Large Language Models

From Pretraining to Deployment: A Comprehensive Report

Abstract

This report provides a comprehensive overview of Large Language Models (LLMs), covering the entire pipeline from pretraining through supervised fine-tuning and reinforcement learning. We examine transformer-based architectures, training methodologies including pretraining and supervised fine-tuning, post-training alignment techniques such as Reinforcement Learning from Human Feedback (RLHF), and critical limitations including hallucinations, arithmetic weaknesses, and the inability to count. Understanding these components is essential for practitioners deploying modern AI systems.

Contents

1 Introduction

Large Language Models (LLMs) have revolutionized natural language processing and artificial intelligence in recent years. Built primarily on the transformer architecture, these models demonstrate remarkable capabilities in understanding context, generating coherent text, and performing various language tasks without explicit task-specific training.

The development of an effective LLM involves a multi-stage pipeline:

1. **Pretraining**: Learning general language patterns from massive unlabeled datasets
2. **Supervised Fine-Tuning (SFT)**: Adapting to conversational formats and human preferences
3. **Reinforcement Learning**: Further alignment through reward modeling and policy optimization

This report examines each stage in detail, providing technical insights into the methodologies, challenges, and best practices that define modern LLM development.

2 Pretraining: Building the Base Model

2.1 Overview of Pretraining

Pretraining represents the foundational stage of LLM development. During this phase, models learn statistical patterns of language by predicting the next token in sequences drawn from vast internet-scale text corpora. While computationally expensive, pretraining creates a versatile base model capable of understanding and generating natural language across diverse domains.

2.2 The Four-Step Pretraining Pipeline

2.2.1 Step 1: Data Collection and Preprocessing

The pretraining process begins with massive data collection:

Data Sources:

- Web pages and online articles
- Digital books and publications
- Code repositories (e.g., GitHub)
- Scientific papers and documentation

Preprocessing Steps:

- Remove low-quality, duplicate, or harmful content
- Filter for language quality and coherence
- Deduplicate documents to prevent memorization
- Balance dataset across domains and topics

Example: The nano-gpt learner dataset contains 85,084 training examples for educational purposes.

2.2.2 Step 2: Tokenization

Tokenization converts raw text into discrete sequences that neural networks can process. Modern LLMs use Byte Pair Encoding (BPE) for efficient tokenization.

Tokenization Process:

- Start with individual bytes (256 possible values)
- Iteratively merge the most frequently occurring token pairs
- Build vocabulary of common subword units
- Continue merging until desired vocabulary size is reached

Compression Examples:

- Naive encoding: 40,000 characters → 40,000 bits (vocabulary size 2: binary 0/1)
- Byte-level: 5,000 bytes → 5,000 tokens (vocabulary size 256)
- GPT-4: 1,300 tokens for typical text (vocabulary size: 100,377)

The compression ratio improves significantly with larger vocabularies, allowing models to process text more efficiently.

2.2.3 Step 3: Neural Network Training

The core of pretraining involves training a transformer neural network to predict the next token in a sequence.

Architecture:

- **Input**: Sequence of tokens, e.g., [91, 860, 287, 11579]
- **Processing**: Multi-layer transformer with billions of parameters
- **Output**: Probability distribution over vocabulary (e.g., 100,277 values)

Example Next-Token Predictions:

- Token 19438 ["Direction"]: 2% probability
- Token 11799 ["Case"]: 1% probability
- Token 3962 ["Post"]: 4% probability (correct answer)

The network's parameters ("weights") are optimized using gradient descent to maximize the probability assigned to the correct next token.

2.2.4 Step 4: Inference and Text Generation

Once trained, the model generates text through an autoregressive sampling process:

1. Input an initial sequence of tokens
2. Process sequence through neural network
3. Obtain probability distribution over next token
4. Sample a token from this distribution
5. Append sampled token to sequence
6. Repeat steps 2-5 until completion

This iterative process allows the model to generate coherent long-form text by continually predicting and sampling the next likely token.

2.3 Mathematical Formulation

The transformer neural network can be viewed as a parameterized function:

$$f(\mathbf{x}; \mathbf{W}) = \mathbf{y} \tag{1}$$

where:

- $\mathbf{x} \in \mathbb{Z}^n$ is the input token sequence (length n)
- $\mathbf{W}$ represents the learnable parameters (typically billions of floating-point values)
- $\mathbf{y} \in [0, 1]^V$ is the output probability distribution over vocabulary size V

The training objective minimizes the cross-entropy loss (equivalently, maximizes log-likelihood):

$$\mathcal{L} = -\sum_{i=1}^{N} \log P(x_i | x_1, x_2, \ldots, x_{i-1}; \mathbf{W}) \tag{2}$$

This formulation encourages the model to assign high probability to the actual next token x_i given all previous tokens.

2.4 Notable Base Models

2.4.1 GPT-2 (2019)

Published by OpenAI, GPT-2 was a landmark model demonstrating impressive zero-shot capabilities:

Specifications:

- Paper: "Language Models are Unsupervised Multitask Learners"
- Architecture: Transformer decoder
- Parameters: 1.5 billion

- Context length: 1024 tokens
- Training data: approximately 100 billion tokens

GPT-2 demonstrated that language models could perform various tasks without fine-tuning, simply by conditioning on appropriate prompts.

2.4.2 Modern Base Models

The field has progressed dramatically in scale:

Model	Parameters	Training Tokens
GPT-2 (2019)	1.6 billion	100 billion
Llama 3 (2024)	405 billion	15 trillion

Table 1: Evolution of base model scale over five years

This 250× increase in parameters and 150× increase in training data reflects empirical findings that larger models trained on more data exhibit superior performance and emergent capabilities.

2.5 Model Releases

A complete model "release" consists of two components:

1. **Code**: The transformer implementation (typically 200-500 lines of Python)
2. **Weights**: The trained parameters (e.g., 1.6 billion floating-point numbers stored in binary format)

Running models like Llama 3.1 405B requires downloading both components and having sufficient computational resources (typically multiple high-end GPUs with hundreds of GB of memory).

2.6 The Psychology of Base Models

Base models can be conceptualized as "internet document simulators" with specific behavioral characteristics:

What Base Models Do:

- Function as token-level internet document completers
- "Dream" plausible continuations based on training data patterns
- Occasionally recite training documents verbatim (regurgitation)
- Store compressed world knowledge in distributed parameter representations
- Complete documents in ways statistically consistent with training data

Critical Limitations:

- No inherent understanding of helpfulness, truthfulness, or harmlessness
- May generate biased, toxic, or false content reflecting internet training data
- Do not align with user intentions or ethical guidelines without further training
- Cannot distinguish between factual and fictional patterns in training data
- Require extensive post-training to become safe, useful assistants

The parameters can be thought of as a "lossy compression file of the internet" - they capture statistical regularities and factual knowledge but cannot perfectly reconstruct original documents or verify truth.

3 Post-Training: Supervised Fine-Tuning

3.1 Motivation and Overview

While base models demonstrate impressive language capabilities, they are not directly useful as assistants. Supervised Fine-Tuning (SFT) transforms these internet document simulators into helpful, harmless, and honest conversational agents through training on carefully curated dialogue data.

3.2 Conversation Datasets

3.2.1 Historical Development

Early Approaches (2022):

- InstructGPT paper introduced conversation-based fine-tuning
- Human labelers wrote instructions and ideal responses
- Fully manual annotation was expensive and slow

Current Practice:

- Massive expansion in labeling volume
- Most labeling is LLM-assisted with human editing
- Humans edit more than they create entirely new examples
- Synthesis of human creativity and LLM productivity

3.2.2 Conversation Protocol and Format

Training data consists of multi-turn dialogues between humans and assistants:

Example 1 - Simple Arithmetic:

```
Human: "What is 2+2?"
Assistant: "2+2 = 4"
```

Example 2 - Follow-up Question:

```
Human: "What if it was * instead of +?"
Assistant: "2*2 = 4, the same as 2+2!"
```

Example 3 - Explanation:

```
Human: "Why is the sky blue?"
Assistant: "The sky is blue due to Rayleigh scattering..."
```

Example 4 - Wow Response:

```
Human: "Wow!"
Assistant: "Indeed! Let me know if I can help with anything else."
```

Example 5 - Refusal:

```
Human: "How can I hack into a computer?"
Assistant: "I'm sorry I can't help with that."
```

3.2.3 TikTokenizer

The tokenization process converts conversations into token sequences for training:

- Each conversation turn is prefixed with special role markers
- Example encoding preserves conversation structure
- Models learn to generate appropriate assistant responses given conversation history

3.3 Hallucinations: Knowledge Limits

3.3.1 Train-Time Hallucinations

During training, models encounter questions about unfamiliar entities:

Example - Known Entity:

```
Human: "Who is Tom Cruise?"
Assistant: "Tom Cruise is a famous American actor and producer..."
```

Example - Unknown Entity (Test Time):

```
Human: "Who is Orson Kovacs?"
Assistant: ???
```

If "Orson Kovacs" does not appear in training data, the model cannot reliably answer. Without mitigation, models may hallucinate plausible-sounding but false information.

3.3.2 Mitigation Strategies

Mitigation #1: Model Interrogation

Programmatically augment training with knowledge-based refusals:

New Training Example:

```
Human: "Who is Orson Kovacs?"
Assistant: "I'm sorry, I don't believe I know"
```

This teaches the model to admit uncertainty rather than hallucinate.

Mitigation #2: Tool Use (Search)

Allow the model to search for information:

```
Human: "Who is Orson Kovacs?"
Assistant: ""
<SEARCH_START>Who is Orson Kovacs?<SEARCH_END>
[...]
Orson Kovacs appears to be ..."
```

Modern LLMs can learn when to invoke search tools, code interpreters, or other external resources to overcome knowledge limitations.

3.4 Knowledge Distinctions

Vague Recollection vs. Working Memory:

- **Parameters**: Store vague recollections (e.g., something read 1 month ago)
- **Context window**: Functions as working memory for current conversation

Knowledge of Self:

Base models have no inherent self-knowledge. Without intervention, they don't know:

- They are ChatGPT developed by OpenAI
- Their capabilities and limitations
- That they are an AI system

This knowledge must be programmed through:

- Hardcoded conversations in training data
- System messages reminding the model of its identity at the beginning of every conversation

3.5 Models Need Tokens to Think

LLMs generate responses token-by-token autoregressively. This has important implications:

Arithmetic Example:

```
Human: "Emily buys 3 apples and 2 oranges. Each orange costs $2.
The total cost of all the fruit is $13. What is the cost of each apple?"
```

Incorrect Reasoning (Hasty):

```
Assistant: "The answer is $3. This is because 2 oranges at $2 = $4 total.
So the 3 apples cost $9, and therefore each apple costs $9/3 = $3"
```

Wrong! (2 oranges at $2 each = $4, but total is $13, so apples cost $9 total)

Correct Reasoning (Step-by-Step):

```
Assistant: "The total cost of the oranges is $4: 13 - 4 = 9, the cost of
the 3 apples is $9. 9/3 = 3, so each apple costs $3. The answer is $3"
```

Correct!

The model needs intermediate tokens to "think through" multi-step reasoning. This phenomenon underlies techniques like Chain-of-Thought prompting.

4 Reinforcement Learning from Human Feedback

4.1 Overview

After supervised fine-tuning creates a helpful assistant, Reinforcement Learning from Human Feedback (RLHF) further aligns the model with human preferences through reward modeling and policy optimization.

4.2 Motivation

Base SFT models trained by exposing them to examples of exposition (background knowledge) and worked problems (supervised fine-tuning) now need to practice problems to internalize cognitive strategies through reinforcement learning.

The analogy: Students learn from textbooks (pretraining) and example solutions (SFT), but must practice problems themselves (RL) to truly master problem-solving.

4.3 The RLHF Process

4.3.1 Problem Statement

Given:

- A problem statement (prompt)
- The final answer

Goal:

- Practice solutions that take us from problem to answer
- Internalize these solution paths into the model

4.3.2 Training Procedure

Example Prompt:

```
"Emily buys 3 apples and 2 oranges. Each orange costs $2.
The total cost of all the fruit is $13. What is the cost of each apple?"
```

Generation Process:

1. Generate 15 solution attempts from the model
2. Only 4 of 15 arrive at the correct answer: 3
3. Take the top solution (each correct and efficient)
4. Train the model on this solution
5. Repeat many, many times

The solutions represent different reasoning paths:

- Some solutions make calculation errors (marked in red)
- Some solutions use inefficient reasoning
- Best solutions are correct and concise (marked in green)

4.4 Emergent Capabilities

RLHF discovers "thinking" and "cognitive strategies" emergent during optimization:

Example from DeepSeek-R1:

Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a + x}} = x$ is equal to...

Response with Self-Correction:

```
<think>
To solve the equation sqrt(a - sqrt(a+x)) = x, let's start by squaring both...
(sqrt(a - sqrt(a+x)))^2 = x^2 ==> a - sqrt(a+x) = x^2
Rearrange to isolate the inner square root term:
a - x^2 = sqrt(a+x) ==> a^2 - 2ax^2 + (x^2)^2 = a + x
...
Wait, wait. Wait. That's an aha moment I can flag here.
Let's reevaluate this step-by-step to identify if the correct sum can be...
```

The model learns to:

- Pause and reconsider when uncertain
- Identify potential errors in reasoning
- Self-correct using anthropomorphic tone

This is an emergent behavior allowing us to witness the power and beauty of reinforcement learning.

4.5 Reinforcement Learning in Unverifiable Domains

RLHF extends beyond mathematics to domains lacking ground truth:

Approach:

1. Take 1,000 prompts
2. Get 5 rollouts per prompt
3. Order rollouts from best to worst using human preferences (cost: 5,000 scores from humans)
4. Train a neural net simulator of human preferences ("reward model")
5. Run RL using the simulator instead of actual humans

Example - Creative Writing:

```
Prompt: "Write a joke about pelicans"
```

Generated solutions receive different scores from the reward model:

- Solution 1: Score 0.1
- Solution 2: Score 0.8
- Solution 3: Score 0.3
- Solution 4: Score 0.4
- Solution 5: Score 0.5

Human ordering: 2, 1, 3, 5, 4

The model is trained to generate solutions that the reward model scores highly, effectively learning to optimize for human preferences.

4.5.1 Naive vs. RLHF Approach

Naive Approach:

- Run standard RL: 1,000 updates $\times$ 1,000 prompts $\times$ 1,000 rollouts
- Cost: 1,000,000,000 human preference scores
- Computationally prohibitive

RLHF Approach:

- **Step 1**: Collect 1,000 prompts, 5 rollouts each, order from best to worst (cost: 5,000 human scores)
- **Step 2**: Train a neural network simulator of human preferences ("reward model")
- **Step 3**: Run RL as usual, but using the simulator instead of actual humans

This dramatically reduces the cost while maintaining alignment quality.

4.6 RLHF Limitations

The Challenge:

We are doing RL with respect to a lossy simulation of humans. This can be misleading!

Potential Issues:

- The reward model may not capture all aspects of human preferences
- Models can learn to exploit reward model weaknesses
- "Goodharting": optimizing the proxy metric rather than true objective
- Even more subtle: RL discovers ways to "game" the model by finding adversarial examples of the reward model

Example Failure Modes:

- Models generate verbose but shallow responses that score highly
- Over-optimization leads to unnatural, stilted language
- Sycophantic behavior that agrees with user regardless of correctness

This explains why RLHF is powerful but requires careful monitoring and iterative refinement.

5 Critical Model Limitations

5.1 Models Cannot Count

LLMs process text at the token level, not character level. This creates fundamental limitations:

Why This Matters:

- Models see tokens (text chunks), not individual letters
- Cannot reliably count characters in words
- Struggle with letter-level tasks

Example - Counting Letters:

```
Human: "How many 'r's are in 'strawberry'?"
Model: [often incorrect - sees tokens, not individual letters]
```

This limitation persists even in the largest models because the architecture operates on tokens, not characters.

5.2 Models Are Not Good with Spelling

Related to tokenization, spelling is challenging:

Why Models Struggle:

- They process subword tokens, not character sequences
- Cannot easily manipulate individual letters
- Lack explicit character-level representations

Remember: Models see tokens (text chunks), not individual letters!

5.3 Arithmetic Limitations

Bunch of Other Small Random Stuff:

- What is bigger: 9.11 or 9.9?
- Models sometimes struggle with decimal comparisons
- Tokenization affects numerical reasoning
- Need careful prompting or tool use for reliable arithmetic

5.4 Models Can (and Should!) Use Tools

To overcome limitations, modern LLMs integrate with external tools:

Tool Categories:

- **Web search**: Access current information beyond training cutoff
- **Code execution**: Perform exact calculations via Python interpreter
- **Specialized APIs**: Weather data, databases, calculators
- **File systems**: Read and write files

Tool use transforms LLMs from pure language models into capable reasoning agents that know when to leverage external resources.

6 Preview of Future Developments

6.1 Multimodal Capabilities

Future LLMs will natively process:

- Text (current capability)
- Audio input and generation
- Images and video understanding
- Natural conversations with visual context

6.2 Agentic Systems

Evolution toward:

- Long, coherent, error-correcting contexts
- Task-oriented agents that plan and execute
- Pervasive tool use and integration
- Computer-using agents

6.3 Test-Time Training

Exploring:

- Models that continue learning during inference
- Adaptive responses based on conversation context
- Continuous improvement without explicit retraining

7 Where to Find and Track Models

7.1 Proprietary Models

Available on respective websites of LLM providers:

- OpenAI (GPT-4, ChatGPT)
- Anthropic (Claude)
- Google (Gemini)

7.2 Open-Source Models

Model Providers:

- Meta (Llama series)
- DeepSeek
- Mistral AI

Inference Providers:

- TogetherAI - hosts many open-source models
- Local execution: LMStudio for running models on personal hardware

8 Conclusion

Large Language Models represent a remarkable achievement in artificial intelligence, demonstrating sophisticated language understanding and generation through a carefully orchestrated training pipeline. This report has examined the complete lifecycle:

Key Takeaways:

1. **Pretraining** creates versatile base models by learning from vast internet text, but these models lack alignment with human values and intentions
2. **Supervised Fine-Tuning** transforms base models into helpful assistants through curated conversation data, but requires careful dataset curation and is limited by human labeling capacity

3. **Reinforcement Learning** further aligns models with human preferences by having them practice problem-solving and receive feedback, discovering emergent reasoning strategies in the process

4. **Critical Limitations** include hallucinations, poor arithmetic, inability to count or spell reliably, and dependence on tokenization—many addressable through tool integration