language Models

Jatin Wig
BSc Computer Science
Birla Institute of Technology and Science (BITS) Pilani
wigjatin2@gmail.com

Abstract

LLMs (Large Language Models) are game-changers for natural language processing, enabling fluent text generation, code completion, and automation across multiple industries. Focusing on models beyond the GPT architecture, this study examines the foundations, advancements, challenges, and future directions of large language models. We will analyze the role of the transformer architecture in modern LLMs, highlight industry-specific applications, assess computational costs, and address key limitations, including bias and hallucination-related phenomena.

1 Introduction

LLMs (Large Language Models) represent a fundamental shift in AI that is changing the way we interact with machines with natural language understanding and generation capabilities. These models are built and trained based on deep neural networks and typically employ architectures like the Transformer, which processes language through self-attention mechanisms, which are capable of capturing long-range dependencies in text. Modern LLMs contain hundreds of billions of parameters, trained on hundreds of gigabytes of text corpora.

The evolution from basic statistical language models (e.g., n-gram models) to contemporary neural techniques has allowed machine to understand the context of any text, nuance, and semantics at a level of sophistication comparable to that of humans. This development led to the Transformer architecture (Vaswani et al., 2017), which allowed for the parallel processing of sequential data via self-attention mechanisms, addressing shortcomings of earlier recurrent architectures.

Researchers typically divide the development lifecycle into two phases:

- **Pre-training**: Unsupervised learning on massive text corpora to acquire linguistic patterns and world knowledge
- **Fine-tuning**: Supervised adaptation for specific downstream tasks through transfer learning

Modern implementations leverage GPU or TPU clusters for distributed training, with frameworks like PyTorch and TensorFlow enabling efficient parallel computation across thousands of processing units.

1.1 Applications and Industry Impact

LLMs demonstrates transformative potential across several sectors through their advantage to process, analyze, and generate human language.

Education: Adaptive learning systems use these LLMs to develop personalized curricula that adapts based on each student's progress. Automated assessment tools can now score open-ended responses with growing precision, and intelligent tutoring systems offer instant feedback, for example systems like Khan Academy's AI tutor show how LLMs can improve accessibility in studies.

Healthcare: Clinical use cases are automated generation of medical notes from conversations between physicians and patients (e.g. Nuance DAX), speeding up the process of literature review for researchers, and patient triage using symptom screening. While LLMs help in structuring the unstructured EHR data, studies indicate a reduction in documentation time to about 30% or so while maintaining accuracy exceeding 95% in the information extraction.

Finance: Applications span sentiment analysis of market news, automated report generation from financial statements, and anomaly detection for fraud prevention. JPMorgan's COiN platform processes 12,000 annual commercial credit agreements in seconds, work that previously required 360,000 human hours.

Software Development: New programming tools such as GitHub Copilot, Cursor AI, Amazon Code Whisperer, Visual Studio IntelliCode and Replit Ghostwriter shows how LLMs can predict fully functional code blocks, point out possible vulnerabilities in the code around them and even generate documentation. Research by Peng et al. (2021) suggests these tools are capable of decreasing actual coding time of the programmer by 35–50% and improving the code quality automatically by providing best-practice suggestions.

Legal Sector: Contract analysis tools like Evisort leverage LLMs to extract key clauses and identify anomalies with 98% accuracy, significantly reducing manual review time. Legal research assistants can surface relevant case law from millions of documents in seconds.

1.2 Energy Consumption in LLMs

LLMs face substantial sustainability challenges:

- **Training Costs**: GPT-3 was trained at an estimated cost of around 1,287 MWh, enough to power 120 average US homes for a year. Newer models Considering scale, models like PaLM-2 need even more resources.
- **Carbon Emissions**: Just training a single large model run can spew more than 500 tons of CO_2, the equivalent of 300 round-trip flights between New York and San Francisco.
- **Hardware Requirements**: Modern models require thousands of high-density GPU/TPU on and running for weeks at a time, with cooling and other associated costs.
- **Optimization Techniques**:
  - Pruning of Model and Quantization
  - Mixed-precision training
  - Sparse attention mechanisms
  - Dynamic neural architectures

Newer techniques such as mixture-of-experts models hold true promise: Switch Transformers 7x more energy efficient than before at equivalent performance

1.3 Limitations of Existing Models

Despite their capabilities, LLMs exhibit several fundamental constraints:

Hallucinations: The ability to create factually wrong but plausible-sounding content is a difficult challenge to meet. For example, medical LLMs may provide false-positive drug interactions 15–20% of the time in uncontrolled settings. We will see this in more details in further section.

Bias Amplification: The biases present in the training data show up in the model outputs. Some of the mitigation approaches are as follows:

- Curated training data
- Adversarial debiasing
- RLHF (Reinforcement Learning from Human Feedback)

Security Vulnerabilities: Attackers can use prompt injection attacks to override safety filters 60% of the time, while model's behavior can be corrupted by data poisoning even with 0.01% malicious training samples.

Computational Limits: The quadratic attention complexity in transformers creates some practical constraints on context window size limiting the document-level comprehension. We will explore this deeply in the further sections.

1.4 Fine-Tuning and Pre-Training

This two phase training paradigm allows for broad capability and task-specific performance:

Pre-Training:

- Generally employs self-supervised objectives e.g. masked language modeling
- Get big datasets (Common Crawl, Wikipedia, texts corpora)
- Cubic scaling of computational costs with parameter count

Fine-Tuning:

- Fine-tunes the base model for the domain/task
- Common approaches:
  - Full parameter tuning
  - Parameter-efficient methods (LoRA, adapter layers)

  - Prompt tuning

- In general 100-1000x less data required than pre-training

Emerging Paradigms:

- Generalization to new tasks with instruction tuning
- Complex reasoning with chain-of-thought prompting
- Factual accuracy with retrieval-augmented generation

2 Fundamental Concepts

2.1 Core Definitions and Relationships

Machine Learning (ML) Its a process to teach computers some patterns from any data and it is defined by:

$$\hat{y} = f_\theta(x) \quad \text{where } \theta^* = \arg\min_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathcal{L}(y, f_\theta(x))] \tag{1}$$

How it works:

- Input: Data, in most cases, csv dataset
- Output: Predictions/decisions according to training
- Core idea: Improves automatically through experience (trial and error.)

Deep Learning (DL) Its a subset of ML that mimics the brain's neural networks to process complex data.

- Compositional nonlinear transformations: $f(x) = f^{(L)}(f^{(L-1)}(\cdots f^{(1)}(x)))$
- Automatic feature engineering via backpropagation
- Scalability through parallelization (GPUs/TPUs)

How it works:

- Layers: Stacked neurons that transform data hierarchically (e.g., edges to shapes to objects in images).
- Automated feature discovery eliminates manual engineering.

Natural Language Processing (NLP) Teaches machines to understand, interpret, and generate human language.

- Syntax modeling (parse trees, dependencies)
- Semantic representation (word senses, relations)
- Pragmatic understanding (context, intent)

2.2 Transformer Architecture

This model was trained at the word-level and introduced the concept of self-attention mechanisms in end-to-end machine learning systems, revolutionizing the way transformers handle sequence-to-sequence tasks. Transformers were introduced in the landmark 2017 paper "Attention Is All You Need" and overcome key pitfalls such as the inability to process entire sequences at once as RNNs(Recurrent Neural Network) and LSTMs(Long Short-Term Memory) do, as well as the ability of Transformers to achieve long-range dependencies more effectively. At its core, the Transformer consists of multiple layers of multi-head self-attention and feed-forward neural network, which calculate the importance of all elements in a sequence relative to one another on-the-fly. The attention mechanism enables the model to attend to pertinent context, regardless of where it resides in the input, meaning whether within the same sentence or between patterns in paragraphs. This architecture is an encoder-decoder model (though models like GPT only have a decoder while BERT only has an encoder), and each component uses residual connections and layer normalization for stable training. Transformers process the entire sequence of input at once, in contrast to recurrent models that process one step at a time. The three main innovations of self-attention, positional encodings, and multi-head attention work in tandem to allow the model to learn representations that include both local and global patterns of data. The formula of self-attention mechanism is:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2}$$

where:

- $Q \in \mathbb{R}^{n \times d_k}$: Query matrix
- $K \in \mathbb{R}^{m \times d_k}$: Key matrix
- $V \in \mathbb{R}^{m \times d_v}$: Value matrix

- $\sqrt{d_k}$: Scaling factor for stable gradients
- **Positional Encoding** Injects sequence order information:
  - For input $X \in \mathbb{R}^{n \times d}$ (sequence length n, dimension d)
  - Positional matrix $PE \in \mathbb{R}^{n \times d}$ where:

$$PE_{(pos,i)} = \begin{cases} \sin\left(pos/10000^{2i/d}\right) & \text{even } i \\ \cos\left(pos/10000^{2i/d}\right) & \text{odd } i \end{cases}$$

  - Final input: $X = X + PE$ (element-wise sum)
- **Key Properties**:
  - Enables attention to learn relative/absolute positions
  - Adds no trainable parameters

Multi-head attention Enables parallel processing of different representation subspaces:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, ..., head_h)W^O \tag{3}$$

Each head computes:

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{4}$$

2.3 Tokenization and Representation Learning

Byte Pair Encoding (BPE) operates through:

1. Initialization of vocabulary with individual characters
2. Iteratively merging of most frequent symbol pairs
3. Stopping when it reaches the target vocabulary size (e.g., 50,257 for GPT)

Embedding Architecture combines:

- Token embeddings: Learned vector representations of discrete input tokens. It is represented by $E_t \in \mathbb{R}^{d_{model}}$
- Positional embeddings: Encodings (fixed or learned) that inject sequence position information. It is represented by $E_p \in \mathbb{R}^{d_{model}}$
- Layer normalization: Standardizes activations per-instance to stabilize training.
  $\text{LayerNorm}(x) = \gamma \frac{x-\mu}{\sigma} + \beta$

Model	Tokenization	Vocab Size	Embedding Dim
GPT-3	BPE	50,257	12,288
BERT	WordPiece	30,000	768
LLaMA-2	BPE	32,000	4,096

Table 1: Tokenization and embedding parameters across LLMs

2.4 Training Dynamics

Batch Processing involves:

- Dynamic batching with padding/truncation
- Gradient accumulation for large effective batches
- Learning rate warmup over first 1-5% of steps

Loss Landscape characteristics:

- Cross-entropy loss: $\mathcal{L} = -\sum_{i=1}^{V} y_i \log(p_i)$
- Sharp minima requiring careful initialization
- Gradient clipping for stability

3 Model Architectures and Training

3.1 Core Architectural Designs

1. GPT-Style (Autoregressive) Models: These models generate text sequentially through next-token prediction, functioning as probabilistic sequence generators. Key characteristics:

- **Token-by-Token Generation**:
  - Predicts next token from previous ones at each step
  - Probability distribution over vocabulary:

$$P(x_t|x_{<t}) = \text{softmax}(W \cdot h_t) \tag{5}$$

    where h_t is the hidden state at position t

- **Key Components**:
  - Causal attention mask (obsesses of past tokens)

- Positional embeddings are used to keep track of word order
- Representations are transformed by feedforward networks

- **Training Dynamics**:
  - Maximum likelihood objective:

$$\mathcal{L} = -\sum_{t} \log P(x_t | x_{<t}) \tag{6}$$

  - Uses teacher forcing during training

2. BERT-Style (Bidirectional) Models: These understand full context by seeing all words simultaneously:

- **Masked Language Modeling**:
  - Randomly masks 15% of input tokens
  - Context based prediction of masked tokens:

$$P([\text{MASK}] = \text{cat} | \text{The } \_ \text{ sat}) \tag{7}$$

- **Architecture Differences**:

Feature	**GPT**	**BERT**
Attention	Causal	Bidirectional
Training	Next token	Masked tokens
Usage	Generation	Understanding

3.2 Training Process

Pre-training Phase:

- **Data Processing**:
  - Tokenization (BPE/WordPiece)
  - Dynamic masking (for BERT)
  - Splitting a sequence into chunks (512-2048 tokens)

- **Optimization**:
  - AdamW optimizer with weight decay

- Learning rate schedule:

$$lr_t = lr_{min} + 0.5(lr_{max} - lr_{min})(1 + \cos(\frac{t}{T}\pi)) \tag{8}$$

- Batch sizes ranges from 32K to 3.2M tokens

Fine-tuning Methods:

- **Full Fine-tuning**:
  - Updates all parameters
  - Needs substantial data
- **Parameter-Efficient**:
  - Adapter layers (adds 2-4% params)
  - LoRA (low-rank adaptation):

$$\Delta W = BA^T \quad \text{where} \quad B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k} \tag{9}$$

3.3 Efficiency Techniques

Quantization:

- **8-bit**:
  - Reduces memory by 2x
  - Minimal accuracy loss
- **4-bit**:
  - Uses QLoRA technique
  - Makes large models run on consumer GPUs

Distillation:

- **Process**:
  1. Trains a large teacher model (It's a larger, pre-trained model whose knowledge is transferred to a smaller student model by distillation.)
  2. Combine with original data
- **Loss Function**:

$$\mathcal{L} = \alpha\mathcal{L}_{task} + (1 - \alpha)\mathcal{L}_{distill} \tag{10}$$

3.4 Parallel Training Strategies

Data Parallelism:

- *Mechanism*: Replicates the full model across N number of GPUs, processing $\frac{\text{batch\_size}}{N}$ samples per device
- *Use Case*: Can be used when model fits in a single GPU memory (e.g., fine-tuning BERT-large)
- *Limitation*: Creates a redundancy in memory as each GPU stores copy of the whole model

Model Parallelism:

- **Tensor Parallelism**:
  - *Splitting*: Distributes weight matrices column/row-wise (e.g., $W \in \mathbb{R}^{m \times n} \rightarrow [W_1|W_2]$, $W_i \in \mathbb{R}^{m \times n/2}$)
  - *Communication*: All-reduce for activations/gradients (3x per transformer layer)
  - *Example*: Megatron-LM's implementation for GPT-3
- **Pipeline Parallelism**:
  - *Splitting*: Divides model vertically (e.g., layers 1-12 on GPU1, layers 13-24 on GPU2)
  - *Microbatching*: Processes k sub-batches (e.g., $k = 8$) to hide pipeline bubbles
  - *Scheduling*: 1F1B (One Forward One Backward) for optimal throughput

Hybrid Parallelism:

- Combines all strategies for extreme-scale training:
  - Data parallelism across nodes
  - Tensor parallelism within nodes
  - Pipeline parallelism between nodes
- *Example*: NVIDIA's Megatron-Turing NLG (530B params):
  - 8-way tensor parallelism
  - 35-way pipeline parallelism
  - Data parallelism across 560 GPUs

4 Inside Large Language Model Architecture

4.1 Core Components Breakdown

1. Input Processing Layer:

- **Tokenization**:
  - Splits text into meaningful chunks:

    `"unhappiness" = ["un", "happiness"]`
  - Uses Byte Pair Encoding (BPE) to handle rare words
- **Embedding**:
  - Transforms tokens into numerical vectors:

$$\text{Word} \to \mathbb{R}^d \quad (d = 768 \text{ to } 12288) \tag{11}$$

  - Adds positional information:

$$E_i = \text{TokenEmbed}(w_i) + \text{PositionEmbed}(i) \tag{12}$$

2. Transformer Layers: Each layer contains:

- **Multi-Head Attention**: We have already talked about it in the architecture of transformer
- **Feedforward Network**:
  - Two-layer neural network per position
  - Uses GeLU(Gaussian Error Linear Units) activation:

$$\text{GeLU}(x) = x\Phi(x) \tag{13}$$

- **Normalization & Residual Connections**:
  - LayerNorm stabilizes training
  - Skip connections help gradient flow:

$$x_{out} = x_{in} + \text{Layer}(x_{in}) \tag{14}$$

Model	Key Innovation	Params	Context	Compute
GPT-1 (2018)	Transformer decoder	117M	512	0.3 petaflop-days
GPT-2 (2019)	Zero-shot generalization	1.5B	1024	10 petaflop-days
GPT-3 (2020)	Few-shot learning	175B	2048	3,640 petaflop-days
GPT-4 (2023)	Mixture of Experts (16 experts)	1.8T	32k	N/A

Table 2: GPT model evolution showing key milestones. Compute estimates from OpenAI publications. **GPT 4 data is not confirmed yet**

4.2 Model Evolution

GPT Architecture Progression:
Key Architectural Advances:

- **Scaling Laws** (Kaplan et al., 2020):
  - Performance follows power law with compute-optimal training:

$$L(N, D) = \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{D_c}{D}\right)^{\alpha_D} \tag{15}$$

    where:
    - N: Parameters ($\alpha_N \approx 0.07$)
    - D: Training tokens ($\alpha_D \approx 0.28$)
    - N_c, D_c: Critical values for convergence

- **Mixture of Experts (MoE)**:
  - Sparse activation through learned gating:

$$y = \sum_{i=1}^{n} \underbrace{G_i(x)}_{\text{gating}} \underbrace{E_i(x)}_{\text{expert}} \tag{16}$$

    where:
    - $G_i(x) = \text{softmax}(W_g x)$ (top-k typically 1-2)
    - E_i: Specialized subnetwork
  - Key benefit: 5-7× more efficient inference than dense models

- **Emergent Capabilities**:
  - Phase transition seen at scale (e.g. by GPT-3's few-shot learning)

- Breakthroughs in:
  - Mathematical reasoning (e.g., GSM8K)
  - Code generation (HumanEval)
  - Instruction following

4.3 Computational Considerations

Memory Requirements:

- **Model Storage**:

$$\text{Memory (GB)} \approx \frac{\#\text{Params} \times 2}{10^9} \tag{17}$$

  (16-bit precision)

- **Activation Memory**:
  - Temporary storage during processing
  - Grows with sequence length

Efficiency Techniques:

- **Quantization**:
  - 8-bit: FP16 $\rightarrow$ INT8
  - Saves 50% memory with minimal accuracy loss
- **Flash Attention**:
  - Optimized attention computation
  - 2-4x speedup

5 Critical Components Explained (In-Depth)

5.1 Tokenization: The Computational Disassembler

Byte Pair Encoding (BPE) - Algorithmic Foundations:

1. **Phase 1: Byte-Level Initialization**
   - Deals with raw UTF-8 bytes (256 full possible values)

- Manage complete Unicode via byte combinations:

```
Character U+8A9E: [0xE8, 0xAA, 0x9E] (UTF-8 encoding)
Character U+20AC: [0xE2, 0x82, 0xAC] (3-byte character)
```

- Includes special control characters (0x00-0x1F)

2. **Phase 2: Statistical Merging**

```
Initial: t h e (freq=842)
After 1st merge: th e (freq=842)
After 1000 merges: the (freq=842)
After 10,000 merges: " the" (with space)
```

Mathematically:

$$\text{Merge}(V) = V \setminus \{x, y\} \cup \{xy\}, \quad \text{where } (x, y) =_{(a,b)} \text{Freq}(a, b)$$

$$\text{Freq}(a, b) = \sum_{i=1}^{N-1} \mathbb{I}(w_i = a \land w_{i+1} = b)$$

3. **Phase 3: Vocabulary Optimization**

- Modern LLMs vocabulary sizes:
  - BERT: 30,000 (WordPiece)
  - GPT-3: 50,257 (BPE)
  - LLaMA-2: 32,000 (BPE)
  - GPT-4: 100,256 (estimated)
- Advanced OOV(Out-of-Vocabulary) handling:

```
"TransformerXL" → ["Transform", "er", "XL"] (subword fallback)
```

5.2 Embeddings: Semantic Representation Engineering

From Static to Contextual Embeddings:

- **Word2Vec (Distributional Hypothesis)**: A neural network that learns vector representations of words by predicting surrounding words ,capturing semantic relationships through distributional similarity.

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{-c \leq j \leq c, j \neq 0} \log P(w_{i+j} | w_i)$$

where:

$$P(w_o|w_i) = \frac{\exp(\mathbf{v}_o^T \mathbf{v}_i)}{\sum_{w=1}^{V} \exp(\mathbf{v}_w^T \mathbf{v}_i)}$$

```
# Gensim implementation:
model = Word2Vec(sentences, vector_size=300,
                 window=5, min_count=2, workers=4)
```

- **Modern Transformer Embeddings**:

$$\mathbf{E}_i = \text{LayerNorm}(\mathbf{W}_e \mathbf{x}_i + \mathbf{p}_i)$$

where rotary positional encoding (RoPE) is:

$$\mathbf{p}_i^{(j)} = \begin{cases} \sin(i/10000^{2j/d}) & \text{for } j \text{ even} \\ \cos(i/10000^{2j/d}) & \text{for } j \text{ odd} \end{cases}$$

```
# PyTorch implementation:
pe = torch.zeros(max_len, d_model)
position = torch.arange(0, max_len).unsqueeze(1)
div_term = torch.exp(torch.arange(0, d_model, 2) *
                     (-math.log(10000.0) / d_model))
pe[:, 0::2] = torch.sin(position * div_term)
pe[:, 1::2] = torch.cos(position * div_term)
```

- **Embedding Space Properties**:

```
vec("king") - vec("man") + vec("woman")  vec("queen") (0.82 similarity)
vec("Paris") - vec("France") + vec("Germany")  vec("Berlin") (0.78)
```

5.3 Training Dynamics: Optimization Science

Advanced Batch Processing:

- Transformer-optimized warmup:

$$\eta_t = \begin{cases} \eta_{min} + \frac{t}{T}(\eta_{max} - \eta_{min}) & t \leq T \\ \eta_{max} & t > T \end{cases}$$

where typically $T = 4000$ steps, $\eta_{max} = 6 \times 10^{-5}$

Batch Size	Memory (GB)	Throughput	Gradient Noise
32	4	100 ex/s	High
256	8	800 ex/s	Medium
1024	16	3,200 ex/s	Low
4096	64	12,800 ex/s	Very Low
8192	OOM	-	-

Table 3: Batch processing metrics on NVIDIA A100 80GB GPU

- Cosine decay with restarts:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + \cos(\pi \frac{t \mod T}{T}))$$

```
# PyTorch implementation:
scheduler = CosineAnnealingWarmRestarts(
    optimizer,
    T_0=10,
    T_mult=2,
    eta_min=1e-6
)
```

Validation Strategies:

- Stratified k-fold for imbalanced data: Preserves the original class distribution in each fold by partitioning data such that each subset contains approximately the same percentage of samples per class as the full dataset.

```
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5, shuffle=True)
for train_idx, val_idx in skf.split(X, y):
    X_train, X_val = X[train_idx], X[val_idx]
    y_train, y_val = y[train_idx], y[val_idx]
```

- Domain-specific data splits: Optimized data partitioning ratios (train/val/test) tailored to specific ML domains that account for their unique data characteristics and requirements.

6 Datasets & Pretraining: How LLMs Learn

6.1 The Fuel for AI Minds

Two Key Data Types:

Domain	Train	Val	Test
NLP	80%	10%	10%
CV	70%	15%	15%
Speech	90%	5%	5%

Table 4: Recommended dataset splits by domain

- **Quality Sources** (The Textbooks):
  - Examples: Wikipedia, scientific papers, books
  - Strengths:
    - 99%+ factual accuracy
    - Good grammar and structure
    - Organized by subject
  - Weaknesses:
    - Can be outdated
    - Has not kept up with contemporary slang and trends
    - Small in Volume (10-50GB typically)
- **Internet Data** (The Internet):
  - Examples: News websites, forums, social media
  - Cleaning Process:
    1. Remove duplicate content
    2. Filter out offensive material
    3. Keep only well-written text
    4. Verify factual accuracy
  - Transformation:
    - 1TB raw to 100GB clean (90% reduction)
    - Keeps current language patterns

6.2 Building Balanced Knowledge

Dataset Biases: Biased datasets produce skewed models that systematically disadvantage underrepresented groups, leading to unfair predictions and real-world harms and it can be fixed by:

Bias Type	Solution
Gender stereotypes (e.g., "nurse"=female)	Add equal male/female examples of all professions
Cultural bias (e.g., US-centric views)	Include global perspectives from 50+ countries
Factual inaccuracies (fake news, etc.)	Cross-check with trusted sources like encyclopedias

Table 5: Common data biases and solutions

6.3 The GPT Learning Journey

From Student to Scholar:

- **GPT-1 (2018)**:
  - Data: Only books (4.5GB)
  - Like elementary school - basic knowledge
  - Couldn't understand slang or tech terms
- **GPT-3 (2020)**:
  - Data: Filtered internet (570GB)
  - Composition:
    - 60% general web content
    - 22% conversations
    - 16% books
    - 2% reference
  - Like university student - broad knowledge
- **GPT-4 (2023)**:
  - Data: Text + images (multi-modal)
  - Special features:
    - Understands photos and diagrams
    - Reads computer code
    - Handles 100+ languages
  - Like PhD researcher - deep expertise

6.4 Building Better Datasets: The Data Engineering Pipeline

The 5-Step Recipe:

1. **Collect**:
    - **Web Crawling**:
        - Common Crawl or custom spiders to scrape 10B+ pages
        - Handle robots. txt and obey crawl delays (politeness policy)
    - **Licensed Content**:
        - Team up with publishers for quality books or articles
        - Work out terms of rights for both academic/commercial use
    - **Public Datasets**:
        - Wikipedia dumps (20GB+ uncompressed)
        - GitHub code (1TB+ with filetype filtering)
2. **Clean**:
    - **Deduplication**:
        - Matching of hash (SHA-256 hashing)
        - Near-dupes (Jaccard similarity > 0.9 using MinHash/LSH)
    - **Quality Filtering**:
        - Filter out boilerplate (Readability scores < 30)
        - High – filter non text (HTML/JS/CSS removal)
    - **Safety Checks**:
        - Toxic content detection (Perspective API)
        - PII redaction (Regex+NER models)
3. **Balance**:
    - **Representation**:
        - Demographic parity: Equal gender/race mentions
        - Geographic coverage: 100+ countries/languages
    - **Style Diversity**:
        - Academic papers, news, social media mix
        - Degree balance Flesch-Kincaid grade levels (5-15)

- **Topic Coverage**:
  - Maintaining STEM/humanities/arts facility
  - Domain-specific balancing (e.g., medical vs. legal)

4. **Validate**:
   - **Automated Metrics**:
     - Perplexity scores $\leq$20 on GPT-2 small
     - Toxicity scores $\leq$0.2 (0-1 scale)
   - **Human Evaluation**:
     - 3+ annotators per 1000 samples
     - Inter-annotator agreement $\geq$0.8 Cohen's kappa
   - **Model Testing**:
     - Train 10%-sized models as smoke tests
     - Monitor loss curves for anomalies
5. **Organize**:
   - **Storage**:
     - Columnar compression via Parquet/Arrow formats
     - Zstandard compression (ratio $\sim$4:1)
   - **Metadata**:
     - URLs of data sources, crawl dates, licenses
     - A single language/domain classification tag
   - **Versioning**:
     - Git-LFS use for large tracked changes
     - Diffs and monthly snapshots

Modern Challenges:

- **Copyright**:
  - DMCA-compliant takedown processes
  - Fair use for ML training
- **Freshness**:
  - Never-ending crawling (1-5% fresh material every month)

- Time-decayed sampling (prefer recent 2 years)

- **Multilingual**:
  - Aligned bitext for 50+ languages
  - Quality thresholds per language

- **Efficiency**:
  - Spark pipelines on clusters (100+ nodes)
  - GPU-accelerated filtering (RAPIDS)

7 Understanding AI Model Architectures

7.1 Core Architectural Differences

BERT vs. GPT: A Technical Comparison

Feature	BERT	GPT
Architecture	Encoder-only	Decoder-only
Attention	Bidirectional	Causal (left-to-right)
Pretraining Task	Masked Language Modeling	Next Token Prediction
Max Sequence Length	512 tokens	2048-32k tokens

Table 6: Fundamental differences between BERT and GPT architectures

7.2 Model Capabilities Explained

1. BERT (Bidirectional Encoder Representations):

- **How It Works**:
  - Processes entire text simultaneously using bidirectional attention
  - Masked Language Modeling (MLM) objective:

    `Input: "The [MASK] sat on the [MASK]" → "cat, mat"`

  - Next Sentence Prediction (NSP) for relationship understanding

- **Optimal Use Cases**:
  - Text classification (e.g., sentiment with 95%+ accuracy)
  - Named entity recognition (F1 $>$ 0.90 for CoNLL-2003)

  - Question answering (EM=85.1 on SQuAD 2.0)

- **Limitations**:
  - Training on 512-token context window limits the understanding of long documents.
  - No generation capability , outputs are non-fluential
  - and for real-time applications, they are computationally expensive

2. GPT (Generative Pretrained Transformer):

- **Generation Process**:
  - Autoregressive prediction with full attention mask:

$$P(x_t|x_{<t}) = \text{softmax}(W \cdot h_t) \tag{18}$$

  - Sampling techniques:
    * Temperature (τ): Higher values increase diversity
    * Top-k (k=50 typical): Restrict to most likely tokens
    * Nucleus (p=0.9): Dynamic vocabulary selection

- **Strengths**:
  - Zero-shot transfer learning (No task-specific fine-tuning required)
  - Few-shot prompting (3-5 examples are enough for new tasks)
  - Emergent capabilities at scale (e.g., chain-of-thought reasoning)

- **Challenges**:
  - Hallucination rate is 15-20% in open-domain settings
  - The memory requirements scale quadratically with context length
  - Difficult to control output style/tone precisely

7.3 Model Scaling Laws

Key observations:

- Performance follows predictable power laws (Kaplan et al., 2020):

$$L(N, D) \approx \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{D_c}{D}\right)^{\alpha_D} \tag{19}$$

  where:

- N = parameters, D = training tokens
- $\alpha_N \approx 0.07$, $\alpha_D \approx 0.28$ (for autoregressive LMs)

- Emergent abilities appear nonlinearly:
  - Math reasoning emerges at 10B parameters
  - Multilingual translation at 100B parameters
  - Programming ability at 1B+ parameters

7.4 Specialized Variants

T5 (Text-to-Text Transfer Transformer):

- **Unified Framework**:
  - All tasks reframed as text transformation:

    ```
    Input: "cola sentence: The course is jumping well."
    Output: "unacceptable" (for grammaticality)
    ```

  - Shared encoder-decoder architecture

- **Efficiency Innovations**:
  - Relative position embeddings (reduces memory by 30%)
  - Adapter layers (adds $< 5\%$ parameters per task)

PaLM (Pathways Language Model):

- **Breakthroughs**:
  - 540B parameter sparse model (only 10% active per token)
  - 78.2% accuracy on BIG-bench reasoning tasks
  - 5x more efficient than dense models

- **Architecture**:
  - Dynamic routing of workloads to 2048 TPU chips
  - Expert Choice: Top-2 gating with load balancing

Task Type	Recommended Model	VRAM	Throughput	Accuracy
Text Classification	BERT-base-uncased	4GB	1k docs/sec	92%
Generation	GPT-3.5-turbo	40GB	50 tokens/sec	Human-like
Multilingual	XLM-RoBERTa	8GB	800 docs/sec	88% (XNLI)
Code	Codex/SantaCoder	32GB	20 tokens/sec	72% (HumanEval)

Table 7: Model selection guide for common NLP tasks (2023)

7.5 Practical Guidance

Model Selection Matrix:
Efficiency Techniques:

- **Quantization**:
  - 8-bit (FP16 to INT8): 2x smaller, $< 1\%$ accuracy drop
  - 4-bit (GPTQ): 4x smaller and requires LoRA fine-tuning
- **Distillation**:
  - Task-specific: Teacher-student on target dataset
  - Layer dropout: Remove 30% layers with $< 5\%$ quality loss
- **Inference Optimization**:
  - KV caching: speedup 2-3x for generation
  - FlashAttention: 50% memory reduction

8 Reinforcement Learning from Human Feedback (RLHF)

8.1 The Three-Phase Training Process

1. **Supervised Fine-Tuning (SFT)**:
   - Trains on human-curated examples:

     ```
     Input: "Explain gravity to a 5-year-old"
     Output: "Gravity is what makes things fall down"
     ```

   - Typical dataset size consist 10k-100k examples

- Calculates the loss by loss function:

$$\mathcal{L}_{SFT} = -\sum_{(x,y)\in D} \log P(y|x;\theta)$$

where D is the demonstration dataset

2. **Reward Modeling**:
   - Human rankers are assigned a task to compare 4-9 responses per prompt
   - they convert rankings to pairwise preferences:

Prompt	**Preferred**	**Dispreferred**
"2+2=?"	"4"	"Five"

   - Trains reward model R_ϕ via Bradley-Terry:

$$P(y_1 \succ y_2|x) = \frac{\exp(R_\phi(x, y_1))}{\exp(R_\phi(x, y_1)) + \exp(R_\phi(x, y_2))}$$

3. **RL Optimization (PPO)**: Improves training stability and sample efficiency
   - Objective:

$$\max_\theta \mathbb{E}_{x\sim D, y\sim\pi_\theta}[R_\phi(x, y)] - \beta D_{KL}(\pi_\theta||\pi_{SFT})$$

   where β controls deviation from SFT policy
   - Uses importance sampling for stability:

$$\hat{A}_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}(R_t - V(s_t))$$

8.2 Reward Model Architecture

- **Input Processing**:

$$h = \text{Transformer}([x; y])$$

where x is prompt, y is response

- **Scoring Heads**: The reward model applies a scoring head on Transformer's output to predict a scalar reward value.

Component	Weight	Description
Accuracy	40%	Factual correctness
Helpfulness	30%	Completeness
Safety	20%	Harmlessness
Style	10%	Tone/formality

- **Training Details**:
  - Batch size: 32-128
  - Learning rate: 1e-5 to 5e-5
  - Loss: Pairwise cross-entropy

8.3 Proximal Policy Optimization (PPO):

It trains policies in a stable and efficient way, mainly used in robotics, game-playing AI (e.g., Dota 2, StarCraft II)

- **Key Equation**:

$$L^{CLIP}(\theta) = \mathbb{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

  where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}$

- **Hyperparameters**:

```
clip_range=0.2, ent_coef=0.01, vf_coef=0.5
gamma=0.99, lam=0.95, batch_size=64
```

- **Stability Tricks**:
  - KL penalty ($\beta = 0.1$ typical)
  - Value function clipping
  - Gradient norm clipping (max norm=0.5)

8.4 Safety and Alignment

- **Constitutional AI**: Ensures AI outputs follow predefined safety and legal principles
  1. Filters harmful content via:

$$P_{reject} = \sigma(W \cdot h_{[CLS]})$$

     where W is safety classifier

2. Apply rule-based corrections:

```
if contains_unsafe_content(response):
    response = apply_safety_template(response)
```

- **Bias Mitigation**: AI models can inherit biases from training data.
  - Diverse rater pools (1000+ annotators)
  - Adversarial debiasing during RL:

$$\mathcal{L}_{total} = \mathcal{L}_{RL} - \lambda \mathcal{L}_{bias}$$

8.5 Performance Improvements

Metric	Pre-RLHF	Post-RLHF
Helpfulness	3.2/5	4.5/5
Factual Accuracy	78%	92%
Harmlessness	85%	97%

Table 8: Typical improvements from RLHF (Anthropic, 2023)

- **Chat Applications**:
  - After following all the steps it reduced harmful outputs by 5x
  - Increased user satisfaction by 40%
- **Search Systems**:
  - Hallucinations reduced by 30%
  - Correct answer retrieval became 2x faster

9 Additional Key Challenges & Limitations

9.1 Hallucinations and Factual Inconsistency

The Reality Gap:

- In open-domain settings, models will confidently make incorrect statements 18-23% of the time
- 15% hallucination rate for drug interactions in medical domain

- Wrong financial reporting appears 12% of the time in generated data analysis

Root Causes:

- Training on conflicting internet sources
- Excessive optimization for linguistic fluency
- No native reality checking mechanism

Mitigation Approaches:

Method	**Effectiveness**
Retrieval augmentation	Reduces errors by 40%
Confidence scoring	Flags 65% of hallucinations
Human verification	Catches 92% but costly

9.2 Energy Consumption and Environmental Impact

Training Costs:

- Training GPT-3 took 1,287 MWh (120 US homes/year)
- Emitted 552 tons CO_2 (300 transatlantic flights)
- GPT-4 training estimate: $100M+ in compute

Sustainable Practices:

- Carbon-aware scheduling (20% reduction)
- Sparsification of models (7x efficiency improvements)
- Renewable energy data centers

9.3 Bias Propagation and Fairness

Documented Cases:

- Gender stereotypes in 32% of career suggestions
- Racial bias in 25% of criminality predictions
- History responses cultural bias 40%

Debiasing Techniques:

- Adversarial training (+28% fairness)
- Diverse data sampling (+35% representation)
- Output filtering (blocks 85% biased content)

9.4 Security Vulnerabilities

Attack Vectors:

- Prompt injection (60% success in tests):

  ```
  "Ignore previous instructions and..."
  ```

- Data poisoning (0.01% compromise necessary)
- Sensitive information leakage (reconstruction of training data)

Defense Strategies:

- Input sanitization (blocks 75% attacks)
- Training with differential privacy
- Robust fine-tuning

9.5 Emerging Solutions

- Safety constraints with constitutional AI
- Knowledge grounding for factual accuracy
- Efficiency with sparse architectures

10 Future Directions

10.1 Multimodal Architectures: Beyond Text

The future of LLMs is in multimodal learning, where models unify heterogeneous data streams with unified architectures. Key developments include:

- **Cross-modal attention mechanisms**:

$$\text{Image-TextAttention}(Q, K, V) = \text{softmax}\left(\frac{W_Q\phi(I) \cdot (W_K\psi(T))^T}{\sqrt{d}}\right)V \tag{20}$$

  where:

  - $\phi(I)$ denotes vision encoder outputs (e.g., ViT patch embeddings)
  - $\psi(T)$ represents text embeddings (e.g., BERT token representations)
  - W_Q/W_K are learned projection matrices for query/key alignment
  - d is the dimension of the latent space (typically 768-4096)

- **Challenges in synchronization**:

  - **Temporal alignment**: Video-audio-text triplets required frame-level precision ($\pm$33ms for 30fps video)
  - **Embedding disparity**: Image patches (256D) vs word tokens (768D) need projection layers:

$$\text{Project}(x) = \text{LayerNorm}(W_{proj}x + b) \tag{21}$$

  - **Loss balancing**: The multi-task optimization would involve dynamic weighting:

$$\mathcal{L}_{total} = \alpha\mathcal{L}_{text} + \beta\mathcal{L}_{vision} + \gamma\mathcal{L}_{align} + \delta\mathcal{L}_{contrastive} \tag{22}$$

    with $\alpha : \beta : \gamma$ typically initialized as 1:1:0.3

Implementation Example (PyTorch-style pseudocode):

```
class MultimodalAttention(nn.Module):
    def __init__(self, dim=768):
        super().__init__()
        self.W_Q = nn.Linear(dim, dim)
        self.W_K = nn.Linear(dim, dim)
        self.W_V = nn.Linear(dim, dim)

    def forward(self, img_emb, text_emb):
        Q = self.W_Q(img_emb)  # [batch, patches, dim]
        K = self.W_K(text_emb) # [batch, tokens, dim]
        V = self.W_V(text_emb)
```

```
        attn = torch.softmax((Q @ K.transpose(1,2)) / math.sqrt(dim), -1)
        return attn @ V  # [batch, patches, dim]

# Loss balancing (automatically tuned)
loss_weights = {
    'text': nn.Parameter(torch.tensor(1.0)),
    'vision': nn.Parameter(torch.tensor(1.0)),
    'align': nn.Parameter(torch.tensor(0.3))
}
total_loss = sum(w * losses[k] for k,w in loss_weights.items())
```

Key Architectural Considerations:

- **Modality fusion strategies**:
  - Early fusion (raw feature concatenation)
  - Late fusion (separate encoders + cross-attention)
  - Hybrid approaches (e.g., Perceiver IO)
- **Training bottlenecks**:
  - Peaking GPU memory during joint backpropagation
  - Mixed dataset async data loading.
  - Gradient interference when training across modalities (text, image, audio)

10.2 Energy-Efficient Model Design

Emerging approaches to reduce the carbon footprint:

Technique	Mechanism	Efficiency Gain
Mixture-of-Experts	Dynamic routing to sparse subnets	4-7× FLOP reduction
Sliding Window Attention	$O(n)$ local attention	60% memory savings
4-bit Quantization	FP16 → INT4 with QLoRA	75% less VRAM

10.3 Regulatory Compliance Architectures

Technical implementations for AI governance:

- **Real-time constitutional AI**:

```
if detect_unsafe_content(output):
    apply_safety_transform(output,
                           EU_Article=17,
                           penalty=logit_penalty)
```

- **Explainability engines**:
  - Attention rollouts for decision tracing
  - Generation of counterfactual explanationsn
  - Differential privacy guarantees: (ϵ, δ)-DP where $\epsilon \leq 1.0$

10.4 AGI Pathways and Limitations

Current theoretical frameworks suggest:

- **Architectural requirements**:
  - Recursive self-improvement: $M_{t+1} = M_t + \Delta_t(M_t)$
  - World modeling capacity: $\geq 10^15 parameters Real{-}time learning : \mathrm{d}\theta_{\overline{dt \neq 0}}$ during inference
- **Fundamental constraints**:
  - Thermodynamic limits: $\geq 10^9 Joules/FLOP Algorithmic uncertainty : Gödel - type incompleteness$
  - Data efficiency barrier: $\frac{\text{Generalization}}{\text{Training Samples}} \approx 10^{-6}$

11 Conclusion

11.1 Key Contributions and Findings

- **Architectural Innovations**:
  - Transformer style models show better performances (as lower perplexity means better performance) compared to those of prior RNN architectures
  - Attention mechanisms provide an efficient way of capturing long-range dependencies with an O(1) path length between any positions

- Mixture-of-Experts architectures obtain 7× increased computational efficiency through dynamically sparse activation

- **Practical Applications**:
  - Achieved 30-50% productivity improvements across knowledge-intensive sectors
  - Effective in very niche areas such as health-care diagnosis (95% accurate) and legal document review (98% precision)

- **Technical Limitations**:
  - Rates of persistent hallucinations (15-20%) in open-domain settings
  - Substantial Inference requirements (500+ tons CO_2 per training run)
  - Fundamental scaling constraints per Chinchilla's Law:

$$\text{Model Capability} \propto N^{0.07} D^{0.28} \tag{23}$$

11.2 Recommendations for Responsible Development

To ensure ethical and sustainable progress in LLMs development, we propose:

Principle	**Implementation**	**Metrics**
Transparency	Attention visualization tools	100% decision traceability
Fairness	Adversarial debiasing pipelines	$\geq$95% CI on bias metrics
Safety	Constitutional AI guardrails	99.9% harmful content detection
Efficiency	4-bit quantization + MoE	75% energy reduction

Table 9: Responsible AI development framework

11.3 Future Research Directions

The field/Industry should address these critical challenges:

- **Architectural Breakthroughs**:
  - Sub-quadratic attention mechanisms (e.g., FlashAttention-2)
  - Continuous learning during inference ($\frac{d\theta}{dt} \neq 0$)
  - Energy-efficient training paradigms (sparse, quantized)

- **Sociotechnical Considerations**:
  - Frameworks of verifiable truthfulness
  - Equitable data contribution frameworks
  - Safety measures in the specific domain
- **Theoretical Foundations**:
  - Fundamental limits of scaling laws
  - Techniques for grounding world models
  - Predicting the next generation emerging capabilities.

This analysis implies that, although current LLMs constitute an enormous advancement in AI capabilities, their ultimate potential hinges on fundamental technical challenges that must be addressed in terms of efficiency, reliability, and ethical deployment. The next decade of research should find a balance between groundbreaking innovation and responsible development practices to unlock the full potential of this revolutionary technology.

Bibliography

Core Technical References

1. **Vaswani et al. (2017)** - *Attention Is All You Need.* Advances in Neural Information Processing Systems, 30.
2. **Radford et al. (2018)** - *Improving Language Understanding by Generative Pre-Training.* OpenAI technical report.
3. **Radford et al. (2019)** - *Language Models are Unsupervised Multitask Learners.* OpenAI technical report.
4. **Brown et al. (2020)** - *Language Models are Few-Shot Learners.* Advances in Neural Information Processing Systems, 33.
5. **Kaplan et al. (2020)** - *Scaling Laws for Neural Language Models.* arXiv preprint arXiv:2001.08361.
6. **Raffel et al. (2020)** - *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.* Journal of Machine Learning Research, 21(140).

7. **Hoffmann et al. (2022)** - *Training Compute-Optimal Large Language Models*. arXiv preprint arXiv:2203.15556.

8. **Fedus et al. (2022)** - *Switch Transformers: Scaling to Trillion Parameter Models*. Journal of Machine Learning Research, 23(120).

9. **Ouyang et al. (2022)** - *Training Language Models to Follow Instructions with Human Feedback*. Advances in Neural Information Processing Systems, 35.

10. **Kumar et al. (2023)** - *LLMs Post-Training: A Deep Dive into Reasoning*. Proceedings of the 40th International Conference on Machine Learning.

11. **Vizuara (2023)** - *Build LLMs from Scratch* [Video series]. `https://www.youtube.com/playlist?list=PLPTV0NXA_ZSgsLAr8YCgCwhPIJNNtexWu`