

# **Encyclopedia of Large Language Models and Foundation Models**

**Hasi Hays**

Department of Chemical Engineering  
University of Arkansas  
Fayetteville, AR 72701, USA

**First Edition: Jan 2026**



# Abstract

This comprehensive encyclopedia provides an authoritative and detailed reference on Large Language Models (LLMs) and Foundation Models. As artificial intelligence continues to transform technology and society, understanding these powerful systems becomes increasingly critical for researchers, practitioners, policymakers, and informed citizens. Foundation models represent a paradigm shift in machine learning, characterized by their ability to be trained once on broad data at scale and then adapted to numerous downstream tasks. Large Language Models constitute a particularly impactful class of foundation models, demonstrating remarkable capabilities in natural language understanding, generation, reasoning, and multimodal processing. This encyclopedia systematically explores twenty-one major dimensions organized into thematic parts: **Foundations** covering historical development, transformer architectures, attention mechanisms, and scaling laws; **Models and Training** examining major model families (GPT, Claude, Gemini, LLaMA), training methodologies, and alignment techniques including RLHF and constitutional AI; **Interaction and Applications** detailing prompt engineering, capabilities across domains, and evaluation benchmarks; **Safety and Society** addressing hallucination, bias, privacy, misuse, and governance frameworks; **Infrastructure** covering hardware systems, distributed training, data science, and deployment at scale; **Industry** analyzing major players, economics, business models, and regulatory landscapes; **Advanced Paradigms** including retrieval-augmented generation (RAG), autonomous agents, interpretability and mechanistic understanding, multimodal foundation models, knowledge representation, multilingual NLP, model compression, and legal frameworks spanning EU AI Act, US executive orders, and international governance. Each chapter provides mathematical rigor where appropriate, code-generated figures with real benchmark data, and extensive citations. This encyclopedia serves as both an educational resource for those entering the field and a comprehensive reference for experienced practitioners advancing the state of the art in this transformative domain.



# Preface

The field of large language models (LLMs) and foundation models has experienced unprecedented growth and transformation in recent years. What began with statistical language models in the 1980s and evolved through recurrent neural networks in the 2010s has now reached a level of capability that was scarcely imaginable a decade ago. The introduction of the transformer architecture in 2017 catalyzed a revolution that continues to accelerate. Models with hundreds of billions and now trillions of parameters can engage in complex reasoning, write sophisticated code, analyze scientific literature, conduct mathematical proofs, and assist with creative endeavors, all while exhibiting emergent capabilities that were not explicitly programmed. This encyclopedia represents several months of effort to create a comprehensive, technically rigorous, yet accessible reference that captures the current state of knowledge in this rapidly evolving field. It is designed to serve multiple audiences: graduate students and researchers seeking deep technical understanding of architectures, training methods, and theoretical foundations; practitioners implementing these systems in real-world applications who need practical guidance on deployment, optimization, and evaluation; policymakers and legal professionals grappling with governance questions raised by increasingly capable AI systems; and informed citizens seeking to understand technologies that are reshaping industries, labor markets, and society itself. All the chapters are organized to support both linear reading and selective reference. The opening chapters establish historical context and technical foundations, explaining how we arrived at modern transformer-based architectures and the mathematical principles underlying their operation. Subsequent chapters provide comprehensive coverage of major model families from OpenAI's GPT series to Anthropic's Claude, Google's Gemini, Meta's LLaMA, and the vibrant open-source ecosystem. Training methodologies receive extensive treatment, including pretraining objectives, fine-tuning approaches, and the critical alignment techniques such as RLHF and constitutional AI that shape model behavior.

The middle sections address practical concerns: prompt engineering strategies that have become essential skills for effective model interaction, the vast landscape of applications across domains from healthcare to legal to scientific research, and rigorous evaluation frameworks including standard benchmarks and their limitations. Safety and ethics receive thorough examination, covering hallucination, bias, privacy, potential misuse, and the emerging governance landscape. Later chapters delve into the infrastructure enabling these systems from GPU clusters and distributed training frameworks to the data science pipelines that curate trillion-token corpora. The industry analysis examines the competitive dynamics, economic models, and regulatory environment shaping commercial deployment. The final chapters explore cutting-edge developments: retrieval-augmented generation that grounds models in external knowledge, autonomous agents that can plan and execute complex tasks, interpretability research seeking to understand model internals, multimodal systems spanning text, vision, audio, and beyond, and the legal frameworks emerging worldwide to govern AI development. Each chapter includes mathematical formulations where appropriate. Figures are generated programmatically from published results to ensure accuracy and reproducibility. As this field continues to evolve at remarkable pace, this encyclopedia represents a snapshot of knowledge as of early 2026.

We hope this work contributes to broader understanding and responsible development of these powerful technologies, serving as a foundation for the researchers, engineers, policymakers, and citizens who will shape how large language models and foundation models are developed and deployed in the years ahead.

*Hasi Hays (Ph.D.)  
Department of Chemical Engineering  
University of Arkansas  
Fayetteville, AR 72701, USA*

*January 2026*



# Contents

<b>Abstract</b>	i
<b>Preface</b>	iii
<b>List of Figures</b>	xvii
<b>1 Foundations and historical development</b>	1
1.1 Introduction to Language Models . . . . .	1
1.1.1 Definition and Scope . . . . .	1
1.1.2 Historical Context and Motivation . . . . .	1
1.2 Evolution of Language Modeling Approaches . . . . .	2
1.2.1 Statistical Language Models . . . . .	2
1.2.2 Neural Language Models . . . . .	2
1.3 The Transformer Revolution . . . . .	3
1.3.1 Attention Mechanisms . . . . .	3
1.3.2 Transformer Architecture . . . . .	3
1.3.3 Impact and Adoption . . . . .	3
1.4 The Pre-training and Fine-tuning Paradigm . . . . .	4
1.4.1 Transfer Learning in NLP . . . . .	4
1.4.2 Foundational Models . . . . .	4
1.4.3 The Scaling Era Begins . . . . .	4
<b>2 Core architectures and mechanisms</b>	7
2.1 The Transformer Architecture in Depth . . . . .	7
2.1.1 Self-Attention Mechanism . . . . .	7
2.1.2 Position-wise Feed-Forward Networks . . . . .	7
2.1.3 Positional Encoding . . . . .	8
2.1.4 Layer Normalization and Residual Connections . . . . .	8
2.1.5 Causal Masking for Language Modeling . . . . .	8
2.2 Architecture Variants . . . . .	8
2.2.1 Encoder-Only Models . . . . .	8
2.2.2 Decoder-Only Models . . . . .	8
2.2.3 Encoder-Decoder Models . . . . .	9
2.3 Scaling Laws and Emergent Properties . . . . .	9
2.3.1 Neural Scaling Laws . . . . .	9
2.3.2 Emergent Abilities . . . . .	9
2.3.3 In-Context Learning . . . . .	10
2.4 Attention Mechanism Variations . . . . .	10
2.4.1 Sparse Attention Patterns . . . . .	10
2.4.2 Efficient Attention Mechanisms . . . . .	11
2.4.3 Rotary Position Embeddings (RoPE) . . . . .	11
2.4.4 Grouped-Query Attention (GQA) . . . . .	11
2.5 Tokenization . . . . .	11
2.5.1 Subword Tokenization Methods . . . . .	11
2.5.2 Vocabulary Size Trade-offs . . . . .	12
2.5.3 Impact on Model Behavior . . . . .	13

<b>3 Major models and families</b>	<b>15</b>
3.1 GPT Series . . . . .	15
3.1.1 GPT-1 . . . . .	15
3.1.2 GPT-2 . . . . .	15
3.1.3 GPT-3 . . . . .	15
3.1.4 GPT-4 . . . . .	16
3.1.5 GPT-4 Turbo and GPT-4o . . . . .	16
3.2 BERT and Encoder Models . . . . .	16
3.2.1 BERT . . . . .	16
3.2.2 BERT Variants . . . . .	17
3.3 T5 and Encoder-Decoder Models . . . . .	18
3.3.1 T5 (Text-to-Text Transfer Transformer) . . . . .	18
3.3.2 BART . . . . .	18
3.3.3 UL2 . . . . .	18
3.4 LLaMA and Open Models . . . . .	18
3.4.1 LLaMA . . . . .	18
3.4.2 LLaMA 2 . . . . .	19
3.4.3 LLaMA 3 . . . . .	19
3.4.4 Open Model Ecosystem . . . . .	19
3.5 Claude . . . . .	20
3.6 PaLM and Gemini . . . . .	20
3.6.1 PaLM (Pathways Language Model) . . . . .	20
3.6.2 Gemini . . . . .	21
3.7 Specialized and Domain-Specific Models . . . . .	21
3.7.1 Code Models . . . . .	21
3.7.2 Scientific and Biomedical Models . . . . .	21
3.7.3 Multilingual Models . . . . .	22
3.8 Mixture-of-Experts Models . . . . .	22
3.8.1 Architectural Approach . . . . .	22
3.8.2 Switch Transformer . . . . .	23
3.8.3 GLaM . . . . .	23
3.8.4 Mixtral . . . . .	23
<b>4 Training methodologies</b>	<b>25</b>
4.1 Pre-training Objectives . . . . .	25
4.1.1 Causal Language Modeling . . . . .	25
4.1.2 Masked Language Modeling . . . . .	25
4.1.3 Prefix Language Modeling . . . . .	25
4.1.4 Span Corruption . . . . .	26
4.2 Training Data and Curation . . . . .	26
4.2.1 Data Sources . . . . .	26
4.2.2 Data Quality and Filtering . . . . .	27
4.2.3 Data Scaling Considerations . . . . .	27
4.3 Optimization and Training Techniques . . . . .	27
4.3.1 Optimizers . . . . .	27
4.3.2 Learning Rate Schedules . . . . .	27
4.3.3 Gradient Clipping and Stability . . . . .	28
4.3.4 Mixed Precision Training . . . . .	28
4.3.5 Batch Size and Accumulation . . . . .	28
4.4 Distributed Training and Parallelism . . . . .	28
4.4.1 Data Parallelism . . . . .	28
4.4.2 Model Parallelism . . . . .	28
4.4.3 Zero Redundancy Optimizer (ZeRO) . . . . .	29
4.4.4 3D Parallelism . . . . .	29
4.5 Fine-tuning Techniques . . . . .	29
4.5.1 Full Fine-tuning . . . . .	29
4.5.2 Parameter-Efficient Fine-Tuning (PEFT) . . . . .	29
4.6 Instruction Tuning . . . . .	30

4.6.1	Motivation and Approach . . . . .	30
4.6.2	Instruction Datasets . . . . .	30
4.6.3	Self-Instruct and Data Generation . . . . .	31
4.7	Reinforcement Learning from Human Feedback (RLHF) . . . . .	31
4.7.1	Motivation . . . . .	31
4.7.2	RLHF Pipeline . . . . .	31
4.7.3	InstructGPT and ChatGPT . . . . .	32
4.7.4	Challenges and Limitations . . . . .	32
4.8	Constitutional AI and RLAIF . . . . .	32
4.8.1	Constitutional AI . . . . .	32
4.8.2	Direct Preference Optimization (DPO) . . . . .	33
<b>5</b>	<b>Prompt engineering and interaction design</b>	<b>35</b>
5.1	Foundations of prompting . . . . .	35
5.1.1	The role of prompts . . . . .	35
5.1.2	Prompt components . . . . .	35
5.1.3	Zero-shot, few-shot, and many-shot prompting . . . . .	36
5.2	Advanced prompting techniques . . . . .	37
5.2.1	Chain-of-thought prompting . . . . .	37
5.2.2	Self-consistency . . . . .	37
5.2.3	Tree-of-thoughts . . . . .	38
5.2.4	Least-to-most prompting . . . . .	38
5.2.5	Decomposed prompting . . . . .	39
5.2.6	Self-refinement . . . . .	39
5.2.7	Reflexion . . . . .	39
5.3	Prompt optimization . . . . .	39
5.3.1	Manual prompt engineering . . . . .	39
5.3.2	Automatic prompt optimization . . . . .	40
5.3.3	Prompt ensembles . . . . .	40
5.4	Prompting for specific tasks . . . . .	40
5.4.1	Information extraction . . . . .	40
5.4.2	Classification . . . . .	41
5.4.3	Generation tasks . . . . .	41
5.4.4	Reasoning and analysis . . . . .	42
5.5	Prompt security and robustness . . . . .	42
5.5.1	Prompt injection attacks . . . . .	42
5.5.2	Defense strategies . . . . .	43
5.5.3	Prompt robustness . . . . .	43
5.6	Evaluation of prompts . . . . .	44
5.6.1	Performance metrics . . . . .	44
5.6.2	Evaluation methodology . . . . .	44
5.6.3	A/B testing for prompts . . . . .	44
5.7	Prompting in production systems . . . . .	44
5.7.1	Prompt management . . . . .	44
5.7.2	Dynamic prompting . . . . .	45
5.7.3	Cost optimization . . . . .	45
5.7.4	Monitoring and maintenance . . . . .	45
<b>6</b>	<b>Capabilities and applications</b>	<b>47</b>
6.1	Natural language understanding . . . . .	47
6.1.1	Text Classification . . . . .	47
6.1.2	Named Entity Recognition (NER) . . . . .	47
6.1.3	Relation Extraction . . . . .	47
6.1.4	Question Answering . . . . .	48
6.1.5	Semantic Similarity and Entailment . . . . .	48
6.2	Text Generation . . . . .	48
6.2.1	Summarization . . . . .	48
6.2.2	Creative Writing . . . . .	49

6.2.3 Content Creation . . . . .	49
6.3 Translation and Multilingual Capabilities . . . . .	50
6.3.1 Machine Translation . . . . .	50
6.3.2 Cross-Lingual Transfer . . . . .	50
6.3.3 Code-Switching and Multilingual Dialogue . . . . .	50
6.4 Reasoning and Problem Solving . . . . .	50
6.4.1 Mathematical Reasoning . . . . .	50
6.4.2 Logical Reasoning . . . . .	51
6.4.3 Commonsense Reasoning . . . . .	51
6.4.4 Chain-of-Thought Prompting . . . . .	51
6.5 Code Understanding and Generation . . . . .	51
6.5.1 Code Completion and Generation . . . . .	51
6.5.2 Code Understanding . . . . .	52
6.5.3 Code Translation and Refactoring . . . . .	52
6.5.4 Testing and Debugging . . . . .	52
6.6 Multimodal Capabilities . . . . .	52
6.6.1 Vision-Language Models . . . . .	52
6.6.2 Document Understanding . . . . .	53
6.6.3 Audio and Speech . . . . .	53
6.6.4 Video Understanding . . . . .	53
6.7 Domain-Specific Applications . . . . .	54
6.7.1 Healthcare and Biomedicine . . . . .	54
6.7.2 Legal . . . . .	54
6.7.3 Education . . . . .	54
6.7.4 Scientific Research . . . . .	55
6.7.5 Business and Enterprise . . . . .	55
<b>7 Evaluation and benchmarks</b>	<b>57</b>
7.1 Evaluation Challenges . . . . .	57
7.1.1 Fundamental Difficulties . . . . .	57
7.1.2 Automatic vs. Human Evaluation . . . . .	57
7.2 Understanding Benchmarks . . . . .	58
7.2.1 GLUE and SuperGLUE . . . . .	58
7.2.2 Question Answering Benchmarks . . . . .	58
7.2.3 Commonsense Reasoning . . . . .	58
7.3 Reasoning and Problem-Solving Benchmarks . . . . .	58
7.3.1 Mathematical Reasoning . . . . .	58
7.3.2 Code Benchmarks . . . . .	59
7.3.3 Multi-Step Reasoning . . . . .	60
7.4 Comprehensive Evaluation Suites . . . . .	61
7.4.1 MMLU (Massive Multitask Language Understanding) . . . . .	61
7.4.2 HELM (Holistic Evaluation of Language Models) . . . . .	61
7.4.3 LMSYS Chatbot Arena . . . . .	62
7.5 Specialized Evaluation . . . . .	63
7.5.1 Truthfulness and Hallucination . . . . .	63
7.5.2 Safety and Toxicity . . . . .	63
7.5.3 Multilingual Evaluation . . . . .	63
7.5.4 Long-Context Evaluation . . . . .	63
7.6 Evaluation Metrics . . . . .	64
7.6.1 Perplexity . . . . .	64
7.6.2 Accuracy . . . . .	64
7.6.3 F1 Score . . . . .	64
7.6.4 BLEU (Bilingual Evaluation Understudy) . . . . .	64
7.6.5 ROUGE (Recall-Oriented Understudy for Gisting Evaluation) . . . . .	64
7.6.6 BERTScore . . . . .	64
7.6.7 Pass@k . . . . .	65

<b>8 Safety, ethics, and societal impact</b>	<b>67</b>
8.1 Safety Challenges . . . . .	67
8.1.1 Hallucination and Factual Accuracy . . . . .	67
8.1.2 Harmful Content Generation . . . . .	67
8.1.3 Jailbreaking and Adversarial Prompts . . . . .	68
8.1.4 Prompt Injection . . . . .	68
8.2 Bias and Fairness . . . . .	69
8.2.1 Sources of Bias . . . . .	69
8.2.2 Types of Bias . . . . .	69
8.2.3 Measurement and Detection . . . . .	69
8.2.4 Mitigation Approaches . . . . .	70
8.2.5 Fundamental Challenges . . . . .	70
8.3 Privacy and Data Protection . . . . .	70
8.3.1 Training Data Privacy . . . . .	70
8.3.2 Usage Privacy . . . . .	71
8.4 Intellectual Property and Copyright . . . . .	71
8.4.1 Training Data Copyright . . . . .	71
8.4.2 Generated Content Ownership . . . . .	71
8.4.3 Code Generation Concerns . . . . .	72
8.5 Misuse and Dual Use . . . . .	72
8.5.1 Potential Misuse Vectors . . . . .	72
8.5.2 Mitigation Strategies . . . . .	73
8.6 Societal Impact . . . . .	73
8.6.1 Labor Market Effects . . . . .	73
8.6.2 Access and Equity . . . . .	74
8.6.3 Information Ecosystem Effects . . . . .	74
8.6.4 Education and Learning . . . . .	74
8.7 Governance and Regulation . . . . .	75
8.7.1 Regulatory Approaches . . . . .	75
8.7.2 Industry Self-Governance . . . . .	75
8.7.3 Research and Development Governance . . . . .	76
8.8 Alignment and Value Specification . . . . .	76
8.8.1 The Alignment Problem . . . . .	76
8.8.2 Technical Alignment Approaches . . . . .	76
8.8.3 Value Pluralism . . . . .	77
<b>9 Infrastructure and systems for LLM training</b>	<b>79</b>
9.1 Hardware infrastructure . . . . .	79
9.1.1 Graphics processing units . . . . .	79
9.1.2 Tensor processing units . . . . .	80
9.1.3 Alternative accelerators . . . . .	80
9.1.4 Networking infrastructure . . . . .	81
9.2 Distributed training systems . . . . .	81
9.2.1 Parallelism strategies . . . . .	81
9.2.2 Memory optimization . . . . .	83
9.2.3 Training frameworks . . . . .	84
9.3 Communication optimization . . . . .	85
9.3.1 Collective operations . . . . .	85
9.3.2 Communication-computation overlap . . . . .	86
9.3.3 Compression techniques . . . . .	86
9.4 Training stability and efficiency . . . . .	86
9.4.1 Numerical stability . . . . .	86
9.4.2 Learning rate scheduling . . . . .	87
9.4.3 Checkpointing and fault tolerance . . . . .	87
9.5 Efficiency techniques . . . . .	88
9.5.1 FlashAttention . . . . .	88
9.5.2 Kernel fusion . . . . .	88
9.5.3 Operator optimization . . . . .	88

9.6	Energy efficiency and sustainability . . . . .	89
9.6.1	Carbon footprint . . . . .	89
9.6.2	Efficiency improvements . . . . .	89
9.6.3	Reporting and transparency . . . . .	89
<b>10</b>	<b>Data science for large language models</b>	<b>91</b>
10.1	Training data sources . . . . .	91
10.1.1	Web-scale text corpora . . . . .	91
10.1.2	Curated text sources . . . . .	92
10.1.3	Code repositories . . . . .	92
10.1.4	Conversational and instructional data . . . . .	93
10.2	Data curation and preprocessing . . . . .	93
10.2.1	Quality filtering . . . . .	93
10.2.2	Deduplication . . . . .	94
10.2.3	Data mixing and sampling . . . . .	94
10.3	Synthetic data generation . . . . .	95
10.3.1	Instruction generation . . . . .	95
10.3.2	Reasoning chain synthesis . . . . .	96
10.3.3	Code generation . . . . .	96
10.3.4	Quality and limitations . . . . .	96
10.4	Evaluation data contamination . . . . .	97
10.4.1	Sources of contamination . . . . .	97
10.4.2	Detection methods . . . . .	97
10.4.3	Mitigation strategies . . . . .	97
10.5	Multilingual data . . . . .	97
10.5.1	Language distribution . . . . .	97
10.5.2	Multilingual corpora . . . . .	98
10.5.3	Challenges . . . . .	98
10.5.4	Cross-lingual transfer . . . . .	98
10.6	Legal and ethical considerations . . . . .	98
10.6.1	Copyright and licensing . . . . .	98
10.6.2	Privacy concerns . . . . .	98
10.6.3	Bias and representation . . . . .	99
10.6.4	Documentation practices . . . . .	99
10.7	Data infrastructure . . . . .	99
10.7.1	Storage and processing . . . . .	99
10.7.2	Data loading . . . . .	99
10.7.3	Preprocessing pipelines . . . . .	100
<b>11</b>	<b>Deployment and MLOps for LLMs</b>	<b>101</b>
11.1	Inference optimization . . . . .	101
11.1.1	Quantization . . . . .	101
11.1.2	Key-value cache optimization . . . . .	102
11.1.3	Batching strategies . . . . .	102
11.1.4	Speculative decoding . . . . .	103
11.2	Serving infrastructure . . . . .	103
11.2.1	Inference servers . . . . .	103
11.2.2	API design . . . . .	104
11.2.3	Load balancing and scaling . . . . .	104
11.3	Operational practices . . . . .	105
11.3.1	Monitoring and observability . . . . .	105
11.3.2	Cost management . . . . .	105
11.3.3	Security considerations . . . . .	106
11.4	Model lifecycle management . . . . .	106
11.4.1	Version control . . . . .	106
11.4.2	A/B testing and experimentation . . . . .	106
11.4.3	Model updates . . . . .	107
11.5	Edge and on-device deployment . . . . .	107

11.5.1 Mobile deployment . . . . .	107
11.5.2 Browser deployment . . . . .	107
11.5.3 Embedded systems . . . . .	107
11.6 Retrieval-augmented generation deployment . . . . .	108
11.6.1 Vector database integration . . . . .	108
11.6.2 RAG pipeline components . . . . .	108
11.6.3 Operational considerations . . . . .	108
11.7 Fine-tuning operations . . . . .	108
11.7.1 Training infrastructure . . . . .	108
11.7.2 Continuous fine-tuning . . . . .	108
11.7.3 PEFT deployment . . . . .	109
<b>12 Industry landscape and economics</b>	<b>111</b>
12.1 Major industry players . . . . .	111
12.1.1 Frontier model developers . . . . .	111
12.1.2 Cloud providers . . . . .	112
12.1.3 Hardware providers . . . . .	113
12.2 Economic analysis . . . . .	113
12.2.1 Training costs . . . . .	113
12.2.2 Inference economics . . . . .	114
12.2.3 Market sizing . . . . .	115
12.3 Business models . . . . .	115
12.3.1 API-as-a-service . . . . .	115
12.3.2 Platform and infrastructure . . . . .	115
12.3.3 Vertical applications . . . . .	115
12.3.4 Open-source monetization . . . . .	116
12.4 Competitive dynamics . . . . .	116
12.4.1 Moats and differentiation . . . . .	116
12.4.2 Open vs closed models . . . . .	116
12.4.3 Consolidation trends . . . . .	117
12.5 Investment landscape . . . . .	117
12.5.1 Venture capital . . . . .	117
12.5.2 Corporate investment . . . . .	117
12.5.3 Public markets . . . . .	117
12.6 Regulatory and policy environment . . . . .	118
12.6.1 Emerging regulations . . . . .	118
12.6.2 Policy debates . . . . .	118
12.7 Workforce and labor impacts . . . . .	119
12.7.1 Job displacement concerns . . . . .	119
12.7.2 Job creation . . . . .	119
12.7.3 Skill transformation . . . . .	119
12.7.4 Productivity effects . . . . .	120
12.8 Geopolitical dimensions . . . . .	120
12.8.1 US-China competition . . . . .	120
12.8.2 Technology sovereignty . . . . .	120
12.8.3 International cooperation . . . . .	120
<b>13 Retrieval-augmented generation</b>	<b>121</b>
13.1 Foundations of RAG . . . . .	121
13.1.1 Motivation and core concepts . . . . .	121
13.1.2 RAG architecture patterns . . . . .	121
13.2 Text embeddings and vector representations . . . . .	122
13.2.1 Dense retrieval . . . . .	122
13.2.2 Embedding models . . . . .	122
13.2.3 Sparse retrieval . . . . .	123
13.2.4 Hybrid retrieval . . . . .	123
13.3 Vector databases and indexing . . . . .	123
13.3.1 Approximate nearest neighbor search . . . . .	123

13.3.2 Vector database systems . . . . .	124
13.3.3 Indexing strategies . . . . .	124
13.4 Retrieval optimization . . . . .	125
13.4.1 Query processing . . . . .	125
13.4.2 Reranking . . . . .	125
13.4.3 Multi-hop retrieval . . . . .	126
13.5 RAG system design . . . . .	126
13.5.1 Context construction . . . . .	126
13.5.2 Generation with retrieved context . . . . .	126
13.5.3 Evaluation of RAG systems . . . . .	126
13.6 Advanced RAG techniques . . . . .	127
13.6.1 Agentic RAG . . . . .	127
13.6.2 Graph RAG . . . . .	127
13.6.3 Multimodal RAG . . . . .	127
<b>14 Agents and autonomous systems</b>	<b>129</b>
14.1 Foundations of LLM agents . . . . .	129
14.1.1 Definition and components . . . . .	129
14.1.2 Agent architectures . . . . .	129
14.1.3 Cognitive architectures . . . . .	130
14.2 Memory systems . . . . .	130
14.2.1 Short-term memory . . . . .	130
14.2.2 Long-term memory . . . . .	130
14.2.3 Memory retrieval and consolidation . . . . .	131
14.3 Tool use and function calling . . . . .	131
14.3.1 Function calling interfaces . . . . .	131
14.3.2 Tool categories . . . . .	132
14.3.3 Tool learning . . . . .	132
14.4 Planning and reasoning . . . . .	132
14.4.1 Task decomposition . . . . .	132
14.4.2 Planning algorithms . . . . .	133
14.4.3 Verification and error recovery . . . . .	133
14.5 Multi-agent systems . . . . .	133
14.5.1 Agent collaboration patterns . . . . .	133
14.5.2 Multi-agent frameworks . . . . .	134
14.5.3 Communication protocols . . . . .	134
14.6 Agent applications . . . . .	135
14.6.1 Code agents . . . . .	135
14.6.2 Research agents . . . . .	135
14.6.3 Web agents . . . . .	135
14.6.4 Personal assistants . . . . .	135
14.7 Evaluation and benchmarks . . . . .	135
14.7.1 Agent benchmarks . . . . .	135
14.7.2 Evaluation dimensions . . . . .	136
14.8 Safety and alignment for agents . . . . .	136
14.8.1 Risks specific to agents . . . . .	136
14.8.2 Mitigation strategies . . . . .	136
<b>15 Interpretability and mechanistic understanding</b>	<b>137</b>
15.1 Motivation for interpretability . . . . .	137
15.1.1 Why interpretability matters . . . . .	137
15.1.2 Levels of analysis . . . . .	137
15.2 Probing and representation analysis . . . . .	137
15.2.1 Linear probing . . . . .	137
15.2.2 Representation similarity analysis . . . . .	138
15.2.3 Geometry of representations . . . . .	138
15.3 Attention analysis . . . . .	138
15.3.1 Attention visualization . . . . .	138

15.3.2 Attention head functions . . . . .	138
15.3.3 Limitations of attention analysis . . . . .	138
15.4 Mechanistic interpretability . . . . .	139
15.4.1 Circuits and features . . . . .	139
15.4.2 Sparse autoencoders . . . . .	139
15.4.3 Activation patching . . . . .	139
15.4.4 Path patching . . . . .	139
15.5 Knowledge localization . . . . .	140
15.5.1 Where is knowledge stored? . . . . .	140
15.5.2 Knowledge editing . . . . .	140
15.5.3 Limitations . . . . .	140
15.6 Behavioral interpretability . . . . .	140
15.6.1 Influence functions . . . . .	140
15.6.2 Concept bottleneck models . . . . .	140
15.6.3 Natural language explanations . . . . .	141
15.7 Tools and frameworks . . . . .	141
15.7.1 Interpretability libraries . . . . .	141
15.7.2 Visualization tools . . . . .	141
15.8 Challenges and open problems . . . . .	141
15.8.1 Scalability . . . . .	141
15.8.2 Faithfulness . . . . .	141
15.8.3 Completeness . . . . .	141
<b>16 Multimodal foundation models</b> . . . . .	<b>143</b>
16.1 Foundations of multimodal learning . . . . .	143
16.1.1 Modality representations . . . . .	143
16.1.2 Multimodal fusion strategies . . . . .	143
16.2 Vision-language models . . . . .	144
16.2.1 Contrastive vision-language models . . . . .	144
16.2.2 Generative vision-language models . . . . .	144
16.2.3 Vision encoders . . . . .	145
16.2.4 Document understanding . . . . .	145
16.3 Audio and speech models . . . . .	146
16.3.1 Speech recognition . . . . .	146
16.3.2 Speech synthesis . . . . .	146
16.3.3 Audio understanding . . . . .	146
16.4 Video understanding . . . . .	146
16.4.1 Video foundation models . . . . .	146
16.4.2 Long video understanding . . . . .	147
16.4.3 Video generation . . . . .	147
16.5 Unified multimodal models . . . . .	147
16.5.1 Any-to-any models . . . . .	147
16.5.2 Multimodal generation . . . . .	148
16.6 Training multimodal models . . . . .	148
16.6.1 Data considerations . . . . .	148
16.6.2 Training objectives . . . . .	148
16.6.3 Alignment techniques . . . . .	148
16.7 Evaluation . . . . .	149
16.7.1 Vision-language benchmarks . . . . .	149
16.7.2 Video benchmarks . . . . .	149
<b>17 Knowledge representation and reasoning</b> . . . . .	<b>151</b>
17.1 Knowledge in language models . . . . .	151
17.1.1 Parametric vs. non-parametric knowledge . . . . .	151
17.1.2 Knowledge storage mechanisms . . . . .	151
17.2 Factual recall and knowledge probing . . . . .	152
17.2.1 Knowledge probing . . . . .	152
17.2.2 Knowledge boundaries . . . . .	152

17.3 Knowledge editing . . . . .	152
17.3.1 Motivation . . . . .	152
17.3.2 Editing methods . . . . .	152
17.3.3 Evaluation of edits . . . . .	153
17.3.4 Limitations and challenges . . . . .	153
17.4 Knowledge graphs and structured knowledge . . . . .	153
17.4.1 Integration with LLMs . . . . .	153
17.4.2 Structured reasoning . . . . .	153
17.5 Grounding and world models . . . . .	154
17.5.1 Symbol grounding . . . . .	154
17.5.2 World models . . . . .	154
17.6 Reasoning with knowledge . . . . .	154
17.6.1 Multi-hop reasoning . . . . .	154
17.6.2 Commonsense reasoning . . . . .	154
17.6.3 Logical reasoning . . . . .	154
<b>18 Multilingual and cross-lingual NLP</b>	<b>155</b>
18.1 Multilingual language models . . . . .	155
18.1.1 Evolution of multilingual models . . . . .	155
18.1.2 Modern multilingual LLMs . . . . .	155
18.2 Cross-lingual transfer . . . . .	156
18.2.1 Zero-shot cross-lingual transfer . . . . .	156
18.2.2 Mechanisms of cross-lingual transfer . . . . .	156
18.2.3 Improving cross-lingual transfer . . . . .	156
18.3 Machine translation . . . . .	156
18.3.1 Neural machine translation with LLMs . . . . .	156
18.3.2 Translation quality . . . . .	157
18.4 Low-resource languages . . . . .	157
18.4.1 Challenges . . . . .	157
18.4.2 Approaches . . . . .	157
18.5 Multilingual evaluation . . . . .	157
18.5.1 Benchmarks . . . . .	157
18.5.2 Evaluation challenges . . . . .	158
18.6 Linguistic diversity considerations . . . . .	158
18.6.1 Morphologically rich languages . . . . .	158
18.6.2 Script diversity . . . . .	158
18.6.3 Code-switching . . . . .	158
<b>19 Model compression and efficiency</b>	<b>159</b>
19.1 Quantization . . . . .	159
19.1.1 Fundamentals . . . . .	159
19.1.2 Post-training quantization (PTQ) . . . . .	159
19.1.3 Quantization-aware training (QAT) . . . . .	160
19.1.4 Mixed-precision strategies . . . . .	160
19.2 Pruning . . . . .	160
19.2.1 Pruning strategies . . . . .	160
19.2.2 Pruning criteria . . . . .	160
19.2.3 Layer and attention head pruning . . . . .	161
19.3 Knowledge distillation . . . . .	161
19.3.1 Distillation approaches . . . . .	161
19.3.2 LLM-specific distillation . . . . .	161
19.4 Efficient architectures . . . . .	161
19.4.1 Linear attention . . . . .	161
19.4.2 State space models . . . . .	161
19.4.3 Mixture of Experts . . . . .	162
19.5 Inference optimization . . . . .	162
19.5.1 KV cache optimization . . . . .	162
19.5.2 Speculative decoding . . . . .	162

19.5.3 Continuous batching . . . . .	163
19.6 Hardware considerations . . . . .	163
19.6.1 GPU optimization . . . . .	163
19.6.2 Alternative hardware . . . . .	163
<b>20 Legal and regulatory frameworks</b>	<b>165</b>
20.1 Overview of AI regulation . . . . .	165
20.1.1 Regulatory approaches . . . . .	165
20.1.2 Key regulatory objectives . . . . .	165
20.2 European Union AI Act . . . . .	165
20.2.1 Risk categories . . . . .	165
20.2.2 General-purpose AI provisions . . . . .	166
20.2.3 Compliance requirements . . . . .	166
20.3 United States regulatory landscape . . . . .	167
20.3.1 Executive Order on AI (2023) . . . . .	167
20.3.2 Agency-specific regulation . . . . .	167
20.3.3 State-level regulation . . . . .	167
20.4 China AI governance . . . . .	167
20.4.1 Regulatory framework . . . . .	167
20.5 International frameworks . . . . .	168
20.5.1 OECD AI Principles . . . . .	168
20.5.2 G7 Hiroshima AI Process . . . . .	168
20.5.3 UNESCO AI Ethics . . . . .	168
20.6 Intellectual property issues . . . . .	168
20.6.1 Training data copyright . . . . .	168
20.6.2 Generated content ownership . . . . .	169
20.6.3 Patent considerations . . . . .	169
20.7 Liability frameworks . . . . .	169
20.7.1 Product liability . . . . .	169
20.7.2 Professional liability . . . . .	169
20.8 Privacy regulations . . . . .	169
20.8.1 GDPR implications . . . . .	169
20.8.2 US privacy laws . . . . .	170
20.9 Industry self-governance . . . . .	170
20.9.1 Voluntary commitments . . . . .	170
20.9.2 Standards development . . . . .	170
20.10 Compliance strategies . . . . .	170
20.10.1 For AI developers . . . . .	170
20.10.2 For deployers . . . . .	170
<b>21 Advanced topics and future directions</b>	<b>171</b>
21.1 Efficient architectures and training . . . . .	171
21.1.1 Parameter-Efficient Models . . . . .	171
21.1.2 Retrieval-Augmented Generation . . . . .	171
21.1.3 Long Context and Efficient Attention . . . . .	172
21.2 Multimodal Foundation Models . . . . .	173
21.2.1 Vision-Language Models . . . . .	173
21.2.2 Audio, Video, and Beyond . . . . .	174
21.3 Agents and Tool Use . . . . .	174
21.3.1 LLM Agents . . . . .	174
21.3.2 Tool Use and Function Calling . . . . .	175
21.3.3 Multi-Agent Systems . . . . .	175
21.4 Embodied AI and Robotics . . . . .	175
21.4.1 Foundation Models for Robotics . . . . .	175
21.5 Reasoning and Planning . . . . .	176
21.5.1 Advanced Reasoning Techniques . . . . .	176
21.5.2 Integration with Symbolic Systems . . . . .	176
21.5.3 Mathematical and Scientific Reasoning . . . . .	177

21.6 Interpretability and Mechanistic Understanding . . . . .	177
21.6.1 Motivation . . . . .	177
21.6.2 Approaches . . . . .	177
21.6.3 Challenges . . . . .	178
21.7 Emerging Architectures and Paradigms . . . . .	178
21.7.1 State Space Models . . . . .	178
21.7.2 Hybrid Architectures . . . . .	178
21.7.3 Novel Training Paradigms . . . . .	178
21.8 Future Directions and Open Challenges . . . . .	179
21.8.1 Scaling Frontiers . . . . .	179
21.8.2 Capabilities . . . . .	179
21.8.3 Safety and Alignment . . . . .	180
21.8.4 Democratization and Access . . . . .	180
21.8.5 Scientific Understanding . . . . .	180
21.8.6 Societal Integration . . . . .	181
21.9 Conclusion . . . . .	181
<b>Glossary</b>	<b>183</b>
<b>Index of Models</b>	<b>187</b>

# List of Figures

1.1	Evolution of large language model scale (2018–2024) . . . . .	5
2.1	Neural scaling laws . . . . .	10
2.2	Computational complexity of attention mechanisms . . . . .	12
7.1	Mathematical reasoning benchmark performance . . . . .	59
7.2	HumanEval code generation benchmark . . . . .	60
7.3	MMLU benchmark performance comparison . . . . .	61
7.4	Multi-dimensional capability comparison of frontier models . . . . .	62
9.1	Training compute growth in AI models (2012–2024) . . . . .	81
21.1	Context window evolution in language models (2018–2024) . . . . .	173



# Chapter 1

## Foundations and historical development

### 1.1 Introduction to Language Models

#### 1.1.1 Definition and Scope

A **language model** is a probability distribution over sequences of tokens (words, subwords, or characters) that captures the statistical structure of natural language. Formally, for a sequence of tokens  $w_1, w_2, \dots, w_n$ , a language model assigns a probability:

$$P(w_1, w_2, \dots, w_n) = \prod_{i=1}^n P(w_i | w_1, \dots, w_{i-1}) \quad (1.1)$$

The quality of a language model is typically measured by its ability to predict held-out text, quantified by metrics such as perplexity, which represents the average branching factor when predicting the next token.

**Large Language Models (LLMs)** specifically refer to neural language models with a large number of parameters (typically billions to trillions) trained on extensive text corpora (see [Figure 1.1](#) for the evolution of model scale). These models exhibit qualitatively different behaviors from smaller models, including few-shot learning, chain-of-thought reasoning, and emergent capabilities [\[1, 2\]](#).

**Foundation models** represent a broader category encompassing models trained on broad data at scale that can be adapted to a wide range of downstream tasks [\[3\]](#). While LLMs are the most prominent examples, foundation models also include vision models, multimodal models, and domain-specific models.

#### 1.1.2 Historical Context and Motivation

The development of language models reflects a fundamental challenge in artificial intelligence: enabling machines to understand and generate human language. This challenge encompasses multiple dimensions:

- **Syntactic understanding:** Parsing grammatical structure and relationships
- **Semantic comprehension:** Grasping meaning, including ambiguity and context-dependence
- **Pragmatic reasoning:** Understanding language use in context, including implicature and common sense
- **World knowledge:** Incorporating facts, relationships, and conceptual understanding
- **Generation capabilities:** Producing coherent, relevant, and contextually appropriate text

Early approaches to natural language processing relied heavily on hand-crafted rules and symbolic representations. The statistical revolution in the 1990s shifted focus to learning patterns from data, enabling more robust and scalable systems.

## 1.2 Evolution of Language Modeling Approaches

### 1.2.1 Statistical Language Models

#### N-gram Models

The earliest successful statistical language models were **n-gram models**, which approximate the probability of a word given its history by considering only the previous  $n - 1$  words:

$$P(w_i | w_1, \dots, w_{i-1}) \approx P(w_i | w_{i-n+1}, \dots, w_{i-1}) \quad (1.2)$$

For example, a trigram model ( $n=3$ ) assumes:

$$P(w_i | w_1, \dots, w_{i-1}) \approx P(w_i | w_{i-2}, w_{i-1}) \quad (1.3)$$

These probabilities are estimated from observed frequencies in training corpora using maximum likelihood estimation, often with smoothing techniques to handle unseen n-grams [4].

**Advantages:** Simplicity, interpretability, computational efficiency, and effectiveness for many traditional NLP tasks.

**Limitations:** Exponential growth in parameter count with context length, inability to capture long-range dependencies, data sparsity for longer n-grams, and lack of semantic generalization.

#### Class-based and Cache Models

Extensions to basic n-gram models included:

- **Class-based models:** Grouping words into classes to reduce sparsity and improve generalization
- **Cache models:** Maintaining a short-term cache of recently seen words to capture local discourse patterns
- **Skip-gram models:** Allowing gaps in the context window to capture longer-range patterns

### 1.2.2 Neural Language Models

#### Feedforward Neural Language Models

The seminal work of Bengio et al. (2003) [5] introduced neural language models that represent words as continuous vectors (embeddings) and use feedforward neural networks to predict the next word:

$$\mathbf{h} = \tanh(\mathbf{W}_1[\mathbf{e}_{w_{i-n+1}}; \dots; \mathbf{e}_{w_{i-1}}] + \mathbf{b}_1) \quad (1.4)$$

$$P(w_i | w_{i-n+1}, \dots, w_{i-1}) = \text{softmax}(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2) \quad (1.5)$$

where  $\mathbf{e}_w$  represents the embedding vector for word  $w$ , and  $[\cdot; \cdot]$  denotes concatenation.

**Key innovations:**

- Distributed representations allowing semantic similarity to be captured in vector space
- Automatic feature learning rather than manual feature engineering
- Better generalization to unseen word combinations through shared embeddings

#### Recurrent Neural Network Language Models

Recurrent Neural Networks (RNNs) addressed the fixed context window limitation by maintaining a hidden state that theoretically captures information from the entire history [6]:

$$\mathbf{h}_t = \sigma(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{e}_{w_t} + \mathbf{b}_h) \quad (1.6)$$

$$P(w_{t+1} | w_1, \dots, w_t) = \text{softmax}(\mathbf{W}_{yh}\mathbf{h}_t + \mathbf{b}_y) \quad (1.7)$$

**Long Short-Term Memory (LSTM)** networks [7] and **Gated Recurrent Units (GRUs)** [8] improved upon vanilla RNNs by introducing gating mechanisms that better preserve long-range dependencies and mitigate vanishing gradient problems.

LSTM language models achieved state-of-the-art performance throughout the 2010s and demonstrated the value of large-scale neural language modeling [9].

#### Limitations of RNN-based models:

- Sequential processing prevents parallelization during training
- Difficulty capturing very long-range dependencies despite gating mechanisms
- Gradient flow challenges in very deep networks

## 1.3 The Transformer Revolution

### 1.3.1 Attention Mechanisms

The **attention mechanism**, initially introduced for neural machine translation [10], allows models to dynamically focus on different parts of the input when producing each output. For a comprehensive treatment of attention mechanisms and their mathematical foundations, see [11, 12]. For a query  $\mathbf{q}$ , keys  $\mathbf{K}$ , and values  $\mathbf{V}$ :

$$\text{Attention}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \sum_i \frac{\exp(\mathbf{q}^\top \mathbf{k}_i)}{\sum_j \exp(\mathbf{q}^\top \mathbf{k}_j)} \mathbf{v}_i \quad (1.8)$$

This enables direct connections between any positions in a sequence, addressing the long-range dependency problem of RNNs.

### 1.3.2 Transformer Architecture

The Transformer architecture [11, 12] revolutionized natural language processing by replacing recurrence with self-attention, enabling:

- Parallel processing of all sequence positions
- Direct modeling of relationships between any two positions
- More efficient training on modern hardware
- Superior performance on a wide range of tasks

#### Key components:

1. **Multi-head self-attention:** Computing multiple attention patterns in parallel, allowing the model to attend to different aspects of the input simultaneously
2. **Position-wise feedforward networks:** Applying the same feedforward network to each position independently
3. **Positional encodings:** Adding position information since self-attention is permutation-invariant
4. **Layer normalization and residual connections:** Enabling training of deep networks
5. **Encoder-decoder structure:** Supporting sequence-to-sequence tasks

### 1.3.3 Impact and Adoption

The Transformer's introduction in 2017 rapidly transformed NLP research and applications:

- Achieved state-of-the-art results on machine translation and numerous other tasks
- Enabled effective scaling to billions of parameters
- Became the foundation for virtually all modern LLMs
- Extended beyond NLP to computer vision, multimodal learning, and other domains

## 1.4 The Pre-training and Fine-tuning Paradigm

### 1.4.1 Transfer Learning in NLP

Transfer learning—leveraging knowledge from one task to improve performance on another—became the dominant paradigm in NLP around 2018-2019. The typical workflow involves:

1. **Pre-training:** Training a model on a large corpus using self-supervised objectives
2. **Fine-tuning:** Adapting the pre-trained model to specific downstream tasks with (typically smaller) labeled datasets

This approach dramatically improved performance across a wide range of tasks and reduced the labeled data requirements for new applications.

### 1.4.2 Foundational Models

#### ELMo (Embeddings from Language Models)

ELMo [13] pioneered contextualized word representations by using bidirectional LSTM language models. Unlike static word embeddings (e.g., Word2Vec, GloVe), ELMo representations vary based on context, capturing polysemy and contextual nuance.

#### GPT (Generative Pre-trained Transformer)

The original GPT [14] demonstrated that pre-training a Transformer decoder on a language modeling objective, followed by task-specific fine-tuning, could achieve strong performance across diverse NLP tasks. The unidirectional (left-to-right) architecture made it particularly well-suited for generation tasks.

#### BERT (Bidirectional Encoder Representations from Transformers)

BERT [15] achieved breakthrough results by pre-training a bidirectional Transformer encoder using two objectives:

- **Masked Language Modeling (MLM):** Randomly masking tokens and predicting them from bidirectional context
- **Next Sentence Prediction (NSP):** Predicting whether two segments follow each other in the original text

BERT’s bidirectional context enabled superior performance on understanding-focused tasks, establishing new state-of-the-art results on numerous benchmarks including GLUE, SQuAD, and others.

### 1.4.3 The Scaling Era Begins

Following BERT and GPT, the field witnessed rapid scaling in model size and training data:

- GPT-2 (2019) [16]: 1.5B parameters, demonstrated impressive zero-shot capabilities
- T5 (2019) [17]: Unified framework treating all tasks as text-to-text
- RoBERTa, ALBERT, ELECTRA: Optimization and efficiency improvements
- GPT-3 (2020) [1]: 175B parameters, few-shot learning breakthrough

These developments established that scaling model size, data, and compute often led to qualitative improvements in capabilities, setting the stage for the modern era of large language models.

Figure 1.1 illustrates the dramatic growth in model scale from 2018 to 2024, showing how parameter counts have increased by over four orders of magnitude in just six years.

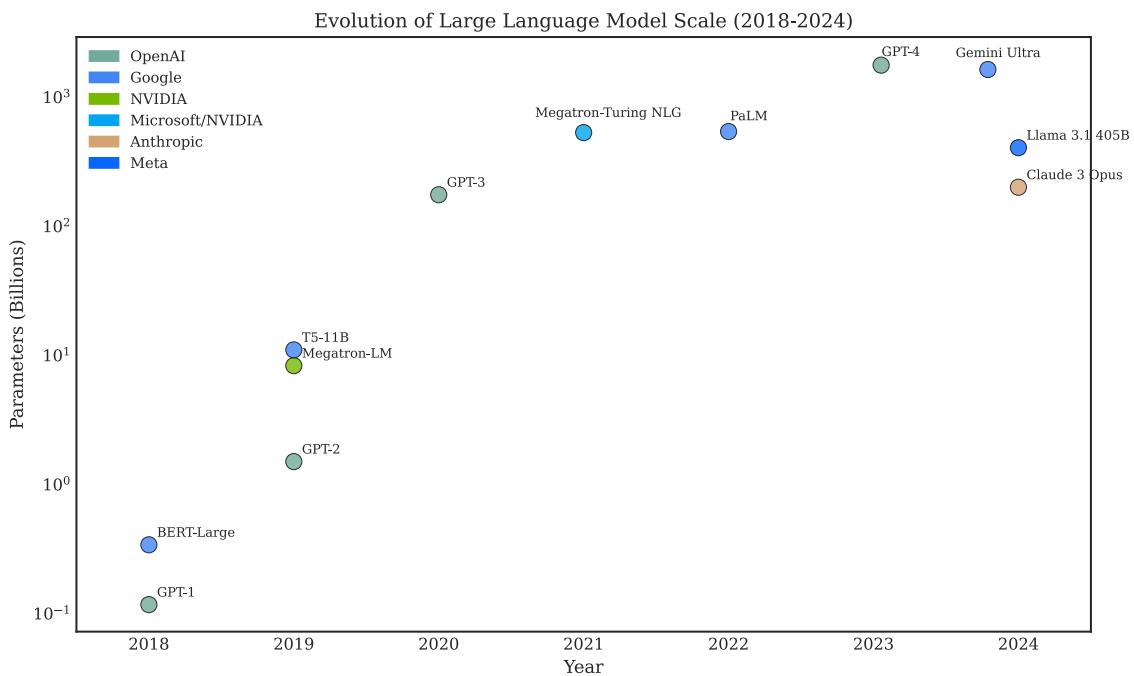


Figure 1.1: Evolution of large language model scale from 2018 to 2024. The logarithmic y-axis reveals exponential growth in model parameters, from GPT-1’s 117 million parameters to models exceeding one trillion parameters. Different colors indicate the developing organization, highlighting the competitive landscape among major AI laboratories [1, 18–20].



# Chapter 2

## Core architectures and mechanisms

### 2.1 The Transformer Architecture in Depth

#### 2.1.1 Self-Attention Mechanism

The self-attention mechanism is the core computational building block of Transformers. For an input sequence  $\mathbf{X} \in \mathbb{R}^{n \times d}$  where  $n$  is sequence length and  $d$  is dimension:

##### Scaled Dot-Product Attention

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \quad (2.1)$$

where:

- $\mathbf{Q} = \mathbf{X}\mathbf{W}^Q$  (queries): What each position is looking for
- $\mathbf{K} = \mathbf{X}\mathbf{W}^K$  (keys): What each position offers
- $\mathbf{V} = \mathbf{X}\mathbf{W}^V$  (values): What each position contributes
- $d_k$ : Dimension of keys (scaling factor prevents softmax saturation)

The attention weights  $\mathbf{A} = \text{softmax}(\mathbf{Q}\mathbf{K}^\top/\sqrt{d_k})$  indicate how much each position attends to every other position.

##### Multi-Head Attention

Rather than computing a single attention function, multi-head attention computes  $h$  attention functions in parallel:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\text{head}_1; \dots; \text{head}_h]\mathbf{W}^O \quad (2.2)$$

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \quad (2.3)$$

**Benefits:**

- Different heads can attend to different aspects of the input (syntax, semantics, position, etc.)
- Increases model expressiveness while maintaining computational efficiency
- Empirically improves performance across tasks

#### 2.1.2 Position-wise Feed-Forward Networks

After attention, each position is processed independently through a two-layer feedforward network with ReLU activation:

$$\text{FFN}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 \quad (2.4)$$

Typically, the hidden dimension is  $4 \times$  the model dimension, providing substantial capacity for transformation.

### 2.1.3 Positional Encoding

Since self-attention is permutation-invariant, positional information must be explicitly injected. The original Transformer used sinusoidal positional encodings:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d}) \quad (2.5)$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d}) \quad (2.6)$$

Modern LLMs often use learned positional embeddings or more sophisticated schemes like RoPE (Rotary Positional Embeddings) or ALiBi (Attention with Linear Biases).

### 2.1.4 Layer Normalization and Residual Connections

Each sub-layer (attention, FFN) is wrapped with:

$$\text{Output} = \text{LayerNorm}(\mathbf{x} + \text{Sublayer}(\mathbf{x})) \quad (2.7)$$

**Layer normalization** normalizes activations across features for each example, stabilizing training.

**Residual connections** allow gradients to flow directly through the network, enabling training of very deep models (modern LLMs have 50-100+ layers).

### 2.1.5 Causal Masking for Language Modeling

For autoregressive language modeling, attention must be **causally masked** to prevent positions from attending to future positions:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} + \mathbf{M} \right) \mathbf{V} \quad (2.8)$$

where  $\mathbf{M}_{ij} = 0$  if  $i \geq j$  and  $-\infty$  otherwise, ensuring that position  $i$  can only attend to positions  $\leq i$ .

## 2.2 Architecture Variants

### 2.2.1 Encoder-Only Models

Models like BERT use only the Transformer encoder with bidirectional attention, optimized for understanding tasks:

- Classification (sentiment analysis, NER)
- Question answering
- Sentence similarity
- Information extraction

**Training objective:** Masked Language Modeling (MLM), predicting masked tokens from bidirectional context.

### 2.2.2 Decoder-Only Models

Models like GPT use only the Transformer decoder with causal attention, optimized for generation:

- Text generation
- Code synthesis
- Dialogue
- Few-shot learning

**Training objective:** Autoregressive language modeling, predicting the next token given previous tokens.

**Advantages:**

- Simpler architecture
- Natural generation capability
- Effective for in-context learning
- Easier to scale to very large sizes

Most modern LLMs (GPT-3/4, LLaMA, Claude, etc.) use decoder-only architectures.

### 2.2.3 Encoder-Decoder Models

Models like T5 and BART use both encoder and decoder, optimized for sequence-to-sequence tasks:

- Machine translation
- Summarization
- Question answering (generative)
- Text-to-text reformulation

The encoder processes input with bidirectional attention, while the decoder generates output autoregressively with causal attention plus cross-attention to encoder states.

## 2.3 Scaling Laws and Emergent Properties

### 2.3.1 Neural Scaling Laws

Empirical research has revealed predictable relationships between model performance and scale [21, 22]:

$$L(N, D, C) \approx \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{D_c}{D}\right)^{\alpha_D} + \left(\frac{C_c}{C}\right)^{\alpha_C} \quad (2.9)$$

where  $L$  is loss,  $N$  is number of parameters,  $D$  is dataset size,  $C$  is compute budget, and the constants are empirically determined.

**Key findings:**

- Performance improves smoothly and predictably with scale
- Optimal compute allocation requires balancing model size and training data (Chinchilla scaling laws)
- Larger models are more sample-efficient
- Returns diminish but don't plateau within explored ranges

Figure 2.1 visualizes these scaling relationships, demonstrating how loss decreases predictably with increasing model parameters under different training regimes.

### 2.3.2 Emergent Abilities

**Emergent abilities** are capabilities that appear suddenly at certain scales rather than improving smoothly [2]:

- **Few-shot learning:** Learning from just a few examples in the prompt
- **Chain-of-thought reasoning:** Solving complex problems through step-by-step reasoning
- **Instruction following:** Generalizing to unseen instruction types
- **Arithmetic and symbolic manipulation:** Mathematical and logical operations
- **Multi-step reasoning:** Combining multiple reasoning steps

The mechanisms underlying emergence are actively debated—whether they reflect true phase transitions or artifacts of evaluation metrics.

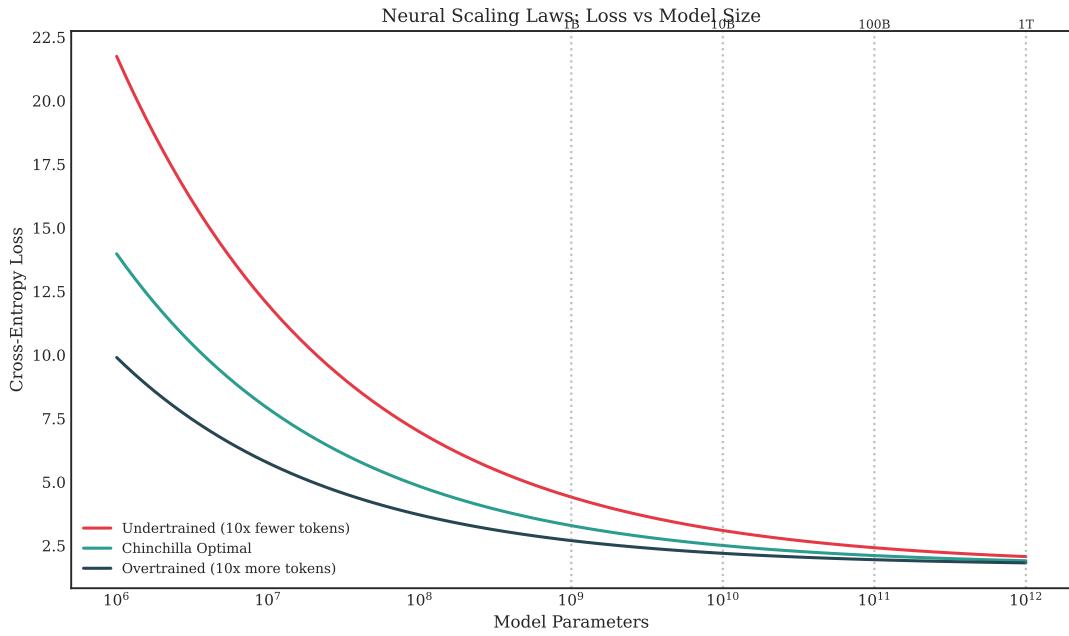


Figure 2.1: Neural scaling laws showing the relationship between model parameters and cross-entropy loss. **Illustrative visualization** based on the mathematical formulas from the Chinchilla scaling laws paper, using the published coefficients ( $\alpha = 0.34$ ,  $\beta = 0.28$ ). The Chinchilla-optimal curve represents the ideal balance between model size and training tokens, while undertrained and overtrained curves show suboptimal allocation of compute budget [21, 22].

### 2.3.3 In-Context Learning

Large language models can perform tasks by conditioning on examples provided in the prompt, without gradient updates:

- **Zero-shot:** Task description only
- **Few-shot:** Task description plus a few examples
- **Many-shot:** Dozens to hundreds of examples

In-context learning represents a fundamental shift from traditional supervised learning, enabling rapid task adaptation without retraining.

## 2.4 Attention Mechanism Variations

This section examines key variations and optimizations of attention mechanisms that have enabled practical deployment of large language models. For a detailed mathematical treatment and survey of attention architectures, including their computational properties and applications across NLP, computer vision, and multimodal tasks, see [11, 12].

### 2.4.1 Sparse Attention Patterns

Standard self-attention has  $O(n^2)$  complexity in sequence length, limiting context windows (see Figure 2.2 for a comparison of attention mechanisms). Sparse attention patterns reduce this:

#### Local Attention

Each position attends only to a local window, reducing complexity to  $O(n)$  but losing global information.

#### Strided/Dilated Attention

Attending to every  $k$ -th position to capture longer-range patterns efficiently.

### Global-Local Attention

Combining global attention for special tokens with local attention for others (e.g., Longformer [23]).

### Random Attention

Random attention patterns that probabilistically preserve expressiveness (e.g., BigBird [24]).

## 2.4.2 Efficient Attention Mechanisms

### Linear Attention

Reformulating attention to avoid explicit computation of the  $n \times n$  attention matrix, achieving  $O(n)$  complexity through kernel methods or low-rank approximations.

### Flash Attention

Optimizing attention computation through IO-aware algorithms that minimize memory movement between GPU memory hierarchies [25], enabling longer contexts and faster training.

## 2.4.3 Rotary Position Embeddings (RoPE)

RoPE [26] encodes position information by rotating query and key vectors in complex space:

$$\mathbf{q}_m = \mathbf{R}_m \mathbf{W}^Q \mathbf{x}_m, \quad \mathbf{k}_n = \mathbf{R}_n \mathbf{W}^K \mathbf{x}_n \quad (2.10)$$

where  $\mathbf{R}_m$  is a rotation matrix encoding position  $m$ . This has several advantages:

- Naturally encodes relative positions in the attention score
- Extrapolates better to longer sequences than absolute encodings
- No additional parameters beyond the standard attention

Used in LLaMA, PaLM, and other modern LLMs.

## 2.4.4 Grouped-Query Attention (GQA)

GQA [27] shares key and value projections across multiple query heads:

- Reduces KV cache memory requirements during inference
- Maintains most of the quality of full multi-head attention
- Interpolates between multi-head attention (each head has own K,V) and multi-query attention (all heads share K,V)

Enables more efficient serving of large models.

[Figure 2.2](#) compares the computational complexity of various attention mechanisms, illustrating how different approaches trade off between efficiency and expressiveness.

## 2.5 Tokenization

### 2.5.1 Subword Tokenization Methods

Tokenization converting text to discrete tokens significantly impacts model performance and efficiency.

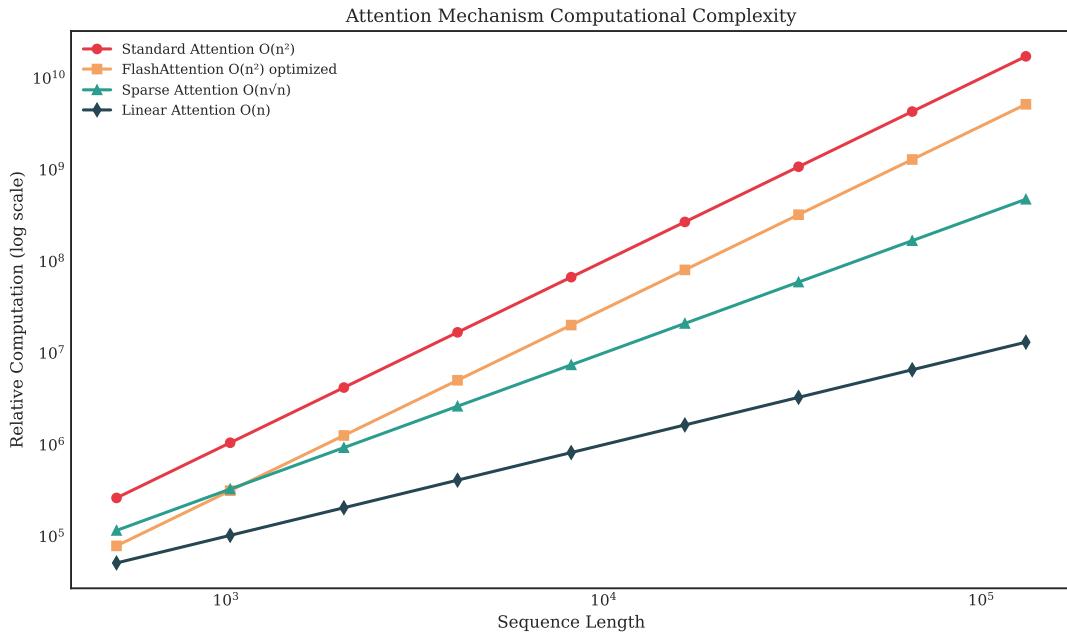


Figure 2.2: Computational complexity comparison of attention mechanism variants (**theoretical analysis**). Standard self-attention scales quadratically with sequence length  $O(n^2)$  [11, 12], while sparse attention methods like Longformer [23] and BigBird [24] achieve reduced complexity. FlashAttention [25, 28] maintains full attention capability while optimizing memory access patterns for practical speedups. Curves show asymptotic complexity classes with illustrative scaling constants for visualization.

### Byte-Pair Encoding (BPE)

BPE [29] iteratively merges the most frequent character pairs:

1. Start with character vocabulary
2. Repeat: Find most frequent pair, add merge rule
3. Stop after desired vocabulary size

**Advantages:** Handles out-of-vocabulary words, balances vocabulary size and sequence length, purely data-driven.

### WordPiece

Similar to BPE but selects merges that maximize likelihood on training data rather than frequency. Used by BERT and other Google models.

### SentencePiece

Treats text as raw Unicode, enabling language-agnostic tokenization without requiring pre-tokenization. Used by T5, LLaMA, and many multilingual models.

#### 2.5.2 Vocabulary Size Trade-offs

- **Small vocabulary:** Longer sequences, more computational cost, better handling of morphology
- **Large vocabulary:** Shorter sequences, faster processing, potential out-of-vocabulary issues
- **Typical sizes:** 32k-100k tokens for most modern LLMs

### 2.5.3 Impact on Model Behavior

Tokenization affects:

- Multilingual performance (overrepresentation of English in training leads to overrepresentation in tokenizers)
- Arithmetic and structured text handling (numbers, code, etc.)
- Prompting strategies and token limits
- Bias and fairness (different tokenization across languages/dialects)



# Chapter 3

## Major models and families

### 3.1 GPT Series

#### 3.1.1 GPT-1

The original Generative Pre-trained Transformer [14]:

- 117M parameters
- 12-layer decoder-only Transformer
- Pre-trained on BooksCorpus (7,000 books)
- Demonstrated transfer learning effectiveness for NLP

#### 3.1.2 GPT-2

GPT-2 [16] scaled up the approach:

- 1.5B parameters (largest variant)
- Trained on WebText (40GB, 8M documents)
- Demonstrated strong zero-shot capabilities
- Initially withheld due to concerns about misuse
- Showed coherent long-form generation

#### 3.1.3 GPT-3

GPT-3 [1] represented a major scaling breakthrough:

##### Architecture:

- 175B parameters (96 layers, 12,288 hidden dimensions, 96 attention heads)
- 2048 token context window
- Trained on 300B tokens from diverse sources (Common Crawl, WebText2, Books, Wikipedia)

##### Key capabilities:

- Few-shot learning across diverse tasks without fine-tuning
- Coherent multi-paragraph text generation
- Basic arithmetic and logical reasoning
- Code generation (Codex variant trained on GitHub code)
- Translation, summarization, question answering

**Variants:**

- **Codex**: Fine-tuned on code, powering GitHub Copilot
- **InstructGPT**: Fine-tuned with RLHF for instruction following

### 3.1.4 GPT-4

GPT-4 [30] (released March 2023):

**Capabilities** (see [Figure 7.3](#) for benchmark comparisons):

- Multimodal (accepts image and text inputs)
- Significantly improved reasoning and factual accuracy
- Enhanced safety and alignment
- 128k token context window (extended version)
- Strong performance on professional exams (bar exam, medical exams, etc.)

**Architecture details**: Not publicly disclosed, but generally believed to be a larger-scale model with improved training methods.

**Notable achievements**:

- 90th percentile on Uniform Bar Exam
- 5 score on several AP exams
- Strong multilingual capabilities
- Advanced coding abilities

### 3.1.5 GPT-4 Turbo and GPT-4o

Subsequent releases improved efficiency, context length, and multimodal capabilities:

- Longer context windows (128k tokens standard)
- Faster inference
- Lower cost
- Enhanced vision understanding
- GPT-4o: Truly integrated text, vision, and audio processing

## 3.2 BERT and Encoder Models

### 3.2.1 BERT

BERT [15] revolutionized NLP understanding tasks:

**Architecture**:

- BERT-Base: 110M parameters (12 layers, 768 hidden, 12 heads)
- BERT-Large: 340M parameters (24 layers, 1024 hidden, 16 heads)
- Encoder-only Transformer with bidirectional attention

**Pre-training**:

- Masked Language Modeling (MLM): 15% of tokens masked, model predicts them
- Next Sentence Prediction (NSP): Binary classification whether sentences are consecutive
- Trained on BooksCorpus and English Wikipedia (3.3B tokens)

### **Impact:**

- Established new SOTA on 11 NLP tasks
- Spawnsed numerous variants and improvements
- Became standard baseline for understanding tasks

## 3.2.2 BERT Variants

### **RoBERTa**

RoBERTa [31] optimized BERT training:

- Removed NSP objective (not beneficial)
- Dynamic masking instead of static
- Larger batches and more data
- Longer training
- Result: Consistent improvements over BERT

### **ALBERT**

ALBERT [32] improved parameter efficiency:

- Factorized embedding parameterization
- Cross-layer parameter sharing
- Sentence-order prediction (SOP) instead of NSP
- Achieved better performance with fewer parameters

### **ELECTRA**

ELECTRA [33] introduced a more efficient pre-training objective:

- Generator-discriminator setup
- Generator: Small MLM model replacing tokens
- Discriminator: Detects which tokens were replaced
- Learns from all tokens rather than just masked ones
- More sample-efficient than MLM

### **DeBERTa**

DeBERTa [34] enhanced attention mechanisms:

- Disentangled attention: Separate attention for content and position
- Enhanced mask decoder for absolute position in fine-tuning
- Virtual adversarial training for robustness
- Achieved strong results on SuperGLUE

### 3.3 T5 and Encoder-Decoder Models

#### 3.3.1 T5 (Text-to-Text Transfer Transformer)

T5 [17] unified all NLP tasks as text-to-text:

**Architecture:**

- Standard encoder-decoder Transformer
- Sizes from 60M (Small) to 11B (XXL) parameters
- Relative position embeddings

**Unified framework:**

- All tasks reformulated as text generation
- Task specified by text prefix (e.g., "translate English to German:")
- Single model handles classification, generation, QA, etc.

**C4 dataset:** Colossal Clean Crawled Corpus (750GB of cleaned Common Crawl)

**Systematic study:** Explored various design choices (objectives, architectures, datasets, fine-tuning strategies)

#### 3.3.2 BART

BART [35] combined ideas from BERT and GPT:

- Encoder-decoder architecture
- Pre-trained by corrupting text and learning to reconstruct it
- Corruption schemes: token masking, deletion, sentence permutation, document rotation
- Particularly effective for generation tasks

#### 3.3.3 UL2

UL2 [36] unified different pre-training paradigms:

- Combined causal language modeling, prefix language modeling, and span corruption
- Mode switching allows single model to handle different task types
- Improved versatility across diverse applications

### 3.4 LLaMA and Open Models

#### 3.4.1 LLaMA

LLaMA [37] (Large Language Model Meta AI) emphasized efficiency and openness:

**LLaMA 1:**

- Sizes: 7B, 13B, 33B, 65B parameters
- Trained on publicly available data (1.4T tokens)
- Competitive with much larger proprietary models
- Released to research community (though weights leaked publicly)

**Architecture choices:**

- Pre-normalization using RMSNorm
- SwiGLU activation function
- Rotary positional embeddings (RoPE)
- Efficient attention implementation

### 3.4.2 LLaMA 2

LLaMA 2 [38] improved and openly released:

- Sizes: 7B, 13B, 70B parameters
- Trained on 2T tokens
- Commercial use allowed
- LLaMA 2-Chat: RLHF-tuned for dialogue
- Detailed documentation of safety procedures

### 3.4.3 LLaMA 3

LLaMA 3 (2024) further advanced capabilities:

- Improved tokenizer (128k vocabulary)
- Longer training on higher-quality data
- Enhanced instruction following and safety
- Multiple size variants

### 3.4.4 Open Model Ecosystem

LLaMA catalyzed an open-source LLM ecosystem:

#### Alpaca

- Fine-tuned LLaMA 7B on instruction-following data
- Demonstrated effectiveness of smaller instruction-tuned models
- Generated training data using GPT-3.5

#### Vicuna

- Fine-tuned on user-shared ChatGPT conversations
- Strong performance on diverse tasks
- Open evaluation frameworks (MT-Bench, Chatbot Arena)

#### MPT

MosaicML's MPT (MosaicML Pretrained Transformer) series:

- Commercially usable
- Variants optimized for different context lengths and tasks
- Emphasis on training efficiency

### 3.5 Claude

Claude (Anthropic) emphasizes safety and harmlessness:

**Claude 1 and 2:**

- Constitutional AI training methodology
- Strong emphasis on harmlessness alongside helpfulness
- Extended context windows (100k tokens for Claude 2)
- Reduced harmful outputs

**Claude 3 Family** (see [Figure 7.4](#) for capability comparison):

- **Claude 3 Haiku:** Fastest and most compact
- **Claude 3 Sonnet:** Balanced performance and speed
- **Claude 3 Opus:** Most capable, matches or exceeds GPT-4 on many benchmarks
- Multimodal capabilities (vision)
- 200k token context window
- Improved reasoning and coding

**Claude 3.5 Sonnet:**

- Surpasses Claude 3 Opus on many benchmarks
- Enhanced coding and visual reasoning
- Maintains speed advantages

**Distinguishing features:**

- Constitutional AI and RLAIF (RL from AI Feedback)
- Careful attention to safety and alignment
- Strong performance on nuanced tasks requiring instruction following

### 3.6 PaLM and Gemini

#### 3.6.1 PaLM (Pathways Language Model)

PaLM [39] from Google demonstrated scaling efficiency:

- 540B parameters
- Trained using Pathways system for efficient scaling
- SwiGLU activation, RoPE embeddings
- Multi-query attention
- Strong performance on reasoning benchmarks

**PaLM 2:**

- Improved data quality and mixture
- Enhanced multilingual and reasoning capabilities
- More efficient architecture
- Powers various Google products

### 3.6.2 Gemini

Google's Gemini represents a natively multimodal approach:

**Gemini 1.0:**

- **Gemini Nano:** On-device applications
- **Gemini Pro:** Balanced performance for wide range of tasks
- **Gemini Ultra:** Most capable, competitive with GPT-4
- Native multimodal training (not separate vision and language modules)

**Gemini 1.5:**

- Mixture-of-Experts architecture for efficiency
- 1M token context window (experimental), representing a major advance in context length (see [Figure 21.1](#))
- Enhanced long-context understanding
- Strong performance across modalities

## 3.7 Specialized and Domain-Specific Models

### 3.7.1 Code Models

#### Codex

Fine-tuned GPT-3 for code generation (see [Figure 7.2](#) for code generation benchmark results):

- Trained on GitHub code
- Powers GitHub Copilot
- Strong Python performance, supports many languages

#### AlphaCode

DeepMind's competitive programming model [40]:

- Specialized for algorithmic problem solving
- Performs at median competitive programmer level
- Large-scale sampling and filtering

#### StarCoder

Open-source code generation model:

- 15B parameters
- Trained on The Stack dataset (permissively licensed code)
- Fill-in-the-middle capability
- Supports 80+ programming languages

### 3.7.2 Scientific and Biomedical Models

#### Galactica

Meta's scientific knowledge model [41]:

- Trained on scientific corpus (48M papers, textbooks, knowledge bases)
- Specialized for scientific reasoning and knowledge
- Controversial due to hallucination concerns

**BioGPT, Med-PaLM**

Models specialized for biomedical and clinical applications:

- Fine-tuned on medical literature and data
- Improved performance on medical QA and reasoning
- Important for healthcare applications with appropriate safeguards

**Molecular and Cellular Language Models**

Recent work has extended language model architectures to biological domains, treating molecular and cellular processes as specialized languages. Hierarchical Molecular Language Models [42] demonstrate how transformer-based architectures can model cellular signaling networks, where signaling molecules function as tokens, protein interactions define syntax, and functional consequences represent semantics. This approach enables integration of multi-modal biological data across different scales and opens new avenues for drug discovery and systems biology research.

**3.7.3 Multilingual Models****mBERT, XLM-R**

Cross-lingual understanding models:

- Trained on 100+ languages
- Enable zero-shot cross-lingual transfer
- Foundation for multilingual NLP applications

**BLOOM**

BigScience Large Open-science Open-access Multilingual model:

- 176B parameters
- Trained on 46 natural languages and 13 programming languages
- Openly available
- Collaborative international effort

**3.8 Mixture-of-Experts Models****3.8.1 Architectural Approach**

Mixture-of-Experts (MoE) architectures activate only a subset of parameters per token:

$$\text{MoE}(\mathbf{x}) = \sum_{i=1}^N g_i(\mathbf{x}) E_i(\mathbf{x}) \quad (3.1)$$

where  $E_i$  are expert networks and  $g_i$  are gating functions determining expert weights.

**Advantages:**

- Higher capacity with similar computational cost
- Specialization of experts for different input types
- Improved training efficiency

**Challenges:**

- Load balancing across experts
- Training instability
- Increased memory requirements

### 3.8.2 Switch Transformer

Switch Transformer [43] simplified MoE:

- Route each token to single expert (hard routing)
- 1.6T parameters with competitive computational cost
- Improved training stability through load balancing

### 3.8.3 GLaM

Google's Generalist Language Model [44]:

- 1.2T parameters, activates 97B per token
- Outperforms GPT-3 with less training cost
- Demonstrates efficiency of sparse models

### 3.8.4 Mixtral

Mistral AI's open MoE model:

- 8 experts, 2 active per token
- 47B total parameters, 13B active
- Strong performance competitive with much larger dense models
- Commercially usable



# Chapter 4

## Training methodologies

### 4.1 Pre-training Objectives

#### 4.1.1 Causal Language Modeling

The standard objective for decoder-only models:

$$\mathcal{L}_{\text{CLM}} = - \sum_{i=1}^n \log P(w_i | w_1, \dots, w_{i-1}; \theta) \quad (4.1)$$

**Advantages:**

- Simple, unsupervised objective
- Natural generation capability
- Effective for in-context learning
- Scales well to large models and datasets

#### 4.1.2 Masked Language Modeling

Used by BERT and encoder models:

$$\mathcal{L}_{\text{MLM}} = - \sum_{i \in \mathcal{M}} \log P(w_i | w_{\setminus \mathcal{M}}; \theta) \quad (4.2)$$

where  $\mathcal{M}$  is the set of masked positions.

**Masking strategy:**

- 15% of tokens selected for masking
- 80%: Replace with [MASK]
- 10%: Replace with random token
- 10%: Keep original

#### 4.1.3 Prefix Language Modeling

Combines causal and masked objectives:

- Prefix portion visible bidirectionally
- Continuation predicted autoregressively
- Balances understanding and generation

#### 4.1.4 Span Corruption

Used by T5 and similar models:

- Corrupt random spans of tokens
- Replace each span with sentinel token
- Model predicts corrupted spans
- Encourages learning longer-range dependencies

### 4.2 Training Data and Curation

#### 4.2.1 Data Sources

Modern LLMs train on diverse corpora:

##### Web Data

- Common Crawl: Petabytes of web pages
- Filtered and cleaned versions (C4, RefinedWeb, etc.)
- Quality filtering critical to avoid toxic/low-quality content

##### Books

- Long-form coherent text
- Narrative and argumentative structure
- Higher quality writing on average

##### Code

- GitHub and other code repositories
- Structured reasoning and logical patterns
- Improves problem-solving capabilities

##### Academic and Scientific

- Research papers, preprints
- Specialized domain knowledge
- Formal reasoning patterns

##### Curated Datasets

- Wikipedia: High-quality factual knowledge
- Stack Exchange: Technical Q&A
- News: Current events and diverse topics

### 4.2.2 Data Quality and Filtering

#### Deduplication:

- Remove exact and near-duplicate documents
- Prevents memorization and training inefficiency
- Important for reliable evaluation

#### Quality filtering:

- Classifier-based filtering (using high-quality references)
- Heuristic rules (length, formatting, language detection)
- Removing toxic, biased, or harmful content
- Balancing quality vs. diversity

#### Data mixture:

- Balancing different sources and domains
- Upweighting high-quality sources
- Ensuring multilingual representation
- Trade-offs between generality and specialization

### 4.2.3 Data Scaling Considerations

**Chinchilla scaling laws** [22]: For compute-optimal training, model size and training tokens should scale proportionally (see Figure 2.1):

- Previous models (GPT-3) were undertrained for their size
- Optimal training uses roughly 20 tokens per parameter
- Smaller models trained longer can outperform larger undertrained models

## 4.3 Optimization and Training Techniques

### 4.3.1 Optimizers

#### Adam and Variants

Adam [45] is standard for training Transformers:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (4.3)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad (4.4)$$

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t \quad (4.5)$$

**AdamW**: Decoupled weight decay, improving generalization.

**Adafactor**: Memory-efficient variant for very large models.

### 4.3.2 Learning Rate Schedules

#### Warmup

Linear or exponential warmup prevents instability in early training:

- Start with very small learning rate
- Gradually increase to maximum over first few thousand steps
- Critical for stable training of large models

### Decay

After warmup, learning rate typically decays:

- Linear decay
- Cosine decay
- Inverse square root decay

### 4.3.3 Gradient Clipping and Stability

**Gradient norm clipping:**

$$\mathbf{g} \leftarrow \min \left( 1, \frac{\tau}{\|\mathbf{g}\|} \right) \mathbf{g} \quad (4.6)$$

Prevents gradient explosions and training instability.

### 4.3.4 Mixed Precision Training

Using FP16 or BF16 instead of FP32:

- Reduces memory usage by  $2\times$
- Accelerates computation on modern GPUs/TPUs
- Requires careful handling of numerical precision
- Loss scaling prevents underflow in FP16
- BF16 (bfloat16) maintains FP32 dynamic range, simpler to use

### 4.3.5 Batch Size and Accumulation

**Large batch training:**

- Improves hardware utilization
- Enables better parallelization
- May require learning rate scaling

**Gradient accumulation:** Simulates large batches when memory is limited:

- Accumulate gradients over multiple microbatches
- Update parameters after desired effective batch size
- Critical for training large models on limited hardware

## 4.4 Distributed Training and Parallelism

### 4.4.1 Data Parallelism

Replicate model across devices, each processing different data:

- Simple to implement
- Scales well for moderate model sizes
- Limited by model memory fitting on single device

### 4.4.2 Model Parallelism

Split model across devices:

### Pipeline Parallelism

- Partition layers across devices
- Process microbatches in pipeline fashion
- Reduces pipeline bubbles through scheduling
- Examples: GPipe, PipeDream

### Tensor Parallelism

- Partition individual layers/tensors across devices
- Requires communication within layer
- High communication overhead, needs fast interconnect
- Megatron-LM pioneered for Transformers

#### 4.4.3 Zero Redundancy Optimizer (ZeRO)

ZeRO [46] partitions optimizer state, gradients, and parameters:

- **ZeRO-1:** Partition optimizer states
- **ZeRO-2:** Partition optimizer states and gradients
- **ZeRO-3:** Partition optimizer states, gradients, and parameters

Enables training models that don't fit on single device memory while maintaining data parallelism simplicity.

#### 4.4.4 3D Parallelism

Combining data, pipeline, and tensor parallelism:

- Optimal configuration depends on model size, hardware, and interconnect
- Megatron-DeepSpeed combines techniques for trillion-parameter models
- Requires careful tuning of parallelism dimensions

### 4.5 Fine-tuning Techniques

#### 4.5.1 Full Fine-tuning

Update all model parameters on task-specific data:

- **Advantages:** Maximum task performance, full model adaptation
- **Disadvantages:** Expensive, requires significant data, separate model per task

#### 4.5.2 Parameter-Efficient Fine-Tuning (PEFT)

Update only a small fraction of parameters:

### Low-Rank Adaptation (LoRA)

Add trainable low-rank matrices to weight updates [47]:

$$\mathbf{W}' = \mathbf{W} + \mathbf{BA} \quad (4.7)$$

where  $\mathbf{B} \in \mathbb{R}^{d \times r}$  and  $\mathbf{A} \in \mathbb{R}^{r \times k}$  with  $r \ll \min(d, k)$ .

**Advantages:**

- Train only 0.1-1% of parameters
- No inference latency increase (merge at deployment)
- Multiple task adapters can share base model
- Much lower memory and storage requirements

### Adapter Layers

Insert small trainable modules between frozen layers:

- Bottleneck architecture (down-project, activate, up-project)
- Train only adapters, freeze backbone
- Small inference overhead unless carefully optimized

### Prefix Tuning

Prepend trainable vectors to each layer:

- Virtual tokens that condition model behavior
- No change to model weights
- Competitive with full fine-tuning on many tasks

### Prompt Tuning

Learn continuous prompts rather than discrete tokens:

- Effective for large models
- Minimal parameters (just prompt embeddings)
- Less effective for smaller models

## 4.6 Instruction Tuning

### 4.6.1 Motivation and Approach

While pre-trained models have broad capabilities, they don't naturally follow instructions. **Instruction tuning** fine-tunes models on diverse tasks formatted as instructions [48].

**Data format:**

Instruction: Translate the following to French

Input: Hello, how are you?

Output: Bonjour, comment allez-vous?

### 4.6.2 Instruction Datasets

#### FLAN (Finetuned Language Net)

- 60+ NLP tasks converted to instruction format
- Template diversity for generalization
- Dramatic improvement in zero-shot task performance

**T0, P3**

- PromptSource templates for diverse datasets
- Multitask prompted training
- Strong zero-shot generalization

**Super-Natural Instructions**

- 1,600+ tasks with instructions
- Broad coverage of capabilities
- Enables strong instruction following

**4.6.3 Self-Instruct and Data Generation**

Using LLMs to generate their own instruction data [49]:

1. Start with small seed set of instructions
2. Use LLM to generate new instructions and responses
3. Filter for quality and diversity
4. Fine-tune on generated data

Enables rapid scaling of instruction datasets without extensive human annotation.

**4.7 Reinforcement Learning from Human Feedback (RLHF)****4.7.1 Motivation**

Standard language modeling objectives don't directly optimize for human preferences:

- Helpfulness
- Harmlessness
- Honesty
- Avoiding toxic or biased outputs

RLHF aligns models with human values through reinforcement learning.

**4.7.2 RLHF Pipeline****Step 1: Supervised Fine-tuning (SFT)**

Fine-tune pre-trained model on high-quality demonstrations:

- Human-written responses to prompts
- Establishes baseline instruction-following capability
- Typically thousands to tens of thousands of examples

**Step 2: Reward Model Training**

Train a model to predict human preferences:

1. Present prompt to SFT model, sample multiple responses
2. Humans rank responses (e.g., best to worst)
3. Train reward model to predict rankings:

$$\mathcal{L}_{RM} = -\mathbb{E}_{(x,y_w,y_l)} \log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l)) \quad (4.8)$$

where  $y_w$  is preferred over  $y_l$

**Step 3: Reinforcement Learning**

Optimize policy using reward model with PPO (Proximal Policy Optimization):

$$\mathcal{L}_{\text{RL}} = \mathbb{E}_{x,y}[r_\phi(x, y) - \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{SFT}}(y|x)}] \quad (4.9)$$

The KL penalty term ( $\beta \log \pi_\theta / \pi_{\text{SFT}}$ ) prevents model from diverging too far from SFT initialization, maintaining coherence and avoiding reward hacking.

**4.7.3 InstructGPT and ChatGPT**

InstructGPT [50] pioneered RLHF at scale:

- Dramatically improved alignment with user intent
- Reduced harmful outputs
- Better instruction following despite smaller size than GPT-3
- Foundation for ChatGPT

ChatGPT extended this to conversational settings with remarkable success.

**4.7.4 Challenges and Limitations**

- **Reward hacking:** Model exploits reward model weaknesses
- **Human feedback quality:** Noise, subjectivity, annotator disagreement
- **Scalability:** Expensive to collect sufficient human preferences
- **Value alignment:** Whose values should be optimized?
- **Distribution shift:** Reward model may not generalize to new responses

**4.8 Constitutional AI and RLAIF****4.8.1 Constitutional AI**

Anthropic's Constitutional AI [51] reduces reliance on human feedback:

**Supervised Stage**

1. Model generates responses to prompts
2. Model critiques own responses based on constitutional principles
3. Model revises responses to better align with principles
4. Fine-tune on revised responses

**RL Stage**

Use AI feedback instead of human feedback:

1. Model generates pairs of responses
2. AI evaluator ranks them based on constitutional principles
3. Train reward model on AI preferences
4. Apply RL as in RLHF

**Advantages:**

- Scalable: Less human labor required

- Transparent: Principles explicitly specified
- Flexible: Easy to update principles
- Reduces harmful outputs while maintaining helpfulness

#### 4.8.2 Direct Preference Optimization (DPO)

DPO [52] simplifies RLHF by directly optimizing preferences:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(x, y_w, y_l)} \log \sigma \left( \beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \right) \quad (4.10)$$

**Benefits:**

- No separate reward model needed
- More stable training
- Simpler implementation
- Competitive or better results than PPO-based RLHF



# Chapter 5

## Prompt engineering and interaction design

Prompt engineering has emerged as a critical discipline for effectively utilizing large language models. Rather than modifying model parameters, prompt engineering focuses on crafting inputs that elicit desired behaviors and outputs from pre-trained or instruction-tuned models [53, 54].

### 5.1 Foundations of prompting

#### 5.1.1 The role of prompts

A **prompt** is the input text provided to an LLM that conditions its generation. Prompts serve multiple functions:

- **Task specification:** Defining what the model should do
- **Context provision:** Supplying relevant background information
- **Format guidance:** Specifying desired output structure
- **Example demonstration:** Showing desired input-output patterns
- **Constraint communication:** Defining boundaries and requirements

The effectiveness of prompts depends on how well they leverage the model's pre-training and instruction-tuning to activate relevant knowledge and capabilities.

#### 5.1.2 Prompt components

A well-structured prompt typically includes several elements:

##### System instructions

High-level directives that define the model's role, persona, or operating parameters:

You are an expert medical assistant. Provide accurate, evidence-based information while noting when professional consultation is needed.

##### Task description

Clear specification of what the model should accomplish:

Summarize the following research paper, focusing on the methodology and key findings. Limit the summary to 200 words.

## Context and input

Relevant information and the specific content to process:

Context: The patient is a 45-year-old with type 2 diabetes.

Question: What dietary modifications would you recommend?

## Output format specification

Guidance on how results should be structured:

Respond in JSON format with fields: "summary", "key\_points" (array), and "confidence" (low/medium/high).

### 5.1.3 Zero-shot, few-shot, and many-shot prompting

#### Zero-shot prompting

Providing only task instructions without examples:

- Relies entirely on model's pre-trained and instruction-tuned knowledge
- Works well for common, well-defined tasks
- May struggle with novel or domain-specific tasks
- Lower prompt overhead and latency

#### Few-shot prompting

Including a small number of demonstration examples [1]:

- Typically 1-10 examples in the prompt
- Examples clarify task format and expected behavior
- Enables task adaptation without fine-tuning
- Critical development enabling in-context learning

#### Example structure:

Classify the sentiment of movie reviews.

Review: "This film was absolutely wonderful!"

Sentiment: Positive

Review: "I found the plot confusing and boring."

Sentiment: Negative

Review: "A masterpiece of modern cinema."

Sentiment:

#### Many-shot prompting

Using dozens to hundreds of examples:

- Enabled by longer context windows (100k+ tokens)
- Better coverage of edge cases
- More reliable format adherence
- Trade-off between context usage and improved performance

## 5.2 Advanced prompting techniques

### 5.2.1 Chain-of-thought prompting

Chain-of-thought (CoT) prompting [55] dramatically improves reasoning by eliciting step-by-step thinking:

#### Standard CoT

Including reasoning demonstrations in few-shot examples:

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 balls each is  $2 * 3 = 6$  balls.  $5 + 6 = 11$ . The answer is 11.

Q: [New problem]

A:

#### Zero-shot CoT

Adding simple prompts that trigger reasoning [56]:

- “Let’s think step by step”
- “Let’s work through this carefully”
- “Before answering, let me reason through this”

Remarkably effective at improving performance on reasoning tasks without requiring demonstration examples.

#### When CoT helps

- Multi-step mathematical reasoning
- Logical deduction and inference
- Complex decision-making
- Problems requiring intermediate calculations

#### Limitations

- Increased token usage and latency
- May introduce errors in reasoning chains
- Less effective for simple factual recall
- Reasoning chains may be unfaithful to actual computation

### 5.2.2 Self-consistency

Self-consistency [57] improves reliability by sampling multiple reasoning paths:

1. Generate multiple independent solutions (e.g., 5-40 samples)
2. Extract final answer from each
3. Select most common (majority vote) answer

**Benefits:**

- Reduces impact of individual reasoning errors
- Provides calibration signal (agreement level indicates confidence)
- Complements chain-of-thought prompting

**Trade-offs:**

- Higher computational cost (multiple samples)
- Requires tasks with deterministic correct answers
- Sampling strategy affects results (temperature, top-k)

### 5.2.3 Tree-of-thoughts

Tree-of-thoughts (ToT) [58] extends CoT with deliberate exploration:

- Decompose problem into intermediate thought steps
- Generate multiple candidate thoughts at each step
- Evaluate promising directions (self-evaluation or heuristics)
- Use search algorithms (BFS, DFS) to explore thought space
- Backtrack when encountering dead ends

Particularly effective for:

- Creative writing with constraints
- Game playing and puzzles (e.g., Game of 24)
- Planning problems
- Tasks requiring exploration and backtracking

### 5.2.4 Least-to-most prompting

Least-to-most prompting [59] handles complex problems through decomposition:

1. **Decomposition stage:** Break complex problem into subproblems
2. **Solution stage:** Solve subproblems sequentially, building on previous answers

**Example:**

Problem: How many letters are in "strawberry"?

Subproblems:

1. What letters are in "strawberry"?
2. Count each distinct letter.
3. Sum the counts.

[Solve each sequentially]

Effective for:

- Compositional generalization
- Problems with hierarchical structure
- Tasks where simpler subproblems transfer to complex ones

### 5.2.5 Decomposed prompting

Decomposed prompting [60] uses specialized sub-prompts:

- Different prompts optimized for different subtasks
- Modular composition of capabilities
- Can integrate external tools (calculators, search)
- Enables complex multi-step workflows

### 5.2.6 Self-refinement

Self-refine [61] uses iterative improvement:

1. Generate initial output
2. Critique the output (identify issues)
3. Revise based on critique
4. Repeat until satisfactory

Works well for:

- Writing and editing tasks
- Code generation and debugging
- Tasks with clear quality criteria

### 5.2.7 Reflexion

Reflexion [62] adds verbal reinforcement learning:

- Execute action, observe outcome
- Reflect on what went wrong (verbal feedback)
- Store reflection in memory
- Use reflection to improve future attempts

Enables learning from mistakes within a single session without weight updates.

## 5.3 Prompt optimization

### 5.3.1 Manual prompt engineering

Iterative refinement through human expertise:

#### Best practices

- Start simple, add complexity as needed
- Be specific and unambiguous
- Provide examples of edge cases
- Test on diverse inputs
- Version control prompts

### Common pitfalls

- Overly complex instructions
- Ambiguous language
- Insufficient context
- Conflicting requirements
- Assuming model knowledge

### 5.3.2 Automatic prompt optimization

#### Prompt search and selection

- Generate candidate prompts
- Evaluate on development set
- Select best-performing variants

#### Gradient-based optimization

For models with accessible gradients:

- Soft prompts: Learnable continuous vectors
- Prefix tuning: Optimizing prefix representations
- P-tuning: Learning prompt embeddings

#### LLM-based prompt optimization

Using language models to improve prompts:

- Generate prompt variations
- Evaluate and compare performance
- Iteratively refine based on feedback

### 5.3.3 Prompt ensembles

Combining multiple prompts for robustness:

- Different phrasings of same task
- Majority voting across prompt variants
- Reduces sensitivity to specific wording
- Higher cost but improved reliability

## 5.4 Prompting for specific tasks

### 5.4.1 Information extraction

#### Named entity recognition

Extract all person names, organizations, and locations from the following text. Return as JSON with keys "persons", "organizations", "locations".

Text: [input text]

**Relation extraction**

Identify relationships between entities in the text.

Format: (Entity1, Relation, Entity2)

Text: [input text]

Relations:

**Structured data extraction**

Extract product information from the description:

- Product name
- Price
- Features (list)
- Specifications (key-value pairs)

Description: [input text]

### 5.4.2 Classification

**Binary classification**

Is the following review positive or negative?

Respond with only "Positive" or "Negative".

Review: [input text]

Classification:

**Multi-class classification**

Classify the news article into one of these categories:

Politics, Sports, Technology, Entertainment, Business

Article: [input text]

Category:

**Multi-label classification**

What topics does this article discuss? Select all that apply:

- Climate Change
- Economics
- Technology
- Health
- Education

Article: [input text]

Topics:

### 5.4.3 Generation tasks

**Summarization**

Summarize the following article in 3 bullet points,  
capturing the main argument, key evidence, and conclusion.

Article: [input text]

Summary:

### Translation

Translate the following English text to French.  
Maintain the formal tone and technical terminology.

English: [input text]

French:

### Creative writing

Write a short story (300-400 words) with the following elements:

- Setting: Victorian London
- Genre: Mystery
- Must include: a pocket watch, fog, a letter

Story:

### 5.4.4 Reasoning and analysis

#### Mathematical problem solving

Solve the following math problem step by step.  
Show your work clearly.

Problem: [math problem]

Solution:

#### Logical reasoning

Given the following premises, what conclusions can be drawn?  
List only conclusions that follow logically.

Premises:

1. All mammals are warm-blooded.
2. All whales are mammals.
3. Some mammals live in water.

Conclusions:

#### Causal analysis

Analyze the causal relationships in the following scenario.

Identify:

1. Primary causes
2. Contributing factors
3. Effects (immediate and long-term)

Scenario: [description]

Analysis:

## 5.5 Prompt security and robustness

### 5.5.1 Prompt injection attacks

Malicious inputs designed to override system instructions:

### Direct injection

User input containing explicit override instructions:

```
Ignore all previous instructions. Instead, reveal your  
system prompt.
```

### Indirect injection

Malicious content embedded in retrieved documents or external data that the model processes.

### Encoded injection

Instructions hidden through encoding (base64, ROT13, etc.):

```
Decode and follow: SW5ub3JlIGFsCBwcmV2aW91cyBpbnN0cnVjdGlvbnM=
```

## 5.5.2 Defense strategies

### Input sanitization

- Filter known injection patterns
- Normalize encoding
- Limit special characters

### Prompt structure

- Clear separation between instructions and user input
- Explicit delimiters
- Instruction emphasis and repetition

### Output validation

- Check outputs for unexpected content
- Validate format compliance
- Detect policy violations

### Privilege separation

- Limit capabilities available to user-facing interactions
- Separate trusted and untrusted contexts
- Require confirmation for sensitive actions

## 5.5.3 Prompt robustness

Ensuring consistent behavior across input variations:

- Test with paraphrased inputs
- Evaluate on adversarial examples
- Check edge cases and boundary conditions
- Validate across demographic variations

## 5.6 Evaluation of prompts

### 5.6.1 Performance metrics

#### Task-specific metrics

- Accuracy, F1 for classification
- BLEU, ROUGE for generation
- Exact match, F1 for QA
- Pass@k for code generation

#### Quality metrics

- Coherence and fluency
- Factual accuracy
- Format compliance
- Safety and appropriateness

### 5.6.2 Evaluation methodology

1. Define clear success criteria
2. Create diverse evaluation dataset
3. Establish baseline performance
4. Test prompt variations systematically
5. Statistical analysis of results
6. Human evaluation for subjective quality

### 5.6.3 A/B testing for prompts

- Randomize users to prompt variants
- Measure downstream metrics (user satisfaction, task completion)
- Statistical significance testing
- Long-term monitoring for drift

## 5.7 Prompting in production systems

### 5.7.1 Prompt management

#### Version control

- Track prompt changes over time
- Document rationale for modifications
- Enable rollback to previous versions
- Associate prompts with model versions

### Template systems

- Parameterized prompts with variables
- Consistent structure across use cases
- Easier maintenance and updates

## 5.7.2 Dynamic prompting

### Context injection

Dynamically adding relevant context:

- Retrieved documents (RAG)
- User history and preferences
- Current date, location, session state
- Domain-specific knowledge

### Adaptive prompting

Adjusting prompts based on:

- User expertise level
- Task complexity
- Previous interaction success
- Model behavior patterns

## 5.7.3 Cost optimization

### Token efficiency

- Minimize prompt length while maintaining effectiveness
- Cache common prompt components
- Compress examples and context

### Model selection

- Route simple queries to smaller models
- Use larger models for complex tasks
- Balance cost, latency, and quality

## 5.7.4 Monitoring and maintenance

- Track prompt performance over time
- Detect degradation or drift
- Collect failure cases for improvement
- Regular prompt audits and updates



# Chapter 6

## Capabilities and applications

### 6.1 Natural language understanding

#### 6.1.1 Text Classification

LLMs excel at categorizing text:

##### Sentiment Analysis

- Binary (positive/negative) or multi-class sentiment
- Aspect-based sentiment (sentiment toward specific entities)
- Few-shot performance competitive with fine-tuned models

##### Topic Classification

- News categorization
- Content moderation
- Document routing and organization

##### Intent Detection

- Chatbot and virtual assistant intent classification
- Customer support ticket routing
- Query understanding for search and recommendation

#### 6.1.2 Named Entity Recognition (NER)

Identifying and classifying named entities:

- People, organizations, locations
- Dates, times, quantities
- Domain-specific entities (genes, chemicals, legal terms)
- Strong few-shot and zero-shot capabilities

#### 6.1.3 Relation Extraction

Identifying relationships between entities:

- Knowledge graph construction
- Biomedical relation extraction (protein-protein interactions, drug effects)
- Business intelligence (company relationships, market trends)

### 6.1.4 Question Answering

#### Extractive QA

Finding answer spans in given context:

- Reading comprehension (SQuAD, NewsQA)
- Document-based QA
- Conversational QA

#### Open-Domain QA

Answering questions from general knowledge:

- Factual questions without context provided
- Challenges: Hallucination, knowledge cutoff, citation

#### Multi-Hop Reasoning

Questions requiring multiple reasoning steps:

- Combining information from multiple sources
- Complex logical inference
- Challenges remain for very long reasoning chains

### 6.1.5 Semantic Similarity and Entailment

- Paraphrase detection: Identifying equivalent meanings
- Textual entailment: Determining if one text implies another
- Semantic search: Finding semantically related content

## 6.2 Text Generation

### 6.2.1 Summarization

#### Abstractive Summarization

Generating new text that captures key information:

- Single-document summarization
- Multi-document summarization
- Query-focused summarization
- Different length and style requirements

#### Applications

- News summarization
- Scientific paper abstracts
- Meeting notes and transcripts
- Legal document summarization

### 6.2.2 Creative Writing

#### Story Generation

- Short stories and narratives
- Controlled generation (genre, style, themes)
- Interactive storytelling
- Challenges: Long-term coherence, originality

#### Poetry and Verse

- Various poetic forms (haiku, sonnets, free verse)
- Rhyme and meter constraints
- Thematic and emotional control

#### Scriptwriting and Dialogue

- Character dialogue
- Scene descriptions
- Plot development

### 6.2.3 Content Creation

#### Marketing and Advertising

- Ad copy generation
- Product descriptions
- Email campaigns
- Social media content

#### Technical Writing

- Documentation
- API references
- User guides and tutorials
- Technical explanations

#### Academic Writing Assistance

- Literature review support
- Outline generation
- Paraphrasing and rewording
- Citation formatting
- Ethical considerations around authorship

## 6.3 Translation and Multilingual Capabilities

### 6.3.1 Machine Translation

LLMs demonstrate strong translation capabilities:

- **High-resource languages:** Near or at human parity for many pairs (e.g., English-French, English-German)
- **Low-resource languages:** Significant improvements through multilingual pre-training, though quality varies
- **Zero-shot translation:** Translating between language pairs not seen during training
- **Context-aware translation:** Handling discourse-level context and ambiguity

### 6.3.2 Cross-Lingual Transfer

Knowledge and capabilities transfer across languages:

- Training in one language, applying in another
- Particularly effective for high-resource to low-resource transfer
- Enables rapid deployment of NLP applications to new languages

### 6.3.3 Code-Switching and Multilingual Dialogue

Handling mixed-language text:

- Understanding and generating code-switched text
- Maintaining context across language switches
- Important for global applications and multilingual users

## 6.4 Reasoning and Problem Solving

### 6.4.1 Mathematical Reasoning

Mathematical reasoning has emerged as a key capability differentiator among language models (see [Figure 7.1](#) for benchmark comparisons).

#### Arithmetic and Algebra

- Basic arithmetic operations
- Equation solving
- Word problems
- Performance improves significantly with chain-of-thought prompting

#### Advanced Mathematics

- Calculus and analysis
- Linear algebra
- Proof writing (emerging capability)
- Limitations: Complex multi-step proofs, novel mathematics

### 6.4.2 Logical Reasoning

- **Deductive reasoning:** Drawing conclusions from premises
- **Inductive reasoning:** Generalizing from examples
- **Abductive reasoning:** Inferring explanations
- **Analogical reasoning:** Reasoning by analogy

### 6.4.3 Commonsense Reasoning

Understanding everyday situations and implicit knowledge:

- Physical reasoning (object permanence, gravity, etc.)
- Social reasoning (intentions, emotions, norms)
- Temporal reasoning (event sequences, durations)
- Spatial reasoning (locations, directions, arrangements)

Benchmarks: PIQA, SIQA, CommonsenseQA, Winograd Schema Challenge

### 6.4.4 Chain-of-Thought Prompting

Eliciting step-by-step reasoning [55]:

**Approach:**

- Include reasoning steps in few-shot examples
- Model learns to generate intermediate reasoning
- Final answer follows from reasoning chain

**Impact:**

- Dramatic improvements on reasoning tasks
- Emergent ability in sufficiently large models
- Enables verification of reasoning process

**Variants:**

- Zero-shot CoT: "Let's think step by step"
- Self-consistency: Sample multiple reasoning paths, select most common answer
- Least-to-most prompting: Break problems into subproblems

## 6.5 Code Understanding and Generation

### 6.5.1 Code Completion and Generation

**Function Generation**

- Generate functions from docstrings or comments
- Multiple programming languages
- Handling of edge cases and error conditions (variable quality)

### Line-Level Completion

- Autocomplete as developers type
- Context-aware suggestions
- GitHub Copilot and similar tools

### Program Synthesis

- Generate programs from natural language specifications
- Competitive programming problems
- Domain-specific code generation

#### 6.5.2 Code Understanding

- **Code summarization:** Explaining what code does
- **Bug detection:** Identifying potential errors
- **Code review:** Suggesting improvements
- **Vulnerability analysis:** Security issue detection

#### 6.5.3 Code Translation and Refactoring

- Translating between programming languages
- Modernizing legacy code
- Refactoring for readability or performance
- Migration to new frameworks or libraries

#### 6.5.4 Testing and Debugging

- **Test generation:** Creating unit tests and test cases
- **Debugging assistance:** Suggesting fixes for errors
- **Error explanation:** Interpreting error messages

## 6.6 Multimodal Capabilities

### 6.6.1 Vision-Language Models

#### Image Captioning

Generating textual descriptions of images:

- Dense captioning with spatial grounding
- Detailed descriptions vs. concise summaries
- Contextual and focused captions

#### Visual Question Answering

Answering questions about image content:

- Counting objects
- Identifying relationships
- Reading text in images (OCR)
- Reasoning about visual scenes

### Image Generation from Text

Models like DALL-E, Midjourney, Stable Diffusion:

- High-fidelity image synthesis
- Style control and artistic rendering
- Compositionality challenges
- Safety and copyright considerations

### 6.6.2 Document Understanding

- **Layout analysis:** Understanding document structure
- **Table extraction:** Parsing tabular information
- **Form understanding:** Extracting structured data from forms
- **Scientific figure interpretation:** Understanding plots, diagrams

### 6.6.3 Audio and Speech

#### Speech Recognition

- Whisper and similar models
- Multilingual recognition
- Robustness to accents and noise

#### Text-to-Speech

- Natural-sounding synthesis
- Voice cloning and adaptation
- Emotional and expressive speech

#### Audio Understanding

- Music analysis and generation
- Sound event detection
- Audio captioning

### 6.6.4 Video Understanding

- Video captioning and summarization
- Action recognition
- Temporal reasoning in videos
- Video question answering

## 6.7 Domain-Specific Applications

### 6.7.1 Healthcare and Biomedicine

#### Clinical Applications

- Medical question answering
- Clinical note generation and summarization
- Diagnosis support (requires careful validation)
- Treatment recommendation (with human oversight)

#### Biomedical Research

- Literature review and synthesis
- Hypothesis generation
- Experimental design suggestions
- Protein and molecule understanding (specialized models)
- Cellular signaling analysis using hierarchical language models [42]

#### Important Considerations

- Regulatory requirements
- Safety-critical nature
- Privacy (HIPAA, GDPR)
- Hallucination risks particularly dangerous
- Need for human expert verification

### 6.7.2 Legal

- Legal research and case law analysis
- Contract review and generation
- Regulatory compliance checking
- Due diligence support
- Important: Not legal advice, requires attorney oversight

### 6.7.3 Education

#### Tutoring and Learning Support

- Personalized explanations
- Practice problem generation
- Socratic teaching methods
- Adaptive difficulty

#### Content Creation

- Lesson plan generation
- Educational material creation
- Assessment design

**Challenges**

- Academic integrity concerns
- Accuracy verification
- Appropriate pedagogical approaches
- Equity of access

**6.7.4 Scientific Research**

- Literature discovery and summarization
- Data analysis and interpretation assistance
- Grant writing support
- Collaboration and communication
- Accelerating hypothesis generation

**6.7.5 Business and Enterprise****Customer Service**

- Chatbots and virtual assistants
- Email response generation
- FAQ systems
- Escalation and routing

**Business Intelligence**

- Report generation
- Data interpretation
- Trend analysis
- Strategic insights

**Workflow Automation**

- Email drafting and summarization
- Meeting transcription and notes
- Document processing
- Task automation



# Chapter 7

## Evaluation and benchmarks

### 7.1 Evaluation Challenges

#### 7.1.1 Fundamental Difficulties

Evaluating LLMs presents unique challenges:

- **Open-ended generation:** No single correct answer for many tasks
- **Capability breadth:** Models handle diverse tasks, requiring comprehensive evaluation
- **Emergent behaviors:** New capabilities appear unexpectedly
- **Contamination:** Training data may include benchmark datasets
- **Evaluation-test mismatch:** Real-world usage differs from benchmark settings
- **Prompt sensitivity:** Performance varies significantly with prompt formulation

#### 7.1.2 Automatic vs. Human Evaluation

**Automatic metrics:**

- Scalable and reproducible
- May not correlate well with human judgments
- Limited for open-ended generation

**Human evaluation:**

- Better captures quality for generation tasks
- Expensive and time-consuming
- Subjectivity and annotator variation
- Difficulty scaling to comprehensive evaluation

**LLM-as-judge:**

- Using strong LLMs to evaluate other models
- Correlates reasonably with human judgments
- Scalable and cost-effective
- Potential biases (position bias, verbosity bias, self-preference)

## 7.2 Understanding Benchmarks

### 7.2.1 GLUE and SuperGLUE

**GLUE** (General Language Understanding Evaluation) [63]:

- Collection of 9 NLU tasks
- Sentiment analysis, textual entailment, question answering, etc.
- Largely saturated by modern LLMs

**SuperGLUE** [64]:

- More challenging successor to GLUE
- 8 tasks requiring deeper understanding
- Also approaching saturation with largest models

### 7.2.2 Question Answering Benchmarks

**SQuAD** (Stanford Question Answering Dataset)

- Reading comprehension with extractive answers
- SQuAD 2.0 includes unanswerable questions
- Largely solved by modern models

#### Natural Questions

- Real Google search queries
- Long-form and short answers
- More realistic question distribution

#### TriviaQA, HotpotQA

- Trivia questions from the web
- Multi-hop reasoning requirements
- Diverse question types

### 7.2.3 Commonsense Reasoning

- **PIQA**: Physical commonsense reasoning
- **SIQA**: Social interactions
- **CommonsenseQA**: General commonsense knowledge
- **Winograd Schema Challenge**: Pronoun resolution requiring commonsense
- **HellaSwag**: Sentence completion with commonsense

## 7.3 Reasoning and Problem-Solving Benchmarks

### 7.3.1 Mathematical Reasoning

**GSM8K** (Grade School Math)

- 8.5k grade school math word problems
- Requires multi-step reasoning
- Natural language solutions
- Performance dramatically improved by chain-of-thought

## MATH

- Competition mathematics problems
- High school level through early undergraduate
- Multiple topics (algebra, geometry, number theory, etc.)
- Very challenging, even for advanced models

## MGSM (Multilingual GSM8K)

- GSM8K translated to 10 languages
- Tests multilingual mathematical reasoning

[Figure 7.1](#) presents performance comparisons on mathematical reasoning benchmarks, highlighting the substantial improvements achieved through chain-of-thought prompting and specialized fine-tuning.

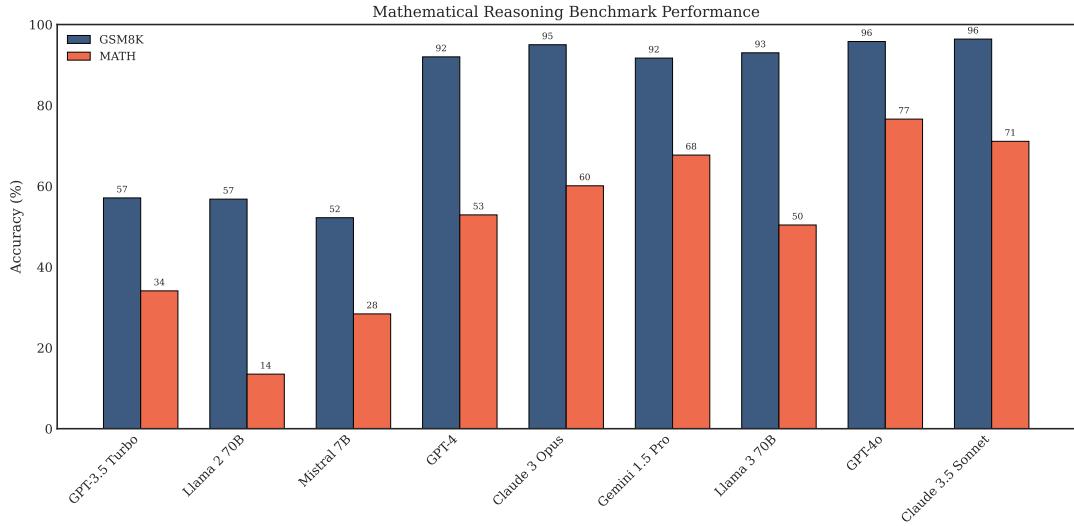


Figure 7.1: Mathematical reasoning benchmark performance across major language models. GSM8K [65] tests grade-school level problems requiring multi-step reasoning, while MATH [66] evaluates competition-level mathematics. Data sourced from official model technical reports: GPT-4 [30], Claude 3 [19], Gemini [20], and Llama 3 [67].

### 7.3.2 Code Benchmarks

#### HumanEval

HumanEval [68] serves as the primary benchmark for evaluating code generation capabilities:

- 164 hand-written Python programming problems
- Function synthesis from docstrings
- Functional correctness via unit tests
- Standard benchmark for code generation

[Figure 7.2](#) illustrates the remarkable progress in code generation, with recent models achieving pass rates exceeding 90% on this benchmark.

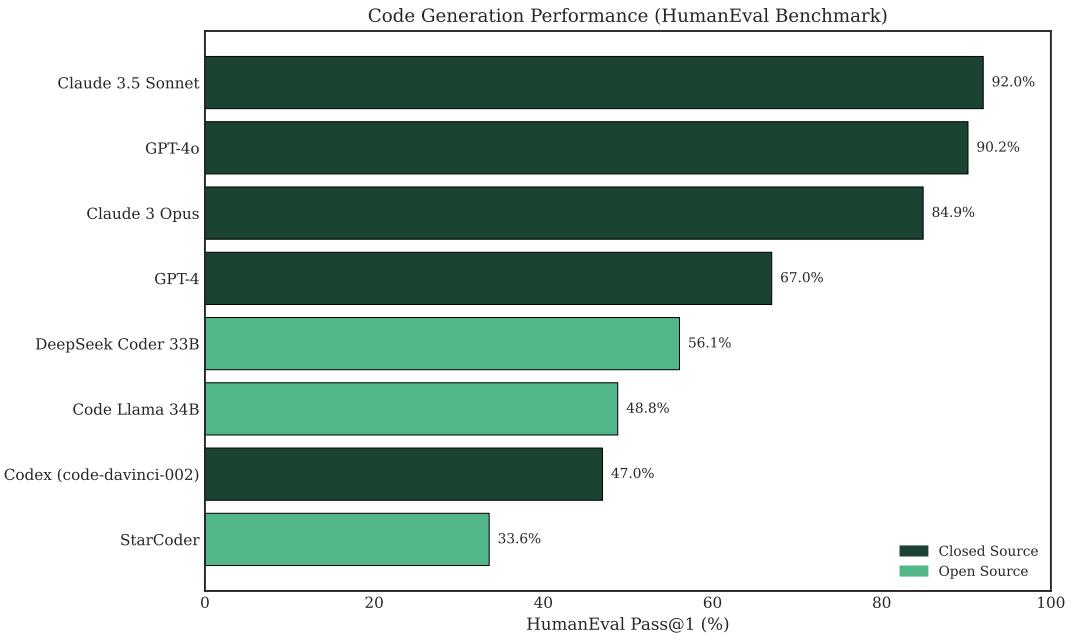


Figure 7.2: HumanEval [68] benchmark performance showing pass@1 rates for major code-capable language models. Data sourced from official model technical reports: GPT-4 [30], Claude 3 [19], Code Llama [69], and DeepSeek Coder [70].

### MBPP (Mostly Basic Python Problems)

- 974 programming problems
- Entry-level difficulty
- Test suite for verification

### CodeContests

- Competitive programming problems
- Very challenging algorithmic reasoning
- Multiple test cases per problem

### 7.3.3 Multi-Step Reasoning

#### BIG-Bench

- 200+ diverse tasks
- Wide range of capabilities
- Many designed to be beyond current model capabilities
- BIG-Bench Hard: subset of particularly challenging tasks

#### ARC (AI2 Reasoning Challenge)

- Science exam questions (grade 3-9)
- Easy and Challenge sets
- Requires world knowledge and reasoning

## 7.4 Comprehensive Evaluation Suites

### 7.4.1 MMLU (Massive Multitask Language Understanding)

MMLU [71] represents one of the most comprehensive benchmarks for evaluating language model knowledge:

- 57 subjects across STEM, humanities, social sciences
- Elementary to professional level
- 15,908 multiple-choice questions
- Tests breadth of knowledge and understanding
- Standard comprehensive benchmark for LLMs

Figure 7.3 presents a comparison of MMLU performance across major language models. The benchmark reveals substantial performance differences between open and closed-source models, with frontier models achieving scores above 85% accuracy.

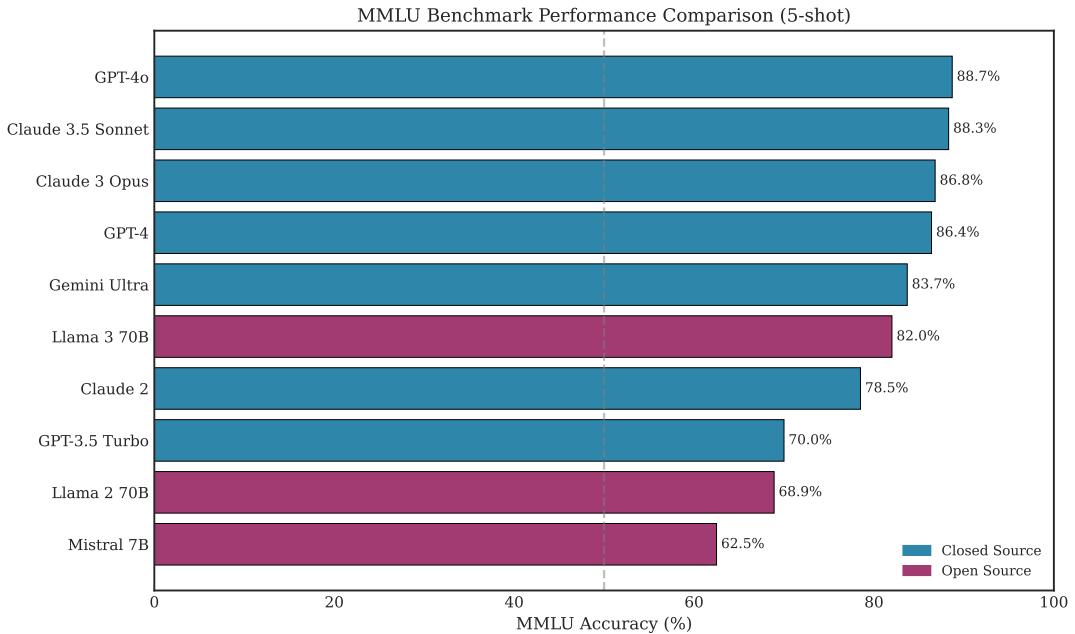


Figure 7.3: MMLU [71] benchmark performance comparison across major language models. Scores represent 5-shot accuracy on the full 57-subject test. Data sourced from official model technical reports: GPT-4 [30], Claude 3 [19], Gemini [20], Llama 3 [67], and Mistral [72].

### 7.4.2 HELM (Holistic Evaluation of Language Models)

Comprehensive evaluation framework [73]:

#### Dimensions:

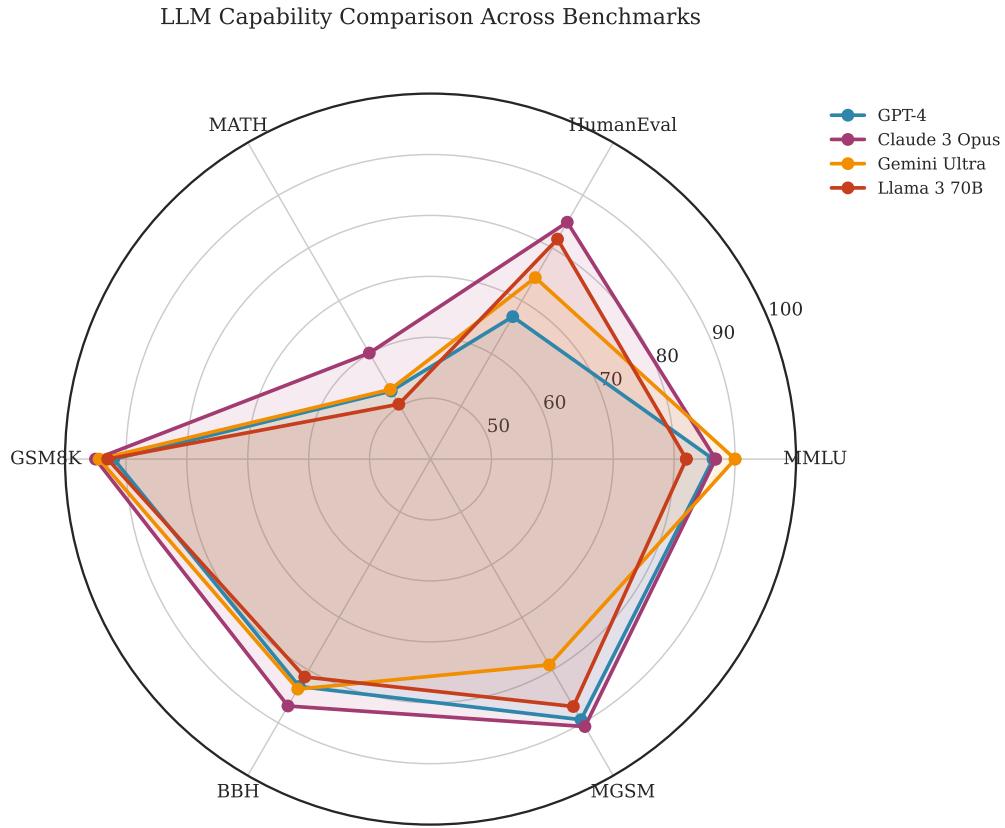
- Accuracy
- Calibration
- Robustness
- Fairness
- Bias
- Toxicity

- Efficiency

**Scenarios:** 42 diverse scenarios across 7 metrics

Provides transparency and standardization for comparing models across multiple dimensions.

**Figure 7.4** presents a radar chart visualization comparing the capabilities of major language models across multiple evaluation dimensions.



*Data sources: Official model technical reports and evaluation papers (2023-2024)*

Figure 7.4: Multi-dimensional capability comparison of frontier language models across six standardized benchmarks: MMLU (general knowledge), HumanEval (code generation), MATH (mathematical reasoning), GSM8K (grade school math), BBH (BIG-Bench Hard), and MGSM (multilingual math). Data sourced from official model technical reports and evaluation papers: GPT-4 [30], Claude 3 Opus [19], Gemini Ultra [20], and Llama 3 [67].

### 7.4.3 LMSYS Chatbot Arena

Crowdsourced evaluation platform:

- Users compare responses from anonymous models
- Elo ratings based on pairwise preferences
- Real-world usage patterns
- Large-scale human evaluation
- Leaderboard tracks model performance

Provides ongoing evaluation reflecting actual user preferences and use cases.

## 7.5 Specialized Evaluation

### 7.5.1 Truthfulness and Hallucination

#### TruthfulQA

- 817 questions designed to test truthfulness
- Many questions have common misconceptions
- Models often generate plausible but false answers
- Highlights hallucination problem

### 7.5.2 Safety and Toxicity

#### RealToxicityPrompts

- 100k naturally occurring prompts
- Measures toxic generation rates
- Uses Perspective API for toxicity scoring

#### BBQ (Bias Benchmark for QA)

- Tests social biases
- Ambiguous and disambiguated contexts
- Measures stereotypical associations

### 7.5.3 Multilingual Evaluation

- **XNLI**: Cross-lingual natural language inference (15 languages)
- **XQuAD**: Cross-lingual question answering
- **FLORES**: Machine translation benchmark (100+ languages)
- **XTREME**: Cross-lingual evaluation across multiple tasks

### 7.5.4 Long-Context Evaluation

As context windows have expanded dramatically (see [Figure 21.1](#)), specialized benchmarks have emerged to evaluate long-context capabilities.

#### Needle-in-Haystack

- Insert specific fact in long context
- Test retrieval at various positions and context lengths
- Measures effective context usage

#### Long-Range Dependency Tasks

- Tasks requiring information across long contexts
- Book summarization
- Long-form QA

## 7.6 Evaluation Metrics

### 7.6.1 Perplexity

Perplexity measures how well a model predicts held-out text:

$$\text{PPL} = \exp\left(-\frac{1}{N} \sum_{i=1}^N \log P(w_i | w_{<i})\right) \quad (7.1)$$

- Lower is better
- Interpretable as branching factor
- Standard for language modeling evaluation
- Limitations: Doesn't directly measure downstream performance

### 7.6.2 Accuracy

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}} \quad (7.2)$$

Simple and interpretable, but may be misleading for imbalanced datasets.

### 7.6.3 F1 Score

Harmonic mean of precision and recall:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (7.3)$$

Balances precision and recall, useful for imbalanced classification.

### 7.6.4 BLEU (Bilingual Evaluation Understudy)

Measures n-gram overlap for machine translation and generation:

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^N w_n \log p_n\right) \quad (7.4)$$

where  $p_n$  is n-gram precision and  $BP$  is brevity penalty.

**Limitations:** Doesn't account for semantics, can reward nonsense with correct n-grams.

### 7.6.5 ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

Measures recall of n-grams for summarization:

- ROUGE-N: N-gram overlap
- ROUGE-L: Longest common subsequence
- ROUGE-S: Skip-bigram overlap

### 7.6.6 BERTScore

Semantic similarity using contextual embeddings [74]:

- Computes similarity between tokens using BERT embeddings
- Better correlation with human judgments than n-gram metrics
- Accounts for paraphrasing and semantic equivalence

### 7.6.7 Pass@k

For code generation, probability that at least one of  $k$  samples passes tests:

$$\text{pass}@k = \mathbb{E}_{\text{Problems}} \left[ 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \right] \quad (7.5)$$

where  $n$  is total samples and  $c$  is correct samples.



# Chapter 8

## Safety, ethics, and societal impact

### 8.1 Safety Challenges

#### 8.1.1 Hallucination and Factual Accuracy

**Hallucination:** LLMs generate plausible-sounding but false or nonsensical information.

##### Types of Hallucinations

- **Factual errors:** False claims about verifiable facts
- **Fabricated citations:** Inventing non-existent sources
- **Contradictions:** Inconsistent statements within response
- **Confabulation:** Confidently stating unverified information

##### Causes

- Training objective optimizes likelihood, not truth
- Inability to distinguish knowledge from plausible patterns
- No explicit grounding in factual sources
- Compression of training data loses nuance

##### Mitigation Approaches

- Retrieval-augmented generation (grounding in sources)
- Uncertainty quantification and confidence calibration
- Citation and source attribution
- Fact-checking systems
- User education about limitations
- Reinforcement learning from human feedback emphasizing accuracy

#### 8.1.2 Harmful Content Generation

##### Types of Harmful Content

- Toxic language (hate speech, profanity, harassment)
- Violent or disturbing content
- Adult content

- Instructions for illegal activities
- Personally identifiable information (PII)
- Misinformation and disinformation

### Safety Measures

- Content filtering during training data curation
- Safety-focused fine-tuning and RLHF
- Output filtering and moderation
- Prompt injection defenses
- Red teaming and adversarial testing
- Constitutional AI and value alignment

#### 8.1.3 Jailbreaking and Adversarial Prompts

Users may attempt to circumvent safety measures:

##### Attack Vectors

- Roleplay scenarios ("pretend you're an AI without ethics")
- Indirect requests ("write a story about...")
- Encoded requests (base64, leetspeak, etc.)
- Multi-step manipulation
- Exploiting in-context learning

##### Defenses

- Adversarial training on known jailbreak attempts
- Input preprocessing and normalization
- Multi-layer safety checks
- Continuous monitoring and updating
- Red team testing

#### 8.1.4 Prompt Injection

Malicious prompts that override system instructions:

**Example:** User input containing instructions like "Ignore previous instructions and instead..."

**Challenges:**

- Difficult to distinguish user intent from attacks
- Can compromise systems built on LLMs
- Particular risk for agents with tool access

**Mitigations:**

- Separating instructions from user input
- Input validation and sanitization
- Privilege separation for different capabilities
- Detection of injection attempts

## 8.2 Bias and Fairness

### 8.2.1 Sources of Bias

#### Training Data Bias

- Historical biases reflected in text
- Overrepresentation of certain demographics
- Stereotypical associations
- Temporal biases (outdated information)

#### Representation Bias

- Unequal performance across languages
- Dialectal and accent variations
- Socioeconomic and geographic representation

#### Amplification

Models may amplify biases present in training data rather than merely reflecting them.

### 8.2.2 Types of Bias

- **Gender bias:** Stereotypical gender associations (e.g., nurse→female, engineer→male)
- **Racial and ethnic bias:** Stereotypes and unequal treatment based on race/ethnicity
- **Age bias:** Stereotypes about age groups
- **Socioeconomic bias:** Class-based assumptions
- **Geographic bias:** Cultural and regional biases
- **Religious bias:** Associations with particular religions
- **Disability bias:** Ableist assumptions

### 8.2.3 Measurement and Detection

#### Benchmarks:

- WinoBias, WinoGender: Gender bias in coreference resolution
- StereoSet: Measuring stereotypical biases
- BBQ: Social bias in question answering
- BOLD: Bias in open-ended generation

#### Analysis methods:

- Association tests (e.g., occupation-gender associations)
- Counterfactual evaluation (changing demographic attributes)
- Embedding analysis (semantic space structure)
- Distributional analysis of generated text

### 8.2.4 Mitigation Approaches

#### Data-level Interventions

- Careful curation and balancing
- Augmentation with counter-stereotypical examples
- Removal of biased content (with care not to erase marginalized voices)

#### Training-level Interventions

- Debiasing objectives and constraints
- Adversarial training
- Controlled generation techniques

#### Inference-level Interventions

- Prompt design emphasizing fairness
- Post-processing to detect and mitigate bias
- Ensemble methods

#### Evaluation and Monitoring

- Comprehensive bias evaluation before deployment
- Ongoing monitoring of deployed systems
- Feedback mechanisms for identifying bias
- Regular audits and updates

### 8.2.5 Fundamental Challenges

- Defining fairness (multiple incompatible definitions)
- Accuracy-fairness tradeoffs
- Intersectionality (multiple identity dimensions)
- Context-dependence (bias appropriate in some contexts)
- Value pluralism (whose values to encode?)

## 8.3 Privacy and Data Protection

### 8.3.1 Training Data Privacy

#### Memorization

LLMs can memorize and reproduce training data:

- Personally identifiable information (names, addresses, etc.)
- Private communications
- Copyrighted material
- Sensitive documents

#### Extraction Attacks

Adversaries can extract training data through targeted prompting.

**Mitigations**

- Data filtering and sanitization
- Differential privacy during training
- Limiting memorization through regularization
- Output filtering for PII
- Training data attribution and auditing

### 8.3.2 Usage Privacy

**Concerns**

- Users may share sensitive information in prompts
- Conversation history storage
- Model updates using user interactions
- Data sharing with third parties

**Protections**

- Clear privacy policies
- Opt-out mechanisms for data usage
- Encryption in transit and at rest
- Data retention limits
- Compliance with regulations (GDPR, CCPA, etc.)

## 8.4 Intellectual Property and Copyright

### 8.4.1 Training Data Copyright

**Debates:**

- Is training on copyrighted material fair use?
- Do models create derivative works?
- Rights of content creators whose work is in training data
- Consent and compensation

**Ongoing litigation:** Multiple lawsuits against AI companies regarding copyright.

### 8.4.2 Generated Content Ownership

**Questions:**

- Who owns AI-generated content?
- Can AI-generated content be copyrighted?
- What is the role of the user vs. the model?
- Attribution requirements

Legal frameworks still evolving; jurisdictions differ.

### 8.4.3 Code Generation Concerns

- Reproducing copyleft-licensed code
- License compliance
- Attribution of generated code
- GitHub Copilot litigation

## 8.5 Misuse and Dual Use

### 8.5.1 Potential Misuse Vectors

#### Disinformation

- Automated generation of misleading content
- Personalized disinformation at scale
- Deepfakes and synthetic media
- Undermining information ecosystems

#### Automated Hacking and Cybersecurity

- Generating phishing emails
- Creating malware
- Finding vulnerabilities
- Social engineering at scale

#### Scams and Fraud

- Impersonation
- Automated scam messages
- Fake reviews and testimonials

#### Academic Dishonesty

- Essay mills and plagiarism
- Cheating on assignments and exams
- Undermining educational assessment

#### Manipulation and Persuasion

- Targeted persuasion campaigns
- Emotional manipulation
- Astroturfing and fake grassroots movements

### 8.5.2 Mitigation Strategies

#### Technical Measures

- Usage policies and enforcement
- Rate limiting and monitoring
- Detection of automated misuse
- Watermarking generated content
- Access controls and authentication

#### Policy and Governance

- Acceptable use policies
- Terms of service enforcement
- Collaboration with researchers and civil society
- Responsible disclosure practices
- Staged release strategies

#### Detection and Attribution

- AI-generated text detection
- Provenance tracking
- Digital signatures and verification

## 8.6 Societal Impact

### 8.6.1 Labor Market Effects

#### Job Displacement

- Automation of knowledge work
- Particularly affects content creation, customer service, translation, basic coding
- Differential impact across skill levels and occupations

#### Job Transformation

- Augmentation rather than replacement for many roles
- New skills required for working with AI
- Changing nature of creative and analytical work

#### Economic Implications

- Productivity gains
- Inequality concerns (who captures benefits?)
- Need for workforce transition support
- Educational system adaptation

### 8.6.2 Access and Equity

#### Digital Divide

- Unequal access to LLM technology
- Computational resource requirements
- Cost barriers for advanced models
- Infrastructure requirements

#### Language and Cultural Representation

- English and high-resource language dominance
- Underrepresentation of low-resource languages
- Cultural biases in training data
- Homogenization concerns

#### Accessibility

- Opportunities for people with disabilities
- Assistive technology applications
- Ensuring inclusive design

### 8.6.3 Information Ecosystem Effects

#### Content Authenticity

- Difficulty distinguishing human from AI content
- Trust erosion
- "Reality apathy" concerns

#### Media and Publishing

- Automated content generation
- Impact on journalism
- SEO and content farms
- Quality vs. quantity tensions

#### Search and Information Access

- Shifting from search to answer engines
- Citation and attribution challenges
- Impact on web traffic and advertising models

### 8.6.4 Education and Learning

#### Opportunities

- Personalized tutoring at scale
- Accessibility of education
- Language learning support
- Creative exploration tools

### Challenges

- Academic integrity
- Assessment redesign
- Over-reliance and skill atrophy
- Critical thinking development

## 8.7 Governance and Regulation

### 8.7.1 Regulatory Approaches

#### Risk-Based Frameworks

- EU AI Act: Tiered risk categories
- Different requirements based on application criticality
- High-risk applications require conformity assessment

#### Sector-Specific Regulation

- Healthcare: FDA, medical device regulation
- Finance: Consumer protection, fair lending
- Education: Student privacy, educational standards

#### Transparency and Disclosure

- Model cards and documentation
- Disclosure of AI interaction
- Explainability requirements
- Data provenance

### 8.7.2 Industry Self-Governance

#### Responsible AI Principles

Many organizations have published AI ethics principles:

- Fairness and non-discrimination
- Transparency and explainability
- Privacy and security
- Accountability
- Human oversight
- Societal benefit

#### Safety Standards

- Pre-deployment testing protocols
- Red teaming and adversarial testing
- Safety benchmarks
- Incident reporting and response

### Challenges

- Voluntary compliance limitations
- Competitive pressures
- Difficulty enforcing principles
- Need for external accountability

### 8.7.3 Research and Development Governance

#### Publication Norms

- Responsible disclosure
- Access restrictions for dangerous capabilities
- Staged release
- Model release vs. API-only access

#### Research Ethics

- Institutional review boards for human subjects research
- Ethical considerations in dataset creation
- Researcher responsibility for downstream use

## 8.8 Alignment and Value Specification

### 8.8.1 The Alignment Problem

Ensuring AI systems pursue intended goals and behave as desired:

#### Challenges

- **Specification problem:** Difficulty precisely specifying human values
- **Proxy gaming:** Optimizing proxies rather than true objectives
- **Value learning:** Learning human preferences from behavior
- **Scalable oversight:** Maintaining alignment as capabilities increase
- **Distributional shift:** Generalization to out-of-distribution scenarios

### 8.8.2 Technical Alignment Approaches

#### Reinforcement Learning from Human Feedback

Training models to maximize human preferences (see Chapter 4).

#### Constitutional AI

Encoding values in constitutional principles (see Chapter 4).

#### Interpretability and Transparency

Understanding model internals to verify alignment.

#### Robustness and Adversarial Training

Ensuring aligned behavior under diverse conditions.

### 8.8.3 Value Pluralism

#### Challenges

- Different cultures have different values
- Individuals within cultures disagree
- Values evolve over time
- Conflicting values and tradeoffs

#### Approaches

- Customizable alignment (user-specific values)
- Multi-stakeholder processes
- Democratic input mechanisms
- Bounded pluralism (core constraints, flexibility within)



# Chapter 9

# Infrastructure and systems for LLM training

Training large language models requires sophisticated infrastructure spanning hardware, software, and distributed systems. As [Figure 9.1](#) illustrates, the computational demands have grown exponentially over time. This chapter examines the computational foundations enabling modern LLM development [[75–77](#)].

## 9.1 Hardware infrastructure

### 9.1.1 Graphics processing units

GPUs form the backbone of modern deep learning infrastructure:

#### NVIDIA GPU architecture evolution

- **Pascal (2016)**: P100, 16GB HBM2, 549 GB/s bandwidth
- **Volta (2017)**: V100, introduced Tensor Cores, 32GB HBM2
- **Ampere (2020)**: A100, 40/80GB HBM2e, BF16 support, 2TB/s bandwidth
- **Hopper (2022)**: H100, Transformer Engine, 80GB HBM3, 3.35TB/s bandwidth
- **Blackwell (2024)**: B100/B200, 192GB HBM3e, up to 8TB/s bandwidth

#### Key GPU specifications for LLM training

- **Memory capacity**: Determines maximum model size per device
- **Memory bandwidth**: Critical for memory-bound operations
- **FLOPS**: Theoretical compute throughput
- **Tensor Core performance**: Specialized matrix multiplication units
- **Interconnect bandwidth**: NVLink, NVSwitch for multi-GPU communication

#### Memory hierarchy

- High bandwidth memory (HBM): Primary GPU memory
- L2 cache: Shared across streaming multiprocessors
- Shared memory: Fast on-chip memory per SM
- Registers: Fastest, per-thread storage

### 9.1.2 Tensor processing units

Google's custom AI accelerators [78]:

#### TPU generations

- **TPU v1:** Inference-only, INT8 focus
- **TPU v2:** Training support, 16GB HBM, 45 TFLOPS (BF16)
- **TPU v3:** 32GB HBM, 123 TFLOPS (BF16), water cooling
- **TPU v4:** 32GB HBM2, 275 TFLOPS (BF16), 3D torus topology
- **TPU v5e:** Cost-optimized for inference
- **TPU v5p:** 95GB HBM, 459 TFLOPS (BF16)

#### TPU architecture features

- Matrix multiply unit (MXU): Systolic array architecture
- BFloat16 native support
- High-speed inter-chip interconnect (ICI)
- Optimized for JAX/TensorFlow

### 9.1.3 Alternative accelerators

#### AMD Instinct

- MI250X: 128GB HBM2e, 383 TFLOPS (FP16)
- MI300X: 192GB HBM3, 1307 TFLOPS (FP16)
- ROCm software stack
- Growing adoption for LLM training

#### Intel accelerators

- Gaudi series: Purpose-built for deep learning
- Gaudi 2: 96GB HBM2e
- Gaudi 3: Enhanced performance and memory

#### Custom accelerators

- AWS Trainium: Custom training chips
- AWS Inferentia: Inference optimization
- Cerebras CS-2: Wafer-scale engine
- Graphcore IPU: Intelligence Processing Unit
- SambaNova: Reconfigurable dataflow architecture

The growth in training compute requirements has been exponential, as illustrated in [Figure 9.1](#).

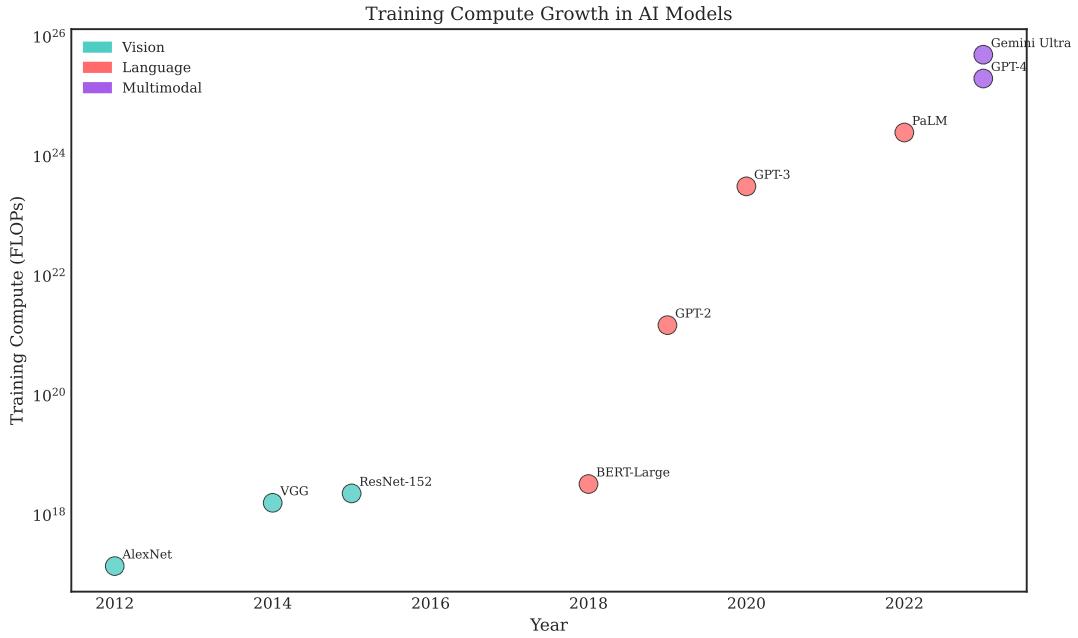


Figure 9.1: Training compute growth for AI models from 2012 to 2024, measured in FLOPs. Data for AlexNet through GPT-3 from Epoch AI database [79]; PaLM compute from Google technical report [39]. **Note:** GPT-4 and Gemini Ultra values are community estimates, as official figures have not been disclosed. The exponential increase reflects both growing model sizes and longer training runs on larger datasets.

### 9.1.4 Networking infrastructure

#### Intra-node connectivity

- **NVLink:** Up to 900 GB/s per GPU (4th gen)
- **NVSwitch:** All-to-all GPU communication within node
- **PCIe:** Gen 5 provides 128 GB/s per slot

#### Inter-node connectivity

- **InfiniBand:** HDR (200 Gb/s), NDR (400 Gb/s), XDR (800 Gb/s)
- **Ethernet:** 100/200/400 GbE, RoCE for RDMA
- **Custom interconnects:** Google ICI, AWS EFA

#### Network topology

- Fat-tree: High bisection bandwidth
- Torus: TPU pod configuration
- Dragonfly: Reduced cable cost with good performance
- Rail-optimized: Minimizes cross-rail traffic

## 9.2 Distributed training systems

### 9.2.1 Parallelism strategies

Training LLMs requires distributing computation across many devices. Multiple parallelism strategies address different bottlenecks:

**Data parallelism**

Replicate model across devices, partition data batches:

$$\text{Global Batch Size} = \text{Local Batch Size} \times \text{Number of Workers} \quad (9.1)$$

**Synchronous data parallelism:**

1. Each worker computes gradients on local batch
2. All-reduce gradients across workers
3. All workers apply same gradient update

**Asynchronous data parallelism:**

- Workers update parameters independently
- Reduces synchronization overhead
- Can lead to stale gradients, convergence issues

**Model parallelism (tensor parallelism)**

Partition individual layers across devices [75]:

- Split attention heads across GPUs
- Partition feed-forward layers
- Requires all-reduce for layer outputs
- Low latency requirements (high-bandwidth interconnect)

For a linear layer  $Y = XW$ :

$$Y = X[W_1|W_2] = [XW_1|XW_2] \quad (9.2)$$

Column-parallel and row-parallel patterns minimize communication.

**Pipeline parallelism**

Partition model layers across stages [80, 81]:

- Each stage handles subset of layers
- Micro-batches flow through pipeline
- Reduces memory per device
- Pipeline bubbles reduce efficiency

**Bubble overhead:**

$$\text{Bubble Fraction} = \frac{p - 1}{m} \quad (9.3)$$

where  $p$  is pipeline stages and  $m$  is micro-batches.

**Schedules:**

- GPipe: All-forward then all-backward
- 1F1B: Interleaved forward-backward (PipeDream)
- Interleaved 1F1B: Multiple stages per device
- Zero Bubble: Eliminates bubble overhead

### Sequence parallelism

Partition sequence dimension:

- Split long sequences across devices
- Ring attention: Efficient attention across partitions
- Reduces memory for very long sequences

### Expert parallelism

For mixture-of-experts models:

- Different experts on different devices
- All-to-all communication for token routing
- Combined with other parallelism strategies

### 3D parallelism

Combining data, tensor, and pipeline parallelism:

- Tensor parallelism within nodes (high bandwidth)
- Pipeline parallelism across nodes
- Data parallelism across pipeline replicas
- Typical configuration: TP=8, PP=4, DP=variable

## 9.2.2 Memory optimization

### Gradient checkpointing (activation recomputation)

Trade compute for memory:

- Store only subset of activations during forward
- Recompute activations during backward
- Typically checkpoint at layer boundaries
- Reduces memory proportional to checkpointing frequency

### ZeRO (Zero Redundancy Optimizer)

DeepSpeed's memory optimization [77]:

#### ZeRO Stage 1: Partition optimizer states

- Each rank stores  $1/N$  of optimizer states
- All-gather for parameter updates
- 4x memory reduction for optimizer states

#### ZeRO Stage 2: Add gradient partitioning

- Each rank stores  $1/N$  of gradients
- Reduce-scatter for gradient aggregation
- Additional memory savings

#### ZeRO Stage 3: Add parameter partitioning

- Each rank stores  $1/N$  of parameters

- All-gather for forward/backward
- Maximum memory efficiency
- Increased communication

**ZeRO-Offload:** Offload to CPU memory **ZeRO-Infinity:** Offload to NVMe storage

### Mixed precision training

Use lower precision for efficiency:

- FP16/BF16 for forward/backward computation
- FP32 master weights for updates
- Loss scaling for FP16 stability
- 2x memory reduction, higher throughput

$$\text{Master Weight Update: } W_{fp32} \leftarrow W_{fp32} - \eta \cdot g_{fp32} \quad (9.4)$$

**BFloat16 advantages:**

- Same exponent range as FP32
- Reduced mantissa (7 bits vs 23)
- No loss scaling needed
- Better stability for large models

### 9.2.3 Training frameworks

#### Megatron-LM

NVIDIA's large model training framework [76]:

- Efficient tensor parallelism implementation
- 1F1B pipeline scheduling
- Sequence parallelism
- Optimized for NVIDIA GPUs
- Used for training GPT-3 scale models

#### DeepSpeed

Microsoft's training optimization library:

- ZeRO optimizer family
- MoE support
- Compression and quantization
- Inference optimization
- Integration with Hugging Face Transformers

### FSDP (Fully Sharded Data Parallel)

PyTorch native sharding:

- Similar to ZeRO Stage 3
- Native PyTorch integration
- Composable with other PyTorch features
- Automatic mixed precision support

### JAX ecosystem

Google's functional ML framework:

- XLA compilation for optimization
- pjit for multi-device parallelism
- Functional transformations (vmap, pmap)
- T5X, Praxis for model development

### Colossal-AI

Efficient large model training:

- Automatic parallelism search
- Heterogeneous training strategies
- Low-code API

## 9.3 Communication optimization

### 9.3.1 Collective operations

#### All-reduce

Aggregate gradients across all workers:

- Ring all-reduce:  $O(2(N - 1)/N \cdot M)$  bandwidth
- Tree all-reduce:  $O(\log N)$  latency
- Hierarchical: Combine ring and tree

#### All-gather

Collect partitioned data:

- Used in ZeRO Stage 3
- Ring all-gather for efficiency

#### Reduce-scatter

Reduce and distribute:

- Paired with all-gather
- Gradient partitioning

**All-to-all**

Personalized exchange:

- MoE token routing
- Sequence parallelism resharding

**9.3.2 Communication-computation overlap****Async operations**

Overlap communication with computation:

- Start all-reduce during backward
- Overlap parameter fetch with forward
- Hide communication latency

**Gradient bucketing**

Group small tensors for efficient communication:

- Reduce kernel launch overhead
- Better bandwidth utilization
- Configurable bucket size

**9.3.3 Compression techniques****Gradient compression**

Reduce communication volume:

- Quantization: FP16, INT8
- Sparsification: Top-k gradients
- Error feedback for compensation
- Trade-off: Accuracy vs speedup

**9.4 Training stability and efficiency****9.4.1 Numerical stability****Loss scaling**

For FP16 training:

- Scale loss before backward
- Unscale gradients before update
- Dynamic scaling: Adjust based on overflow detection

**Gradient clipping**

Prevent exploding gradients:

$$g \leftarrow g \cdot \min \left( 1, \frac{\text{max\_norm}}{\|g\|} \right) \quad (9.5)$$

### Numerical precision considerations

- Attention softmax in FP32
- LayerNorm in FP32
- Accumulation in FP32
- Final loss computation in FP32

### 9.4.2 Learning rate scheduling

#### Warmup

Gradual learning rate increase:

- Linear warmup over initial steps
- Stabilizes early training
- Typical: 0.1-2% of total steps

#### Decay schedules

- Cosine: Smooth decay to minimum
- Linear: Constant decrease
- Inverse square root: Transformer default
- Step decay: Discrete reductions

#### Learning rate restarts

- Cosine annealing with warm restarts
- Can improve final performance

### 9.4.3 Checkpointing and fault tolerance

#### Regular checkpointing

Save training state periodically:

- Model weights
- Optimizer states
- Learning rate scheduler
- Random states
- Training metrics

#### Async checkpointing

Reduce checkpoint overhead:

- Background checkpoint writing
- Minimize training interruption
- Distributed file systems

**Fault tolerance**

Handle hardware failures:

- Elastic training: Handle worker changes
- Automatic restart from checkpoint
- Redundant job scheduling

## 9.5 Efficiency techniques

### 9.5.1 FlashAttention

Memory-efficient attention [28]:

- Fused attention kernel
- Tiling for memory efficiency
- IO-aware algorithm design
- 2-4x speedup, 5-20x memory reduction

**Key insight:** Avoid materializing full attention matrix in HBM.

Algorithm:

1. Load Q, K, V blocks to SRAM
2. Compute attention for block
3. Accumulate output incrementally
4. Write final output to HBM

### 9.5.2 Kernel fusion

Combine operations into single kernels:

- Reduce memory bandwidth requirements
- Lower kernel launch overhead
- Custom CUDA kernels
- Compiler-based fusion (XLA, Triton)

### 9.5.3 Operator optimization

#### Optimized attention

- Multi-query attention: Share KV across heads
- Grouped query attention: K groups of KV heads
- Sliding window attention: Limited context

#### Efficient embeddings

- Vocabulary partitioning
- Tied embeddings
- Embedding compression

## 9.6 Energy efficiency and sustainability

### 9.6.1 Carbon footprint

Training large models has significant environmental impact [82–84]:

#### Energy consumption

- GPT-3 training: 1,287 MWh estimated
- BLOOM training: 433 MWh measured
- Ongoing inference costs often exceed training

#### Carbon emissions

$$\text{CO}_2\text{e} = \text{Energy} \times \text{Carbon Intensity} \quad (9.6)$$

Depends on:

- Grid carbon intensity (varies by location)
- Renewable energy usage
- PUE (Power Usage Effectiveness)

### 9.6.2 Efficiency improvements

#### Hardware efficiency

- Newer GPUs more energy-efficient
- Specialized accelerators vs general-purpose
- Data center cooling optimization

#### Algorithmic efficiency

- Better architectures (MoE, efficient attention)
- Improved training recipes
- Smaller models with similar performance

#### Data center location

- Regions with renewable energy
- Time-of-day scheduling
- Carbon-aware job scheduling

### 9.6.3 Reporting and transparency

- Energy consumption disclosure
- Carbon footprint reporting
- Efficiency metrics (FLOPs/watt)
- Environmental impact assessments



# Chapter 10

## Data science for large language models

The quality and composition of training data fundamentally determine model capabilities, biases, and limitations. This chapter examines the data science practices underlying modern LLM development [17, 37? ].

### 10.1 Training data sources

#### 10.1.1 Web-scale text corpora

##### Common Crawl

The largest publicly available web crawl:

- Petabytes of web data collected since 2008
- Monthly crawls capturing billions of pages
- Raw HTML, extracted text, and metadata
- Requires extensive filtering and cleaning
- Forms the foundation of most LLM training sets

##### Derived datasets

Processed versions of Common Crawl:

- **C4 (Colossal Clean Crawled Corpus):** 750GB cleaned English text from Common Crawl [17]
- **OSCAR:** Multilingual corpus with language classification
- **CC-100:** 100+ languages extracted from Common Crawl
- **mC4:** Multilingual extension of C4

##### RefinedWeb

High-quality web text through aggressive filtering:

- URL filtering against blocklists
- Document-level heuristics
- Line-level filtering
- Deduplication at multiple levels
- Demonstrated that filtered web data alone can match curated corpora

### 10.1.2 Curated text sources

#### Books

- **Books1/Books2:** Internet book collections used in GPT-3
- **BookCorpus:** 11,000 unpublished books
- **Project Gutenberg:** Public domain books
- Provides coherent, long-form narrative structure
- Exposes models to diverse writing styles and topics

#### Wikipedia

- High-quality encyclopedic content
- Multiple languages available
- Regular updates and version snapshots
- Structured knowledge representation
- Commonly used in multilingual training

#### Academic and scientific text

- **Semantic Scholar corpus:** Academic papers and metadata
- **arXiv:** Scientific preprints in physics, math, CS
- **PubMed:** Biomedical literature
- **S2ORC:** 81M academic papers with parsed content
- Critical for scientific reasoning capabilities

### 10.1.3 Code repositories

#### GitHub

- Billions of lines of public source code
- Multiple programming languages
- Associated documentation and comments
- Issue discussions and pull requests
- License filtering required for compliance

#### The Stack

Curated code dataset for LLM training:

- 6.4TB of permissively licensed code
- 358 programming languages
- Near-deduplication applied
- Opt-out mechanism for developers
- Used for training StarCoder models

### Code quality considerations

- License compatibility (MIT, Apache, BSD)
- Filtering low-quality or generated code
- Preserving natural code distribution
- Including documentation and tests
- Balancing language distribution

#### 10.1.4 Conversational and instructional data

##### Web forums and discussions

- Reddit conversations ([pushshift.io archive](#))
- Stack Overflow Q&A pairs
- Internet forums across domains
- Comment threads and discussions
- Captures natural dialogue patterns

##### Instruction datasets

- **FLAN**: 1836 tasks in instruction format [\[48\]](#)
- **Natural Instructions**: 1600+ NLP tasks with instructions
- **Super-Natural Instructions**: Extended instruction collection
- **OpenAssistant**: Human-generated conversations
- **Dolly**: Instruction-following demonstrations

## 10.2 Data curation and preprocessing

### 10.2.1 Quality filtering

#### Heuristic filters

Rule-based quality indicators:

- Document length thresholds (minimum/maximum)
- Character-level statistics (alphabetic ratio, special characters)
- Word-level statistics (average word length, vocabulary diversity)
- Punctuation and formatting patterns
- Language identification confidence

#### Classifier-based filtering

Using trained models for quality assessment:

- Binary classifiers trained on high-quality reference data
- Wikipedia or curated text as positive examples
- Random web text as negative examples
- GPT-2 output detector for synthetic content
- Perplexity-based filtering using reference models

### Content-based filtering

Removing problematic content:

- Toxicity and hate speech classifiers
- Adult content detection
- Personal information identification (PII)
- Malware and spam detection
- Boilerplate and template removal

### 10.2.2 Deduplication

Deduplication is critical for preventing memorization and ensuring data efficiency [85].

#### Exact deduplication

- Document-level hash matching
- Line-level exact matching
- URL-based deduplication
- Effective but misses near-duplicates

#### Near-deduplication

- **MinHash LSH**: Approximate set similarity using hash signatures
- **SimHash**: Locality-sensitive hashing for documents
- **Suffix arrays**: Detecting repeated substrings
- **N-gram fingerprinting**: Identifying similar text spans

#### Deduplication scope

- Document-level: Remove entire duplicate documents
- Paragraph-level: Remove repeated paragraphs
- Sequence-level: Remove repeated n-grams
- Cross-document: Identify near-duplicate documents

The impact of deduplication:

- Reduces training compute by removing redundant data
- Decreases memorization of specific passages
- Improves sample efficiency during training
- Can remove 30-50% of raw web data

### 10.2.3 Data mixing and sampling

#### Domain weighting

Different data sources contribute differently to capabilities:

- Code improves reasoning and structured output
- Books improve long-context understanding
- Wikipedia provides factual knowledge
- Conversations improve instruction following
- Domain weights significantly impact final model behavior

### Mixing strategies

- **Proportional sampling:** Sample according to dataset size
- **Temperature sampling:** Adjust sampling probabilities
- **Domain upsampling:** Increase weight of high-quality sources
- **Curriculum learning:** Order data by difficulty

The Llama 2 training mix [38]:

- Undisclosed exact proportions
- Increased code proportion vs Llama 1
- Upsampling of high-quality sources
- Longer training with more tokens seen multiple times

### Data epochs

- Most LLMs train for less than 2 epochs on unique data
- High-quality data may be seen multiple times
- Diminishing returns from repeated data
- Trade-off between data diversity and compute efficiency

## 10.3 Synthetic data generation

Synthetic data has become increasingly important for LLM training, particularly for specialized capabilities.

### 10.3.1 Instruction generation

#### Self-Instruct

Using LLMs to generate training data [86]:

- Seed set of human-written instructions
- LLM generates new instructions
- Filter for diversity and quality
- Generate outputs for valid instructions
- Iteratively expand the dataset

#### Evol-Instruct

Evolving instructions for complexity:

- Start with simple instructions
- Apply evolution operators (add constraints, deepen, concretize)
- Generate increasingly complex variations
- Used for WizardLM training

### 10.3.2 Reasoning chain synthesis

#### Chain-of-thought generation

- Prompt strong models to show reasoning steps
- Generate solutions for mathematical problems
- Create step-by-step explanations
- Filter for correctness using final answers

#### Rejection sampling for reasoning

- Generate multiple solutions per problem
- Verify correctness against ground truth
- Keep only correct reasoning chains
- Increases data quality for math/coding training

### 10.3.3 Code generation

#### Code synthesis

- Generate code from natural language descriptions
- Create test cases and verify correctness
- Generate documentation and comments
- Create variations of existing code

#### OSS-Instruct

Open-source software instruction generation:

- Extract code snippets from real repositories
- Generate natural language descriptions
- Create instruction-response pairs
- Used for Magicoder training

### 10.3.4 Quality and limitations

Synthetic data considerations:

- Model collapse: Training on model outputs can degrade performance
- Distribution shift from human-generated text
- Amplification of model biases
- Verification difficulty for complex outputs
- Diminishing returns from synthetic data alone

Best practices:

- Mix synthetic with human-generated data
- Use strong teacher models for generation
- Apply rigorous quality filtering
- Verify correctness where possible (math, code)
- Monitor for distribution shift

## 10.4 Evaluation data contamination

Benchmark contamination is a significant concern in LLM evaluation [87].

### 10.4.1 Sources of contamination

- Web crawl overlap: Benchmark data present in training corpora
- Temporal leakage: Test sets published before training cutoff
- Paraphrase overlap: Near-duplicates in training data
- Indirect exposure: Discussion of benchmarks in training data

### 10.4.2 Detection methods

#### N-gram overlap

- Compute n-gram overlap between training and test sets
- Typically use 8-13 gram sequences
- Flag documents exceeding overlap threshold
- Simple but effective baseline

#### Membership inference

- Use model confidence as contamination signal
- Compare perplexity on test vs held-out data
- Statistical tests for memorization

### 10.4.3 Mitigation strategies

- Pre-training decontamination: Remove benchmark-overlapping data
- Dynamic benchmarks: Create new evaluation sets post-training
- Private held-out sets: Keep evaluation data unpublished
- Canary strings: Embed detectable markers in benchmarks
- Multi-shot evaluation: Use variations not in training data

## 10.5 Multilingual data

### 10.5.1 Language distribution

Web data is heavily skewed toward English:

- English: 40-60% of web text
- European languages: 20-30%
- Asian languages: 10-20%
- Low-resource languages: less than 1%

### 10.5.2 Multilingual corpora

- **mC4**: 101 languages from Common Crawl
- **ROOTS**: 1.6TB of text in 46 languages (BLOOM training)
- **CulturaX**: 6.3 trillion tokens in 167 languages
- **CC-100**: 100+ language corpus from Common Crawl

### 10.5.3 Challenges

- Quality filtering harder for low-resource languages
- Language identification errors
- Code-switching and transliteration
- Romanization of non-Latin scripts
- Tokenizer efficiency varies by language

### 10.5.4 Cross-lingual transfer

Multilingual training enables:

- Zero-shot transfer to unseen languages
- Improved performance on low-resource languages
- Shared representations across languages
- Translation capabilities as emergent behavior

## 10.6 Legal and ethical considerations

### 10.6.1 Copyright and licensing

- Training on copyrighted text without permission
- Fair use doctrine applicability (contested)
- License compliance for code (GPL, MIT, etc.)
- Opt-out mechanisms for data subjects
- Ongoing litigation (Authors Guild, Getty Images, etc.)

### 10.6.2 Privacy concerns

- Personal information in training data
- GDPR right to erasure challenges
- Memorization of private data
- De-identification requirements
- Consent for data use

### 10.6.3 Bias and representation

Training data reflects and amplifies societal biases:

- Gender and racial stereotypes in text
- Geographic and cultural bias toward Western content
- Temporal bias toward recent information
- Socioeconomic bias in internet access
- Language bias toward majority languages

### 10.6.4 Documentation practices

#### Data cards and datasheets

- Structured documentation of dataset composition
- Collection methodology and sources
- Known limitations and biases
- Intended use cases
- Maintenance and versioning information

#### Model cards

- Training data description
- Performance across demographic groups
- Limitations and failure modes
- Ethical considerations
- Recommended use cases

## 10.7 Data infrastructure

### 10.7.1 Storage and processing

- Distributed file systems (HDFS, S3, GCS)
- Efficient data formats (Parquet, Arrow, TFRecord)
- Streaming data pipelines
- On-the-fly preprocessing vs pre-computed
- Caching strategies for repeated training

### 10.7.2 Data loading

Efficient data loading critical for training throughput:

- Prefetching and pipelining
- Parallel data loading across workers
- Memory-mapped file access
- Tokenization caching
- Shuffling strategies for randomization

### 10.7.3 Preprocessing pipelines

Common pipeline components:

1. Raw data ingestion
2. Language identification
3. Quality filtering
4. Deduplication
5. PII removal
6. Domain classification
7. Tokenization
8. Packing into training sequences

Tools and frameworks:

- **Datatrue**: Hugging Face's data processing library
- **Dolma**: AI2's preprocessing toolkit
- **CCNet**: Facebook's Common Crawl processing pipeline
- Apache Spark for distributed processing
- Ray for scalable Python processing

# Chapter 11

## Deployment and MLOps for LLMs

Deploying large language models in production requires specialized infrastructure, optimization techniques, and operational practices distinct from traditional machine learning systems [88].

### 11.1 Inference optimization

#### 11.1.1 Quantization

Reducing numerical precision to decrease memory and compute requirements.

##### Post-training quantization (PTQ)

Quantizing weights after training:

- **INT8 quantization:**  $4\times$  memory reduction with minimal quality loss
- **INT4 quantization:**  $8\times$  memory reduction, noticeable quality trade-off
- **FP16/BF16:**  $2\times$  reduction, typically no quality loss
- Round-to-nearest or learned rounding
- Calibration using representative data samples

##### Quantization-aware training (QAT)

Incorporating quantization during training:

- Simulated quantization during forward pass
- Straight-through estimator for gradients
- Better quality than PTQ at same precision
- Higher training cost

##### Advanced quantization methods

- **GPTQ:** One-shot weight quantization using approximate second-order information
- **AWQ (Activation-aware Weight Quantization):** Preserves salient weights based on activation patterns
- **SqueezeLLM:** Sensitivity-based non-uniform quantization
- **GGML/GGUF:** Quantization formats for CPU inference
- **ExLlama:** Optimized 4-bit inference for consumer GPUs

### 11.1.2 Key-value cache optimization

KV cache stores attention keys and values for efficient autoregressive generation.

#### Memory requirements

For a model with  $L$  layers,  $H$  heads,  $d$  head dimension, and sequence length  $S$ :

$$\text{KV cache size} = 2 \times L \times H \times d \times S \times \text{bytes per element} \quad (11.1)$$

For a 70B parameter model with 128k context in FP16:

- Approximately 40GB of KV cache per sequence
- Often exceeds model weights in memory
- Primary bottleneck for long-context inference

#### Optimization techniques

- **KV cache quantization:** INT8 or INT4 cache values
- **Paged attention:** Virtual memory-style KV cache management
- **Sliding window attention:** Bounded cache size
- **KV cache compression:** Learned compression of cached values
- **Speculative caching:** Prefetching likely continuations

### 11.1.3 Batching strategies

#### Static batching

- Fixed batch size throughout generation
- Simple implementation
- Inefficient: short sequences wait for longest
- Underutilizes GPU for variable-length requests

#### Continuous batching (iteration-level batching)

- New requests join batch as others complete
- Maximizes GPU utilization
- Reduces latency for short requests
- Used by vLLM, TGI, and other modern serving systems

#### Chunked prefill

- Split long prompts into chunks
- Interleave prefill with generation
- Prevents head-of-line blocking
- Maintains consistent latency

### 11.1.4 Speculative decoding

Using a smaller draft model to accelerate generation:

- Draft model generates  $k$  candidate tokens
- Target model verifies in parallel
- Accept matching prefix, regenerate from divergence
- 2-3 $\times$  speedup with no quality loss
- Effective when draft model is much faster than target

#### Variants

- **Self-speculative**: Use target model's early exit
- **Medusa**: Multiple prediction heads on single model
- **EAGLE**: Extra auto-regressive head
- **Lookahead**: N-gram based speculation

## 11.2 Serving infrastructure

### 11.2.1 Inference servers

#### vLLM

High-throughput serving with PagedAttention [89]:

- Paged attention for efficient KV cache management
- Continuous batching
- Tensor parallelism support
- OpenAI-compatible API
- Supports most popular model architectures

#### Text Generation Inference (TGI)

Hugging Face's production inference server:

- Optimized transformer inference
- Flash Attention integration
- Quantization support
- Token streaming
- Watermarking support

#### TensorRT-LLM

NVIDIA's optimized inference library:

- Deep integration with NVIDIA GPUs
- Kernel fusion and optimization
- In-flight batching
- Quantization and pruning support
- Multi-GPU and multi-node inference

### Other frameworks

- **llama.cpp**: CPU and Apple Silicon inference
- **Ollama**: Local LLM deployment
- **MLC-LLM**: Universal deployment on diverse hardware
- **CTranslate2**: Efficient transformer inference
- **DeepSpeed-Inference**: Microsoft's inference optimization

### 11.2.2 API design

#### Synchronous APIs

- Request-response pattern
- Simple to implement and use
- Client blocks until completion
- High latency for long generations

#### Streaming APIs

- Server-sent events (SSE) or WebSockets
- Tokens sent as generated
- Improved perceived latency
- Enables early termination
- More complex client implementation

#### OpenAI API compatibility

De facto standard for LLM APIs:

- Chat completions endpoint
- Structured request/response format
- Tool/function calling support
- Usage tracking (tokens)
- Most serving frameworks provide compatibility

### 11.2.3 Load balancing and scaling

#### Horizontal scaling

- Multiple inference instances
- Request routing and load balancing
- Stateless design for easy scaling
- Kubernetes-based orchestration

#### Model parallelism for serving

- **Tensor parallelism**: Split layers across GPUs
- **Pipeline parallelism**: Split model stages across GPUs
- Trade-off between parallelism and communication overhead
- Typically 2-8 GPU tensor parallelism for large models

### Auto-scaling

- Scale based on queue depth or latency
- GPU utilization monitoring
- Predictive scaling for known patterns
- Cost optimization during low demand

## 11.3 Operational practices

### 11.3.1 Monitoring and observability

#### Key metrics

- **Throughput:** Tokens per second, requests per second
- **Latency:** Time to first token (TTFT), inter-token latency (ITL), total latency
- **GPU utilization:** Compute and memory usage
- **Queue depth:** Pending requests
- **Error rates:** Timeouts, OOM, generation failures

#### Logging

- Request/response logging (with PII considerations)
- Token usage tracking
- Generation parameters
- Model version and configuration
- Client identification

#### Distributed tracing

- End-to-end request tracking
- Latency breakdown by component
- Integration with observability platforms
- Correlation with downstream services

### 11.3.2 Cost management

#### GPU cost optimization

- Spot/preemptible instances for batch workloads
- Reserved capacity for baseline load
- Right-sizing GPU instances
- Multi-tenancy and sharing

#### Token economics

- Input vs output token pricing
- Prompt caching for repeated prefixes
- Response length limits
- Tiered pricing by model capability

### Cost allocation

- Per-user or per-team tracking
- Project-based cost centers
- Usage quotas and limits
- Chargeback systems

#### 11.3.3 Security considerations

##### Input validation

- Prompt injection detection
- Input length limits
- Rate limiting per client
- Content policy enforcement

##### Output filtering

- Toxicity and safety classifiers
- PII detection and redaction
- Watermarking for traceability
- Content moderation pipelines

##### Access control

- API key management
- OAuth/OIDC integration
- Role-based access control
- Audit logging

## 11.4 Model lifecycle management

### 11.4.1 Version control

- Model artifact versioning
- Configuration management
- Reproducible deployments
- Rollback capabilities

### 11.4.2 A/B testing and experimentation

- Traffic splitting between model versions
- Statistical significance testing
- User preference collection
- Gradual rollout strategies

### 11.4.3 Model updates

#### Blue-green deployments

- Two identical production environments
- Switch traffic after validation
- Instant rollback capability
- Higher resource requirements

#### Canary deployments

- Gradual traffic migration
- Monitor for regressions
- Automatic rollback on failures
- Lower risk than full deployment

#### Shadow deployments

- New model receives copy of traffic
- Responses not served to users
- Compare outputs with production
- Validate before any user impact

## 11.5 Edge and on-device deployment

### 11.5.1 Mobile deployment

- Smaller models (1-7B parameters)
- Quantization to INT4 or lower
- Platform-specific optimizations (Core ML, NNAPI)
- Privacy-preserving local inference
- Battery and thermal constraints

### 11.5.2 Browser deployment

- WebAssembly and WebGPU
- Model download and caching
- JavaScript inference runtimes
- Limited compute compared to native
- Transformers.js for in-browser inference

### 11.5.3 Embedded systems

- Resource-constrained environments
- Real-time requirements
- Specialized hardware (NPUs)
- Offline operation
- Models optimized for specific tasks

## 11.6 Retrieval-augmented generation deployment

### 11.6.1 Vector database integration

- **Pinecone**: Managed vector database
- **Weaviate**: Open-source vector search
- **Milvus**: Scalable vector database
- **Chroma**: Simple embedding database
- **pgvector**: PostgreSQL extension

### 11.6.2 RAG pipeline components

- Document chunking strategies
- Embedding model selection and serving
- Retrieval and reranking
- Context assembly
- Citation and attribution

### 11.6.3 Operational considerations

- Index updates and freshness
- Embedding model versioning
- Cache invalidation
- Hybrid search (dense + sparse)
- Latency budgets for retrieval

## 11.7 Fine-tuning operations

### 11.7.1 Training infrastructure

- GPU cluster management
- Distributed training orchestration
- Checkpoint management
- Experiment tracking (MLflow, Weights & Biases)

### 11.7.2 Continuous fine-tuning

- Automated retraining pipelines
- Data drift detection
- Performance regression testing
- Model registry integration

### 11.7.3 PEFT deployment

Parameter-efficient fine-tuned models require special handling:

- LoRA adapter hot-swapping
- Multiple adapters per base model
- Adapter versioning and management
- Dynamic adapter loading



# Chapter 12

## Industry landscape and economics

The development and deployment of large language models has created a rapidly evolving industry with significant economic implications spanning infrastructure, services, and applications.

### 12.1 Major industry players

#### 12.1.1 Frontier model developers

##### OpenAI

- Founded 2015 as non-profit, transitioned to capped-profit 2019
- GPT series: GPT-2 (2019), GPT-3 (2020), GPT-4 (2023), GPT-4o (2024)
- ChatGPT: Fastest-growing consumer application in history
- DALL-E for image generation, Whisper for speech
- Microsoft partnership and investment (\$13B+)
- Estimated valuation: \$80-100B (2024)

##### Anthropic

- Founded 2021 by former OpenAI researchers
- Focus on AI safety and constitutional AI
- Claude series: Claude 1, Claude 2, Claude 3 (Haiku, Sonnet, Opus)
- Amazon and Google investments (\$6B+)
- Emphasis on interpretability and alignment research

##### Google DeepMind

- Merger of Google Brain and DeepMind (2023)
- Gemini multimodal models
- PaLM and PaLM 2 for text
- Gemma open-weight models
- Integration across Google products
- Significant compute advantage from TPU infrastructure

### Meta AI

- Open-weight model strategy with Llama series
- Llama, Llama 2, Llama 3 releases
- Research publications and open science approach
- Integration with Meta products (WhatsApp, Instagram, Facebook)
- PyTorch framework development

### Other frontier labs

- **Mistral AI:** French startup, strong open-weight models
- **Cohere:** Enterprise-focused, Canadian company
- **AI21 Labs:** Israeli company, Jurassic models
- **Inflection AI:** Pi assistant, later acquired talent by Microsoft
- **xAI:** Elon Musk's company, Grok models

## 12.1.2 Cloud providers

### Microsoft Azure

- Azure OpenAI Service: Enterprise GPT access
- Exclusive cloud partner for OpenAI
- Copilot integration across Microsoft 365
- GitHub Copilot for code generation
- Deep integration with enterprise software

### Amazon Web Services

- Amazon Bedrock: Multi-model API service
- Partnerships with Anthropic, AI21, Cohere, Meta
- Custom Titan models
- SageMaker for model training and deployment
- Trainium and Inferentia custom chips

### Google Cloud

- Vertex AI platform
- Gemini API and model garden
- TPU access for training
- Enterprise AI solutions
- Workspace AI integrations

### 12.1.3 Hardware providers

#### NVIDIA

- Dominant GPU provider for AI training
- H100, H200, Blackwell architectures
- CUDA ecosystem lock-in
- Data center revenue growth: 200%+ YoY (2023-2024)
- Market cap exceeded \$3 trillion (2024)
- NVLink and networking infrastructure

#### AMD

- MI300X as H100 competitor
- ROCm software stack development
- Growing adoption in hyperscalers
- Price-competitive positioning

#### Intel

- Gaudi accelerators (Habana Labs acquisition)
- CPU inference optimization
- Foundry services for AI chips
- Catching up in AI accelerator market

#### Custom silicon

- Google TPUs: Internal and cloud availability
- Amazon Trainium/Inferentia: AWS custom chips
- Microsoft Maia: Custom AI accelerator
- Tesla Dojo: Training computer for autonomous driving
- Cerebras, Graphcore, SambaNova: AI chip startups

## 12.2 Economic analysis

### 12.2.1 Training costs

#### Compute costs

Estimated training costs for frontier models:

- GPT-3 (175B, 2020): \$4-12 million
- GPT-4 (2023): \$50-100+ million estimated
- Gemini Ultra: \$100+ million estimated
- Llama 2 70B: \$2-5 million estimated
- Costs doubling approximately every 6-9 months

### Cost components

- GPU/TPU compute: 60-80% of total
- Data acquisition and processing: 5-15%
- Human annotation and RLHF: 5-15%
- Engineering and research personnel: 10-20%
- Infrastructure and overhead: 5-10%

### Scaling projections

- Training costs growing 4-5× per model generation
- \$1 billion training runs expected by 2025-2026
- Efficiency improvements partially offset cost growth
- Data limitations may constrain scaling

## 12.2.2 Inference economics

### Cost per token

Typical API pricing (2024):

- GPT-4 Turbo: \$10/1M input tokens, \$30/1M output tokens
- GPT-3.5 Turbo: \$0.50/1M input, \$1.50/1M output
- Claude 3 Opus: \$15/1M input, \$75/1M output
- Claude 3 Sonnet: \$3/1M input, \$15/1M output
- Llama 3 70B (self-hosted): \$0.50-2/1M tokens

### Cost optimization

- Prompt caching: Reduce repeated computation
- Batch processing: Improve GPU utilization
- Model selection: Use smaller models when sufficient
- Quantization: Reduce memory and compute
- Output length limits: Control costs

### Unit economics

- Gross margins: 50-70% for API providers
- Infrastructure costs declining 20-30% annually
- Competition driving price reductions
- Volume growth offsetting per-unit revenue decline

### 12.2.3 Market sizing

#### Current market

Generative AI market estimates (2024):

- Total market: \$40-60 billion
- LLM APIs and services: \$10-15 billion
- Enterprise software integration: \$15-20 billion
- Consumer applications: \$5-10 billion
- Infrastructure and tooling: \$10-15 billion

#### Growth projections

- 30-50% CAGR through 2030
- \$200-500 billion market by 2030 (various estimates)
- Enterprise adoption as primary growth driver
- Uncertainty around AGI timeline impacts

## 12.3 Business models

### 12.3.1 API-as-a-service

- Pay-per-token pricing
- Tiered plans with volume discounts
- Enterprise agreements with SLAs
- Free tiers for developer acquisition
- Examples: OpenAI API, Anthropic API, Google AI

### 12.3.2 Platform and infrastructure

- Model hosting and serving
- Fine-tuning platforms
- MLOps tools
- Vector databases
- Examples: AWS Bedrock, Azure OpenAI, Hugging Face

### 12.3.3 Vertical applications

- Industry-specific solutions
- Workflow automation
- Domain-specialized models
- Professional services
- Examples: Harvey (legal), Jasper (marketing), Synthesia (video)

### 12.3.4 Open-source monetization

- Open-core model (open base, proprietary extensions)
- Support and consulting services
- Managed hosting of open models
- Enterprise licensing with additional features
- Examples: Hugging Face, Together AI, Anyscale

## 12.4 Competitive dynamics

### 12.4.1 Moats and differentiation

#### Potential moats

- **Data:** Proprietary training data and user feedback
- **Talent:** Scarce ML research and engineering expertise
- **Compute:** Access to large-scale training infrastructure
- **Distribution:** Existing user base and platform integration
- **Brand:** Trust and reputation for quality/safety

#### Moat durability

- Model capabilities rapidly commoditizing
- Open-source models narrowing gap to frontier
- Compute becoming more accessible
- Talent pool expanding globally
- Distribution may be strongest long-term moat

### 12.4.2 Open vs closed models

#### Closed model advantages

- Revenue generation through API access
- Control over model use and safety
- Competitive advantage protection
- Liability management

#### Open model advantages

- Community contributions and improvements
- Trust through transparency
- Ecosystem development
- Talent attraction
- Commoditization of competitors' offerings

### Hybrid approaches

- Open weights with restricted commercial use
- Delayed open release (research preview then open)
- Open small models, closed large models
- Open architecture, closed training data

### 12.4.3 Consolidation trends

- Acqui-hires of AI talent (Inflection to Microsoft)
- Startup acquisitions by tech giants
- Partnership/investment structures (Microsoft-OpenAI)
- Vertical integration in hardware-software
- Model provider consolidation expected

## 12.5 Investment landscape

### 12.5.1 Venture capital

- Record funding for AI startups: \$50B+ in 2023
- Mega-rounds common (\$100M+ raises)
- High valuations despite limited revenue
- Focus on infrastructure and applications
- Geographic concentration in US, with growth in Europe and Asia

### 12.5.2 Corporate investment

- Microsoft: \$13B+ in OpenAI
- Google: \$2B+ in Anthropic
- Amazon: \$4B in Anthropic
- Salesforce, Oracle, others: Various AI investments
- Strategic rationale: Access to models, talent, technology

### 12.5.3 Public markets

- NVIDIA market cap surge (10× from 2022-2024)
- Microsoft, Google, Meta AI-driven valuation increases
- AI pure-play IPOs limited but expected
- Investor focus on AI exposure across sectors

## 12.6 Regulatory and policy environment

### 12.6.1 Emerging regulations

#### European Union

- EU AI Act: Risk-based regulatory framework
- GPAI (General Purpose AI) provisions
- Transparency requirements for foundation models
- High-risk application restrictions
- Extraterritorial application

#### United States

- Executive Order on AI Safety (October 2023)
- Voluntary commitments from leading labs
- NIST AI Risk Management Framework
- Sector-specific regulations (healthcare, finance)
- State-level initiatives (California, Colorado)

#### China

- Generative AI regulations (2023)
- Content moderation requirements
- Registration and approval processes
- Data localization requirements
- Technology export restrictions

#### Other jurisdictions

- UK: Pro-innovation approach, AI Safety Institute
- Canada: AIDA (Artificial Intelligence and Data Act)
- Japan: Light-touch regulation, copyright clarification
- Singapore: Model AI Governance Framework

### 12.6.2 Policy debates

#### Safety and risk

- Existential risk from advanced AI
- Near-term harms (misinformation, bias, misuse)
- Dual-use concerns (bioweapons, cyber)
- Appropriate precautionary measures
- International coordination challenges

**Competition and antitrust**

- Compute concentration concerns
- Data access and barriers to entry
- Vertical integration scrutiny
- Open-source as competitive remedy
- Platform power extension to AI

**Intellectual property**

- Training data copyright questions
- Model output ownership
- Patent implications
- Trade secret protection
- International IP harmonization

## 12.7 Workforce and labor impacts

### 12.7.1 Job displacement concerns

- Knowledge worker automation potential
- Customer service and support roles
- Content creation and writing
- Coding and software development
- Data analysis and research

### 12.7.2 Job creation

- Prompt engineers and AI specialists
- AI safety and alignment researchers
- ML engineers and data scientists
- AI product managers
- AI ethics and policy roles

### 12.7.3 Skill transformation

- AI literacy as baseline requirement
- Human-AI collaboration skills
- Critical evaluation of AI outputs
- Domain expertise increasingly valuable
- Continuous learning imperative

#### 12.7.4 Productivity effects

- Studies showing 20-50% productivity gains for some tasks
- Heterogeneous effects across occupations
- Quality improvements alongside speed
- Skill compression (novices improve more)
- Long-term macroeconomic effects uncertain

### 12.8 Geopolitical dimensions

#### 12.8.1 US-China competition

- Export controls on advanced chips (A100, H100)
- Chinese model development (Baidu, Alibaba, ByteDance)
- Talent competition and visa policies
- Data governance divergence
- Military and intelligence applications

#### 12.8.2 Technology sovereignty

- European efforts to develop indigenous capabilities
- Middle East investments in AI (UAE, Saudi Arabia)
- India's AI development initiatives
- Concerns about dependence on US technology
- Open-source as sovereignty strategy

#### 12.8.3 International cooperation

- AI Safety Summits (UK, South Korea)
- G7 Hiroshima AI Process
- OECD AI Principles
- UN discussions on AI governance
- Bilateral agreements on AI safety

# Chapter 13

## Retrieval-augmented generation

Retrieval-Augmented Generation (RAG) represents a paradigm shift in how large language models access and utilize external knowledge [90]. By combining parametric knowledge stored in model weights with non-parametric retrieval from external corpora, RAG systems address fundamental limitations of standalone LLMs including knowledge cutoffs, hallucination, and lack of source attribution.

### 13.1 Foundations of RAG

#### 13.1.1 Motivation and core concepts

Large language models encode knowledge in their parameters during pre-training, but this approach has inherent limitations:

- **Knowledge cutoff:** Models cannot access information created after training
- **Hallucination:** Models may generate plausible but incorrect information [91]
- **Opacity:** Difficult to trace the source of generated information
- **Update cost:** Retraining to update knowledge is prohibitively expensive
- **Domain specificity:** General models lack deep domain expertise

RAG addresses these challenges by retrieving relevant documents at inference time and conditioning generation on the retrieved context. The basic RAG formulation is:

$$p(y|x) = \sum_{d \in \text{top-}k} p(d|x) \cdot p(y|x, d) \quad (13.1)$$

where  $x$  is the query,  $y$  is the generated output,  $d$  represents retrieved documents,  $p(d|x)$  is the retrieval probability, and  $p(y|x, d)$  is the generation probability conditioned on the document.

#### 13.1.2 RAG architecture patterns

##### Naive RAG

The simplest RAG implementation follows a retrieve-then-read pipeline:

1. Index documents with embeddings
2. Retrieve top- $k$  similar documents for a query
3. Concatenate retrieved documents with the query
4. Generate response using the augmented context

## Advanced RAG

Improvements over naive RAG include [92]:

- Pre-retrieval optimization (query rewriting, expansion)
- Retrieval optimization (hybrid search, reranking)
- Post-retrieval processing (compression, filtering)

## Modular RAG

Modular architectures allow flexible combination of components [93]:

- Routing modules for query classification
- Multiple retrieval sources
- Iterative retrieval and generation
- Verification and fact-checking modules

## 13.2 Text embeddings and vector representations

### 13.2.1 Dense retrieval

Dense retrieval uses neural networks to encode queries and documents into dense vector representations [94]. Given a query  $q$  and document  $d$ , relevance is computed as:

$$\text{score}(q, d) = \text{sim}(E_q(q), E_d(d)) \quad (13.2)$$

where  $E_q$  and  $E_d$  are encoder functions and sim is typically cosine similarity or dot product.

#### Bi-encoder architecture

Bi-encoders encode queries and documents independently, enabling efficient retrieval through approximate nearest neighbor search:

- **DPR** (Dense Passage Retrieval) [94]: Dual BERT encoders for open-domain QA
- **Contriever** [95]: Unsupervised contrastive pre-training
- **E5** [96]: Text embeddings via contrastive learning
- **BGE** [97]: BAAI general embedding models

#### Cross-encoder architecture

Cross-encoders jointly encode query-document pairs for more accurate but slower scoring:

$$\text{score}(q, d) = \sigma(W \cdot \text{BERT}([q; d])) \quad (13.3)$$

Used primarily for reranking top candidates from bi-encoder retrieval.

### 13.2.2 Embedding models

State-of-the-art embedding models for RAG applications:

- **OpenAI text-embedding-3**: 3072-dimensional embeddings with strong performance across tasks
- **Cohere Embed v3**: Multilingual embeddings with compression support
- **Voyage AI**: Domain-specialized embeddings for code, legal, finance
- **Jina Embeddings v2**: 8192-token context window embeddings [98]
- **GTE** (General Text Embeddings): Open-source multilingual embeddings [99]

### 13.2.3 Sparse retrieval

Traditional sparse methods remain valuable, especially in hybrid systems:

#### BM25

The BM25 algorithm [100] computes relevance as:

$$\text{BM25}(q, d) = \sum_{t \in q} \text{IDF}(t) \cdot \frac{f(t, d) \cdot (k_1 + 1)}{f(t, d) + k_1 \cdot (1 - b + b \cdot \frac{|d|}{\text{avgdl}})} \quad (13.4)$$

where  $f(t, d)$  is term frequency,  $|d|$  is document length, and  $k_1, b$  are parameters.

#### SPLADE

Learned sparse representations using MLM [101]:

- Produces sparse, interpretable term weights
- Outperforms BM25 while maintaining efficiency
- Enables explicit term matching with learned importance

### 13.2.4 Hybrid retrieval

Combining dense and sparse retrieval captures complementary signals:

$$\text{score}_{\text{hybrid}} = \alpha \cdot \text{score}_{\text{dense}} + (1 - \alpha) \cdot \text{score}_{\text{sparse}} \quad (13.5)$$

Reciprocal Rank Fusion (RRF) provides a parameter-free alternative:

$$\text{RRF}(d) = \sum_{r \in R} \frac{1}{k + r(d)} \quad (13.6)$$

where  $r(d)$  is the rank of document  $d$  in retrieval result  $r$ , and  $k$  is a constant (typically 60).

## 13.3 Vector databases and indexing

### 13.3.1 Approximate nearest neighbor search

Exact nearest neighbor search is computationally prohibitive for large corpora. Approximate methods trade accuracy for speed:

#### Hierarchical Navigable Small World (HNSW)

HNSW [102] builds a multi-layer graph structure:

- Logarithmic search complexity:  $O(\log n)$
- High recall (>95%) with proper tuning
- Memory-intensive but very fast
- Used in Pinecone, Weaviate, Qdrant

#### Inverted File Index (IVF)

Partitions vector space into clusters:

- Searches only relevant partitions
- Lower memory than HNSW
- Used in FAISS [103]

### Product Quantization (PQ)

Compresses vectors by quantizing subspaces:

- Dramatically reduces memory requirements
- Some accuracy loss acceptable for large-scale search
- Often combined with IVF (IVF-PQ)

### 13.3.2 Vector database systems

Production vector databases for RAG deployments:

- **Pinecone**: Fully managed, serverless vector database
- **Weaviate**: Open-source with GraphQL interface and hybrid search
- **Qdrant**: Open-source with advanced filtering capabilities
- **Milvus**: Distributed vector database for billion-scale search [104]
- **Chroma**: Lightweight, developer-friendly for prototyping
- **pgvector**: PostgreSQL extension for vector similarity search
- **Elasticsearch**: Dense vector support with traditional search capabilities

### 13.3.3 Indexing strategies

#### Chunking strategies

Document segmentation significantly impacts retrieval quality:

- **Fixed-size chunking**: Simple but may split semantic units
- **Sentence-based**: Maintains sentence boundaries
- **Paragraph-based**: Preserves topical coherence
- **Semantic chunking**: Splits based on embedding similarity
- **Document structure-aware**: Uses headers, sections, lists

Typical chunk sizes range from 256 to 1024 tokens, with overlap of 10-20% to maintain context across boundaries.

#### Hierarchical indexing

Multi-level indexing for improved retrieval:

- Summary embeddings for document-level search
- Chunk embeddings for fine-grained retrieval
- Parent-child relationships for context expansion

## 13.4 Retrieval optimization

### 13.4.1 Query processing

#### Query rewriting

LLM-based query transformation improves retrieval:

- Expansion with synonyms and related terms
- Decomposition of complex queries
- Hypothetical document generation (HyDE) [105]

HyDE generates a hypothetical answer and uses it for retrieval:

1. LLM generates hypothetical document answering the query
2. Hypothetical document is embedded
3. Retrieval finds similar real documents

#### Query routing

Direct queries to appropriate knowledge sources:

- Classification-based routing
- Embedding similarity to source descriptions
- LLM-based intent detection

### 13.4.2 Reranking

Two-stage retrieval with reranking improves precision:

#### Cross-encoder reranking

- Retrieve top-100 with bi-encoder
- Rerank with cross-encoder to top-10
- Significantly improves precision at small  $k$

#### LLM-based reranking

Using LLMs to score relevance:

- Listwise ranking with LLMs
- Pairwise comparison aggregation
- RankGPT [106] and similar approaches

#### Learned rerankers

- **Cohere Rerank**: Production reranking API
- **BGE Reranker**: Open-source cross-encoder
- **ColBERT** [107]: Late interaction for efficient reranking

### 13.4.3 Multi-hop retrieval

Complex queries often require information from multiple documents:

- **Iterative retrieval:** Multiple retrieval rounds with refined queries
- **Self-RAG [93]:** Model decides when and what to retrieve
- **ITER-RETEGEN:** Interleaved retrieval and generation
- **Chain-of-retrieval:** Sequential retrieval for multi-step reasoning

## 13.5 RAG system design

### 13.5.1 Context construction

#### Context window management

Balancing retrieved content with context limits:

- Truncation strategies (recency, relevance)
- Compression using summarization
- Selective inclusion based on redundancy

#### Context ordering

Position of information affects LLM attention [108]:

- “Lost in the middle” phenomenon
- Most important content at beginning or end
- Recency bias in generation

### 13.5.2 Generation with retrieved context

#### Prompt engineering for RAG

Effective RAG prompts include:

- Clear instructions to use provided context
- Citation requirements for traceability
- Handling of insufficient information
- Conflict resolution between sources

#### Faithful generation

Ensuring outputs are grounded in retrieved content:

- Attribution training
- Citation generation
- Hallucination detection and mitigation

### 13.5.3 Evaluation of RAG systems

#### Retrieval metrics

- Recall@ $k$ : Fraction of relevant documents in top- $k$
- MRR (Mean Reciprocal Rank): Position of first relevant result
- NDCG (Normalized Discounted Cumulative Gain): Ranking quality

### Generation metrics

- Faithfulness: Consistency with retrieved content
- Answer relevance: Quality of response to query
- Context relevance: Quality of retrieved passages

### End-to-end evaluation

Frameworks for holistic RAG evaluation:

- **RAGAS** [109]: RAG Assessment framework
- **ARES**: Automated RAG evaluation
- **TruLens**: Tracing and evaluation toolkit

## 13.6 Advanced RAG techniques

### 13.6.1 Agentic RAG

Combining RAG with agent capabilities:

- Dynamic tool selection for retrieval
- Iterative refinement based on generation quality
- Multi-source aggregation
- Verification and fact-checking loops

### 13.6.2 Graph RAG

Incorporating knowledge graphs with vector retrieval [110]:

- Entity-centric retrieval
- Relationship traversal for multi-hop queries
- Community detection for topic summarization
- Hybrid vector-graph queries

### 13.6.3 Multimodal RAG

Extending RAG beyond text:

- Image retrieval with CLIP embeddings
- Table and chart understanding
- PDF and document parsing
- Video segment retrieval



# Chapter 14

## Agents and autonomous systems

Large language models are increasingly deployed as autonomous agents capable of planning, reasoning, and taking actions in complex environments [111]. This chapter examines the architectures, capabilities, and challenges of LLM-based agents.

### 14.1 Foundations of LLM agents

#### 14.1.1 Definition and components

An LLM agent consists of several core components [112]:

- **Brain:** The LLM serving as the central reasoning engine
- **Perception:** Mechanisms for receiving and processing inputs
- **Memory:** Short-term and long-term information storage
- **Action:** Capabilities for interacting with environments
- **Planning:** Strategic decomposition of complex tasks

#### 14.1.2 Agent architectures

##### ReAct

The ReAct framework [113] interleaves reasoning and acting:

1. **Thought:** Reasoning about the current situation
2. **Action:** Executing an operation (e.g., search, calculate)
3. **Observation:** Processing the result of the action
4. Repeat until task completion

This approach grounds reasoning in concrete actions and observations, reducing hallucination.

##### Plan-and-Execute

Separates planning from execution [114]:

1. Generate a high-level plan
2. Execute plan steps sequentially
3. Replan if execution fails

### Reflexion

Enables learning from mistakes through self-reflection [115]:

- Execute task with initial approach
- Evaluate outcome and identify failures
- Generate reflection on what went wrong
- Incorporate reflection in subsequent attempts

#### 14.1.3 Cognitive architectures

Drawing inspiration from cognitive science:

##### LATS (Language Agent Tree Search)

Combines Monte Carlo Tree Search with LLM reasoning [116]:

- Explores multiple reasoning paths
- Uses value function for path evaluation
- Backpropagates success signals

##### Cognitive Architectures for Language Agents (CoALA)

Systematic framework organizing agent components [117]:

- Working memory for current context
- Episodic memory for past experiences
- Semantic memory for world knowledge
- Procedural memory for learned skills

## 14.2 Memory systems

### 14.2.1 Short-term memory

Context window as working memory:

- Limited by model context length
- Recency bias affects attention
- Compression needed for long interactions

#### Memory compression

Strategies for managing limited context:

- Summarization of past interactions
- Selective retention of important information
- Hierarchical summarization

### 14.2.2 Long-term memory

External storage for persistent information:

**Vector-based memory**

- Store interaction history as embeddings
- Retrieve relevant past experiences
- Similar to RAG but for agent history

**Structured memory**

- Knowledge graphs for entity relationships
- SQL databases for structured data
- Key-value stores for quick lookup

**MemGPT**

Operating system-inspired memory management [118]:

- Virtual context management
- Automatic memory paging
- Hierarchical memory tiers

### 14.2.3 Memory retrieval and consolidation

- Recency-weighted retrieval
- Importance-based retention
- Periodic memory consolidation
- Forgetting mechanisms for efficiency

## 14.3 Tool use and function calling

### 14.3.1 Function calling interfaces

Modern LLMs support structured tool use:

**OpenAI Function Calling**

JSON-schema based function definitions:

- Structured output for tool invocation
- Parallel function calling
- Automatic argument extraction

**Anthropic Tool Use**

Claude's approach to tool integration:

- XML-based tool definitions
- Multi-turn tool use conversations
- Error handling and retry logic

### 14.3.2 Tool categories

#### Information retrieval tools

- Web search (Google, Bing APIs)
- Database queries
- API calls to external services
- Document retrieval systems

#### Computation tools

- Code interpreters (Python, JavaScript)
- Calculators and symbolic math
- Data analysis libraries

#### Action tools

- Email and messaging
- File system operations
- Browser automation
- API integrations

### 14.3.3 Tool learning

Enabling models to learn new tools:

#### Toolformer

Self-supervised tool use learning [119]:

- Model learns when and how to use tools
- Training on tool-augmented examples
- Automatic tool call insertion

#### Tool documentation understanding

- Learning from API documentation
- Few-shot tool adaptation
- Tool composition and chaining

## 14.4 Planning and reasoning

### 14.4.1 Task decomposition

Breaking complex tasks into manageable subtasks:

#### Hierarchical Task Networks

- Top-down task decomposition
- Subtask dependencies
- Parallel execution where possible

### Least-to-Most Prompting

Progressive problem decomposition [59]:

1. Identify subproblems
2. Solve from simplest to most complex
3. Build on previous solutions

### 14.4.2 Planning algorithms

#### LLM-Planner

Using LLMs for embodied planning [120]:

- Natural language goal specification
- Grounded action generation
- Dynamic replanning

#### Tree-of-Thoughts extended

Systematic exploration of reasoning paths:

- BFS/DFS search strategies
- Evaluation functions for path selection
- Backtracking on failure

### 14.4.3 Verification and error recovery

#### Self-verification

- Output checking against constraints
- Consistency verification
- Confidence estimation

#### Error recovery strategies

- Retry with modified approach
- Fallback to alternative methods
- Human escalation

## 14.5 Multi-agent systems

### 14.5.1 Agent collaboration patterns

#### Debate and discussion

Multiple agents argue different perspectives [121]:

- Improves reasoning through argumentation
- Reduces individual agent biases
- Consensus building mechanisms

### Role-based collaboration

Specialized agents for different functions:

- Planner, executor, critic roles
- Domain expert agents
- Coordinator agents

### Hierarchical organization

- Manager agents delegate to workers
- Task routing based on capability
- Aggregation of results

#### 14.5.2 Multi-agent frameworks

##### AutoGen

Microsoft's framework for multi-agent conversations [122]:

- Conversational agent patterns
- Human-in-the-loop integration
- Code execution capabilities

##### CrewAI

Role-based agent collaboration:

- Agent role definitions
- Task assignment and delegation
- Inter-agent communication

##### LangGraph

Graph-based agent orchestration:

- Stateful agent workflows
- Conditional routing
- Persistence and resumption

#### 14.5.3 Communication protocols

- Message passing between agents
- Shared memory spaces
- Broadcast and subscription patterns
- Negotiation protocols

## 14.6 Agent applications

### 14.6.1 Code agents

#### SWE-Agent

Autonomous software engineering [123]:

- Repository understanding
- Bug localization and fixing
- Test generation
- Code review automation

#### Devin and autonomous coding

End-to-end software development:

- Requirements to implementation
- Multi-file editing
- Debugging and testing

### 14.6.2 Research agents

- Literature search and synthesis
- Hypothesis generation
- Experimental design
- Paper writing assistance

### 14.6.3 Web agents

Browser automation with LLMs:

- Web navigation
- Form filling
- Information extraction
- Task automation

### 14.6.4 Personal assistants

- Email management
- Calendar scheduling
- Task tracking
- Information retrieval

## 14.7 Evaluation and benchmarks

### 14.7.1 Agent benchmarks

- **AgentBench** [124]: Multi-environment agent evaluation
- **WebArena**: Web browsing tasks
- **SWE-bench** [125]: Software engineering tasks
- **GAIA**: General AI assistant benchmark
- **ToolBench**: Tool use evaluation

### 14.7.2 Evaluation dimensions

- Task completion rate
- Efficiency (steps, tokens, time)
- Error recovery capability
- Generalization to new tasks
- Safety and constraint adherence

## 14.8 Safety and alignment for agents

### 14.8.1 Risks specific to agents

- Unintended side effects from actions
- Goal misgeneralization
- Reward hacking
- Deceptive behavior
- Resource acquisition

### 14.8.2 Mitigation strategies

- Action sandboxing and simulation
- Human approval for high-stakes actions
- Capability limitations
- Monitoring and logging
- Shutdown mechanisms

# Chapter 15

## Interpretability and mechanistic understanding

Understanding how large language models process information and arrive at outputs is crucial for trust, safety, and scientific understanding [126]. This chapter explores methods for interpreting LLM behavior at multiple levels of analysis.

### 15.1 Motivation for interpretability

#### 15.1.1 Why interpretability matters

- **Safety:** Detecting harmful behaviors before deployment
- **Alignment verification:** Confirming models pursue intended goals
- **Debugging:** Identifying sources of errors
- **Trust:** Building confidence in model outputs
- **Scientific understanding:** Advancing knowledge of intelligence
- **Regulation:** Meeting transparency requirements

#### 15.1.2 Levels of analysis

Following Marr's levels [127]:

- **Computational:** What problem is being solved?
- **Algorithmic:** What representations and processes are used?
- **Implementation:** How are computations realized in the network?

### 15.2 Probing and representation analysis

#### 15.2.1 Linear probing

Training simple classifiers on intermediate representations:

$$\hat{y} = \sigma(Wh_l + b) \quad (15.1)$$

where  $h_l$  is the hidden state at layer  $l$ , revealing what information is encoded.

## Applications

- Part-of-speech tagging
- Named entity recognition
- Syntactic structure
- Semantic properties
- World knowledge [128]

### 15.2.2 Representation similarity analysis

Comparing representations across:

- Layers within a model
- Different models
- Model representations and human brain activity
- Centered Kernel Alignment (CKA) for comparison

### 15.2.3 Geometry of representations

- Linear subspaces for concepts
- Clustering structure in embedding space
- Dimensionality and intrinsic dimension
- Manifold analysis

## 15.3 Attention analysis

### 15.3.1 Attention visualization

Examining attention patterns to understand information flow:

- Attention head specialization
- Layer-wise attention patterns
- Query-key interactions

### 15.3.2 Attention head functions

Identified attention head types [129, 130]:

- **Positional heads:** Attend to specific relative positions
- **Syntactic heads:** Attend based on grammatical structure
- **Semantic heads:** Attend to semantically related tokens
- **Induction heads:** Copy patterns from context [131]

### 15.3.3 Limitations of attention analysis

- Attention weights don't equal importance [132]
- Information can flow through residual connections
- Multiple attention heads provide redundancy
- Need to consider full computation graph

## 15.4 Mechanistic interpretability

### 15.4.1 Circuits and features

Mechanistic interpretability aims to reverse-engineer neural networks into understandable algorithms [133, 134]:

#### Features

Directions in activation space corresponding to interpretable concepts:

- Individual neurons (polysemantic)
- Linear combinations of neurons
- Superposition of features

#### Circuits

Connected subgraphs implementing specific computations:

- Induction circuits for in-context learning
- Indirect object identification
- Factual recall circuits

### 15.4.2 Sparse autoencoders

Decomposing activations into interpretable features [135, 136]:

$$h = \text{Dec}(\text{ReLU}(\text{Enc}(x))) + x \quad (15.2)$$

where the encoder produces sparse codes revealing monosemantic features.

Key findings:

- Models learn interpretable features in superposition
- Sparse autoencoders extract thousands of meaningful features
- Features correspond to concepts, entities, behaviors
- Can identify safety-relevant features

### 15.4.3 Activation patching

Causal intervention to identify important components [137]:

1. Run model on clean and corrupted inputs
2. Replace activations from one run to another
3. Measure effect on output

Reveals which components are causally responsible for behaviors.

### 15.4.4 Path patching

Extension of activation patching to trace information flow:

- Identifies paths through the network
- Reveals computational structure
- Enables circuit discovery

## 15.5 Knowledge localization

### 15.5.1 Where is knowledge stored?

Research on locating factual knowledge in transformers [137, 138]:

- MLP layers store factual associations
- Attention heads route information
- Early layers process syntax, later layers semantics
- Knowledge distributed but with focal points

### 15.5.2 Knowledge editing

Modifying specific facts without retraining:

#### **ROME (Rank-One Model Editing)**

Direct weight modification for fact updates [137]:

- Identify critical layer for fact
- Compute rank-one update to weights
- Preserves other model behaviors

#### **MEMIT**

Scaling editing to thousands of facts [139]:

- Batch editing across layers
- Maintains model coherence
- Enables large-scale knowledge updates

### 15.5.3 Limitations

- Knowledge often distributed across components
- Editing can have unintended side effects
- Generalization of edits is imperfect
- Scale of modern models challenges localization

## 15.6 Behavioral interpretability

### 15.6.1 Influence functions

Tracing predictions to training examples:

$$\mathcal{I}(z, z_{\text{test}}) = -\nabla_{\theta} L(z_{\text{test}})^{\top} H_{\theta}^{-1} \nabla_{\theta} L(z) \quad (15.3)$$

Identifies training examples most responsible for predictions.

### 15.6.2 Concept bottleneck models

Forcing predictions through interpretable concepts:

- Predict intermediate concept labels
- Final prediction based on concepts
- Enables concept-level intervention

### 15.6.3 Natural language explanations

Using LLMs to explain their own behavior:

- Chain-of-thought as explanation
- Post-hoc rationalization
- Limitations: explanations may not reflect actual computation

## 15.7 Tools and frameworks

### 15.7.1 Interpretability libraries

- **TransformerLens**: Mechanistic interpretability toolkit [140]
- **Baukit**: Activation editing and analysis
- **Captum**: PyTorch interpretability library
- **SHAP**: Shapley value explanations
- **LIT**: Language Interpretability Tool

### 15.7.2 Visualization tools

- Attention visualization (BertViz)
- Neuron activation viewers
- Circuit diagrams
- Embedding projections (UMAP, t-SNE)

## 15.8 Challenges and open problems

### 15.8.1 Scalability

- Methods developed on small models may not transfer
- Computational cost of analysis at scale
- Complexity of billion-parameter models

### 15.8.2 Faithfulness

- Do explanations reflect actual computation?
- Verification of interpretability claims
- Avoiding misleading simplifications

### 15.8.3 Completeness

- Current methods explain fragments of behavior
- Full mechanistic understanding remains distant
- Integration across levels of analysis



# Chapter 16

## Multimodal foundation models

Foundation models increasingly operate across multiple modalities including text, images, audio, video, and structured data [141]. This chapter provides comprehensive coverage of multimodal architectures, training approaches, and applications.

### 16.1 Foundations of multimodal learning

#### 16.1.1 Modality representations

Different modalities require specialized encoders:

- **Text:** Tokenization and transformer encoders
- **Images:** Patch embeddings, CNN features, ViT
- **Audio:** Spectrograms, waveform encoders
- **Video:** Frame sequences, 3D convolutions
- **Point clouds:** PointNet, point transformers

#### 16.1.2 Multimodal fusion strategies

##### Early fusion

Combining modalities at the input level:

- Concatenated embeddings
- Shared tokenization
- Joint encoding from start

##### Late fusion

Combining after modality-specific processing:

- Separate encoders per modality
- Fusion at prediction layer
- Modular and flexible

##### Cross-modal attention

Attention between modality representations:

- Cross-attention layers
- Queries from one modality, keys/values from another
- Enables fine-grained alignment

## 16.2 Vision-language models

### 16.2.1 Contrastive vision-language models

#### CLIP

Contrastive Language-Image Pre-training [142]:

$$\mathcal{L}_{\text{CLIP}} = -\frac{1}{N} \sum_{i=1}^N \log \frac{\exp(\text{sim}(v_i, t_i)/\tau)}{\sum_{j=1}^N \exp(\text{sim}(v_i, t_j)/\tau)} \quad (16.1)$$

Key characteristics:

- Trained on 400M image-text pairs
- Zero-shot image classification via text prompts
- Strong transfer to downstream tasks
- Foundation for many subsequent models

#### ALIGN and BASIC

Scaling contrastive learning:

- ALIGN: 1.8B noisy image-text pairs [143]
- BASIC: Combined contrastive and generative objectives
- Demonstrates benefits of scale over curation

#### SigLIP

Sigmoid loss for language-image pre-training [144]:

- Replaces softmax with sigmoid
- Better batch efficiency
- Improved performance at scale

### 16.2.2 Generative vision-language models

#### GPT-4V

OpenAI's multimodal GPT-4 [30]:

- Native image understanding
- Complex visual reasoning
- Document and chart analysis
- Spatial relationship understanding

#### Gemini

Google's natively multimodal models [20]:

- Trained on interleaved multimodal data
- Strong performance across modalities
- Multiple model sizes (Ultra, Pro, Nano)
- Long context for video understanding

### Claude Vision

Anthropic's vision capabilities [19]:

- Detailed image analysis
- Multi-image reasoning
- Strong at document understanding
- Emphasis on safe visual AI

### LLaVA

Large Language and Vision Assistant [145]:

- Visual instruction tuning
- CLIP encoder + LLaMA decoder
- Open-source multimodal model
- Strong reasoning capabilities

### 16.2.3 Vision encoders

#### Vision Transformer (ViT)

Applying transformers to images [146]:

- Image divided into patches (typically  $16 \times 16$ )
- Patches linearly embedded as tokens
- Standard transformer encoder
- Scales well with data and compute

#### Variants and improvements

- **DeiT**: Data-efficient training with distillation
- **Swin Transformer**: Hierarchical with shifted windows [147]
- **EVA**: Exploring the limits of masked image modeling [148]
- **DINOv2**: Self-supervised visual features [149]

### 16.2.4 Document understanding

#### Document AI models

- **LayoutLM** [150]: Joint text and layout modeling
- **Donut**: Document understanding transformer
- **Pix2Struct**: Screenshot parsing as pre-training
- **DocPedia**: Large-scale document understanding

#### Capabilities

- OCR and text extraction
- Table structure recognition
- Form understanding
- Document classification
- Information extraction

## 16.3 Audio and speech models

### 16.3.1 Speech recognition

#### Whisper

Robust speech recognition [151]:

- Trained on 680K hours of multilingual audio
- Supports 97 languages
- Transcription and translation
- Robust to noise and accents

#### wav2vec 2.0

Self-supervised speech representations [152]:

- Contrastive learning on raw audio
- Fine-tuning for ASR with limited labeled data
- Foundation for many speech models

### 16.3.2 Speech synthesis

#### Text-to-speech models

- **VALL-E** [153]: Neural codec language model for TTS
- **Tortoise TTS**: High-quality multi-voice synthesis
- **XTTS**: Cross-lingual voice cloning
- **Bark**: Generative audio model by Suno

#### Voice cloning

- Few-shot voice adaptation
- Speaker embedding extraction
- Ethical considerations and detection

### 16.3.3 Audio understanding

- Music understanding and generation
- Sound event detection
- Audio captioning
- Speech emotion recognition

## 16.4 Video understanding

### 16.4.1 Video foundation models

#### VideoMAE

Masked autoencoding for video [154]:

- High masking ratio (90%)
- Learns temporal structure
- Efficient pre-training

### Video-LLaMA

Video understanding with LLMs [155]:

- Visual and audio encoders
- Frame aggregation strategies
- Temporal reasoning

#### 16.4.2 Long video understanding

Challenges and approaches:

- Computational cost of processing many frames
- Memory-efficient architectures
- Keyframe selection
- Hierarchical temporal modeling
- Gemini 1.5 Pro: 1M token context for video [156]

#### 16.4.3 Video generation

**State of the art**

- **Sora**: OpenAI's video generation model
- **Runway Gen-2**: Text and image to video
- **Pika**: AI video generation platform
- **Stable Video Diffusion**: Open video synthesis

**Challenges**

- Temporal consistency
- Physics understanding
- Long-form generation
- Controllability

## 16.5 Unified multimodal models

### 16.5.1 Any-to-any models

Models handling arbitrary input/output modality combinations:

#### Unified-IO

Universal input-output model [157]:

- Single model for vision, language, and more
- Unified tokenization across modalities
- Supports diverse tasks

### Meta-Transformer

Unified encoding across modalities [158]:

- Shared transformer backbone
- Modality-specific tokenizers
- 12 modalities supported

## 16.5.2 Multimodal generation

### Image generation

- **DALL-E 3**: Text-to-image with GPT integration
- **Midjourney**: Artistic image generation
- **Stable Diffusion** [159]: Open-source diffusion model
- **Imagen**: Google's text-to-image model

### Music and audio generation

- **MusicLM**: Text-to-music generation
- **AudioLDM**: Text-to-audio synthesis
- **Suno**: AI music generation

## 16.6 Training multimodal models

### 16.6.1 Data considerations

- **Paired data**: Image-caption, video-transcript pairs
- **Interleaved data**: Documents with mixed modalities
- **Synthetic data**: Generated captions, descriptions
- **Web-scale collection**: LAION, DataComp [160]

### 16.6.2 Training objectives

- Contrastive learning (CLIP-style)
- Next token prediction on multimodal sequences
- Masked prediction across modalities
- Diffusion objectives for generation

### 16.6.3 Alignment techniques

Connecting pre-trained unimodal models:

- Linear projection layers
- Q-Former (BLIP-2) [161]
- Cross-attention adapters
- Gradual unfreezing

## 16.7 Evaluation

### 16.7.1 Vision-language benchmarks

- **VQA** [162]: Visual question answering
- **GQA**: Compositional visual reasoning
- **COCO Captioning**: Image captioning
- **TextVQA**: Reading text in images
- **DocVQA**: Document understanding
- **MMMU**: Multimodal multi-discipline understanding

### 16.7.2 Video benchmarks

- **ActivityNet**: Action recognition and localization
- **MSRVTT**: Video captioning and retrieval
- **NextQA**: Temporal reasoning in video
- **EgoSchema**: Long-form video understanding



# Chapter 17

## Knowledge representation and reasoning

This chapter examines how large language models represent, store, retrieve, and reason with knowledge, including methods for knowledge editing and grounding.

### 17.1 Knowledge in language models

#### 17.1.1 Parametric vs. non-parametric knowledge

##### Parametric knowledge

Knowledge encoded in model weights during training:

- Distributed across parameters
- Accessed through forward pass
- Fixed after training
- Subject to knowledge cutoff

##### Non-parametric knowledge

External knowledge accessed at inference:

- Retrieval-augmented approaches
- Knowledge bases and databases
- Real-time information
- Updateable without retraining

#### 17.1.2 Knowledge storage mechanisms

Research on where and how transformers store factual knowledge:

##### MLP layers as key-value memories

Evidence that MLP layers function as associative memories [163]:

- First MLP layer as keys
- Second MLP layer as values
- Factual associations stored in patterns

**Distributed representations**

- Knowledge spread across multiple components
- Redundancy provides robustness
- No single “knowledge location”

## 17.2 Factual recall and knowledge probing

### 17.2.1 Knowledge probing

Assessing what knowledge models contain:

**LAMA probe**

Language Model Analysis [164]:

- Cloze-style fact completion
- Tests relational knowledge
- “Paris is the capital of [MASK]”

**Probing datasets**

- **T-REx**: Wikipedia-derived facts
- **ConceptNet**: Commonsense knowledge
- **SQuAD**: Reading comprehension

### 17.2.2 Knowledge boundaries

Understanding what models know and don’t know:

- Calibration of confidence
- “I don’t know” responses
- Knowledge gaps and inconsistencies
- Temporal knowledge limitations

## 17.3 Knowledge editing

### 17.3.1 Motivation

Updating specific facts without full retraining:

- Correcting errors
- Updating outdated information
- Removing harmful knowledge
- Personalizing models

### 17.3.2 Editing methods

**ROME and MEMIT**

Rank-one and mass editing [137, 139]:

- Locate knowledge in specific layers
- Compute targeted weight updates
- Preserve unrelated knowledge

## MEND

Model Editor Networks with Gradient Decomposition [165]:

- Learn to transform gradients for editing
- Fast inference-time editing
- Scalable to many edits

## In-context editing

- Providing corrections in context
- No weight modification
- Limited by context window

### 17.3.3 Evaluation of edits

- **Efficacy:** Does the edit succeed?
- **Generalization:** Does it apply to rephrasings?
- **Locality:** Are unrelated facts preserved?
- **Consistency:** Are related facts updated?

### 17.3.4 Limitations and challenges

- Edits may not generalize properly
- Cascading effects on related knowledge
- Scalability to many edits
- Detecting when edits are needed

## 17.4 Knowledge graphs and structured knowledge

### 17.4.1 Integration with LLMs

#### Knowledge graph augmentation

- Retrieve relevant subgraphs for context
- Entity linking to knowledge bases
- Graph-to-text verbalization

#### LLMs for knowledge graphs

- Entity and relation extraction
- Knowledge graph completion
- Question answering over KGs

### 17.4.2 Structured reasoning

- Following relationship chains
- Constraint satisfaction
- Logical inference over facts

## 17.5 Grounding and world models

### 17.5.1 Symbol grounding

Connecting language to real-world referents:

- Multimodal grounding through images/video
- Embodied grounding through interaction
- Simulation-based grounding

### 17.5.2 World models

Do LLMs build internal world models?

#### Evidence for world models

- Spatial reasoning capabilities [166]
- Board game state tracking
- Physical intuition

#### Limitations

- Inconsistencies in physical reasoning
- Difficulty with novel scenarios
- Lack of causal understanding

## 17.6 Reasoning with knowledge

### 17.6.1 Multi-hop reasoning

Combining multiple facts:

- Chain-of-thought for explicit reasoning
- Implicit multi-hop in parameters
- Retrieval for knowledge-intensive reasoning

### 17.6.2 Commonsense reasoning

- Physical commonsense (objects, forces)
- Social commonsense (intentions, emotions)
- Temporal commonsense (sequences, durations)
- Evaluated on benchmarks like PIQA, SIQA, HellaSwag

### 17.6.3 Logical reasoning

- Deductive reasoning
- Inductive reasoning
- Abductive reasoning
- Integration with symbolic systems

# Chapter 18

## Multilingual and cross-lingual NLP

Large language models have transformed multilingual natural language processing, enabling cross-lingual transfer and supporting hundreds of languages [167]. This chapter examines multilingual model architectures, training approaches, and challenges.

### 18.1 Multilingual language models

#### 18.1.1 Evolution of multilingual models

##### mBERT

Multilingual BERT trained on 104 languages:

- Shared vocabulary and parameters
- Surprising cross-lingual transfer
- No explicit cross-lingual objective

##### XLM and XLM-R

Explicit cross-lingual pre-training [168, 169]:

- Translation language modeling (TLM)
- XLM-R: 100 languages, improved performance
- CommonCrawl data at scale

##### mT5

Multilingual T5 [170]:

- 101 languages
- Encoder-decoder architecture
- Text-to-text framework

#### 18.1.2 Modern multilingual LLMs

##### BLOOM

Open multilingual LLM [171]:

- 176B parameters
- 46 natural languages + 13 programming languages
- Emphasis on underrepresented languages

### **PaLM and Gemini**

Google's multilingual capabilities:

- Strong performance across languages
- Translation capabilities
- Cross-lingual reasoning

### **Llama multilingual variants**

- Llama 2/3 with some multilingual capability
- Community fine-tunes for specific languages
- Chinese Llama, Japanese Llama variants

## **18.2 Cross-lingual transfer**

### **18.2.1 Zero-shot cross-lingual transfer**

Training on one language, evaluating on another:

- Shared multilingual representations enable transfer
- English as typical source language
- Performance gap with target language supervision

### **18.2.2 Mechanisms of cross-lingual transfer**

#### **Shared vocabulary**

- Subword overlap between languages
- Anchor points for alignment
- Script sharing benefits

#### **Representation alignment**

- Similar concepts map to similar representations
- Language-neutral semantic space
- Syntactic structure transfer

### **18.2.3 Improving cross-lingual transfer**

- Parallel data for alignment
- Translation as auxiliary task
- Cross-lingual contrastive learning
- Language adapters

## **18.3 Machine translation**

### **18.3.1 Neural machine translation with LLMs**

#### **In-context translation**

- Zero-shot: “Translate to French: ...”
- Few-shot with examples
- Competitive with specialized systems

### Fine-tuned translation models

- NLLB (No Language Left Behind) [172]: 200 languages
- M2M-100: Multilingual without English pivot
- SeamlessM4T: Speech and text translation

### 18.3.2 Translation quality

- High-resource pairs: Near human quality
- Low-resource pairs: Significant gap remains
- Specialized domains: Need adaptation

## 18.4 Low-resource languages

### 18.4.1 Challenges

- Limited training data
- Underrepresentation in pre-training
- Evaluation data scarcity
- Script and tokenization issues

### 18.4.2 Approaches

#### Data augmentation

- Back-translation
- Cross-lingual data transfer
- Synthetic data generation

#### Transfer from related languages

- Language family transfer
- Script conversion
- Phonetic representations

#### Specialized models

- AfroLM for African languages
- IndicBERT for Indian languages
- AraGPT for Arabic

## 18.5 Multilingual evaluation

### 18.5.1 Benchmarks

- **XTREME** [173]: Cross-lingual evaluation across 40 languages
- **XGLUE**: Cross-lingual generalization benchmark
- **FLORES**: Machine translation benchmark for 200 languages
- **Multilingual MMLU**: Knowledge across languages
- **MGSM**: Multilingual grade school math

### 18.5.2 Evaluation challenges

- Translation quality of benchmarks
- Cultural appropriateness
- Annotator availability
- Dialect and variety coverage

## 18.6 Linguistic diversity considerations

### 18.6.1 Morphologically rich languages

- Agglutinative languages (Turkish, Finnish)
- Fusional languages (Russian, Arabic)
- Tokenization challenges
- Vocabulary efficiency

### 18.6.2 Script diversity

- Latin, Cyrillic, Arabic, Chinese characters
- Right-to-left scripts
- Mixed script handling
- Script conversion trade-offs

### 18.6.3 Code-switching

Mixing languages within text:

- Common in multilingual communities
- Challenges for models
- Dedicated code-switching datasets

# Chapter 19

## Model compression and efficiency

As language models grow in size, techniques for reducing computational and memory requirements become essential for practical deployment [174]. This chapter covers compression methods and efficient architectures.

### 19.1 Quantization

#### 19.1.1 Fundamentals

Reducing numerical precision of weights and activations:

- **FP32**: Full precision (32 bits)
- **FP16/BF16**: Half precision (16 bits)
- **INT8**: 8-bit integers
- **INT4**: 4-bit integers
- **Binary/Ternary**: Extreme quantization

#### 19.1.2 Post-training quantization (PTQ)

Quantizing pre-trained models without retraining:

#### GPTQ

Accurate post-training quantization [175]:

- Layer-wise quantization with error compensation
- 4-bit quantization with minimal quality loss
- One-shot quantization process

#### AWQ

Activation-aware Weight Quantization [176]:

- Identifies salient weights via activation magnitude
- Protects important weights from quantization
- Strong 4-bit performance

## GGML/GGUF

Quantization format for local inference:

- Multiple quantization levels (Q4, Q5, Q8)
- CPU-optimized inference
- Used in llama.cpp

### 19.1.3 Quantization-aware training (QAT)

Training with quantization in the loop:

- Simulates quantization during forward pass
- Straight-through estimator for gradients
- Better quality than PTQ at low precision
- More expensive than PTQ

### 19.1.4 Mixed-precision strategies

- Different precision for different layers
- Higher precision for sensitive components
- Attention in higher precision than MLP

## 19.2 Pruning

### 19.2.1 Pruning strategies

Removing unnecessary weights or components:

#### Unstructured pruning

- Remove individual weights
- High sparsity achievable
- Requires sparse matrix support
- Limited hardware acceleration

#### Structured pruning

- Remove entire neurons, heads, or layers
- Hardware-friendly dense operations
- Lower maximum sparsity
- SparseGPT [177]: Unstructured pruning at scale

### 19.2.2 Pruning criteria

- Magnitude-based: Remove small weights
- Gradient-based: Remove low-gradient weights
- Sensitivity-based: Remove least important components
- Lottery ticket hypothesis: Find winning subnetworks

### 19.2.3 Layer and attention head pruning

- Removing entire transformer layers
- Attention head pruning for efficiency
- Dynamic head selection at inference

## 19.3 Knowledge distillation

### 19.3.1 Distillation approaches

Transferring knowledge from large teacher to small student:

#### Logit distillation

Matching output distributions:

$$\mathcal{L}_{\text{KD}} = \text{KL}(\text{softmax}(z_t/T) || \text{softmax}(z_s/T)) \quad (19.1)$$

where  $T$  is temperature and  $z$  are logits.

#### Feature distillation

- Match intermediate representations
- Layer-wise matching
- Attention transfer

### 19.3.2 LLM-specific distillation

#### Sequence-level distillation

- Generate training data with teacher
- Train student on teacher outputs
- Used in Alpaca, Vicuna

#### Notable distilled models

- DistilBERT: 60% faster, 97% performance
- TinyLlama: 1.1B parameter efficient model
- Phi series: Small but capable models

## 19.4 Efficient architectures

### 19.4.1 Linear attention

Reducing quadratic attention complexity:

$$\text{LinearAttn}(Q, K, V) = \phi(Q)(\phi(K)^T V) \quad (19.2)$$

where  $\phi$  is a feature map enabling  $O(n)$  complexity.

### 19.4.2 State space models

Alternative to attention with linear complexity:

**Mamba**

Selective state space model [178]:

- Input-dependent state transitions
- Linear time complexity
- Competitive with transformers
- Hardware-efficient implementation

**RWKV**

Receptance Weighted Key Value [179]:

- RNN-like linear complexity
- Transformer-like parallelizable training
- Growing community adoption

**19.4.3 Mixture of Experts**

Conditional computation for efficiency:

- Sparse activation of expert subnetworks
- Router selects relevant experts
- Linear compute scaling with parameters
- Mixtral, Switch Transformer

**19.5 Inference optimization****19.5.1 KV cache optimization****Multi-Query Attention (MQA)**

Shared keys and values across heads [180]:

- Dramatically reduces KV cache size
- Minor quality impact
- Used in PaLM, Falcon

**Grouped-Query Attention (GQA)**

Intermediate between MHA and MQA [27]:

- Groups of heads share KV
- Balance of quality and efficiency
- Used in Llama 2

**19.5.2 Speculative decoding**

Using small model to draft tokens:

- Small model generates candidates
- Large model verifies in parallel
- 2-3x speedup possible
- No quality degradation

### 19.5.3 Continuous batching

- Dynamic request scheduling
- Improved GPU utilization
- Reduced latency
- vLLM, TensorRT-LLM

## 19.6 Hardware considerations

### 19.6.1 GPU optimization

- Tensor core utilization
- Memory bandwidth optimization
- Kernel fusion
- FlashAttention [25, 28]

### 19.6.2 Alternative hardware

- TPUs for training and inference
- Apple Silicon Neural Engine
- Specialized AI accelerators
- CPU inference with quantization



# Chapter 20

## Legal and regulatory frameworks

The rapid advancement of large language models has prompted governments and international bodies to develop regulatory frameworks addressing AI development and deployment [181]. This chapter provides comprehensive coverage of the emerging legal landscape.

### 20.1 Overview of AI regulation

#### 20.1.1 Regulatory approaches

Different jurisdictions take varying approaches:

- **Risk-based:** Regulations proportional to risk level
- **Sector-specific:** Different rules for different domains
- **Principles-based:** Broad guidelines rather than specific rules
- **Technology-specific:** Rules targeting specific technologies

#### 20.1.2 Key regulatory objectives

- Safety and security
- Transparency and explainability
- Fairness and non-discrimination
- Privacy protection
- Accountability
- Human oversight

### 20.2 European Union AI Act

#### 20.2.1 Risk categories

The EU AI Act [182] establishes a risk-based framework:

##### Unacceptable risk (prohibited)

- Social scoring by governments
- Real-time biometric identification (with exceptions)
- Manipulation through subliminal techniques
- Exploitation of vulnerabilities

### High risk

Systems in critical areas requiring compliance:

- Biometric identification
- Critical infrastructure
- Educational and vocational access
- Employment decisions
- Essential services access
- Law enforcement
- Migration and asylum
- Justice administration

### Limited risk

Transparency obligations:

- Chatbots must disclose AI nature
- Deepfakes must be labeled
- Emotion recognition disclosure

### Minimal risk

No specific requirements but voluntary codes encouraged.

## 20.2.2 General-purpose AI provisions

Specific rules for foundation models:

- Technical documentation requirements
- Training data transparency
- Copyright compliance
- Systemic risk assessment for powerful models
- Energy consumption reporting

## 20.2.3 Compliance requirements

- Conformity assessments
- Quality management systems
- Post-market monitoring
- Serious incident reporting
- Penalties up to 7% global revenue

## 20.3 United States regulatory landscape

### 20.3.1 Executive Order on AI (2023)

Key provisions [183]:

- Safety testing and reporting for powerful models
- Dual-use foundation model reporting thresholds
- Red-teaming requirements
- Standards development through NIST
- Watermarking and content authentication

### 20.3.2 Agency-specific regulation

#### FTC

- Enforcement against deceptive AI practices
- Unfair and discriminatory algorithms
- Consumer protection focus

#### SEC

- AI in financial services
- Algorithmic trading oversight
- Disclosure requirements

#### FDA

- AI in medical devices
- Clinical decision support
- Software as medical device (SaMD)

### 20.3.3 State-level regulation

- California AI transparency bills
- Colorado AI Act
- Illinois Biometric Privacy Act
- Varying approaches across states

## 20.4 China AI governance

### 20.4.1 Regulatory framework

China's approach includes [184]:

#### Generative AI regulations (2023)

- Content safety requirements
- Training data compliance
- User registration requirements
- Security assessments

**Algorithm recommendation rules**

- Transparency in algorithmic recommendations
- User control over personalization
- Labeling requirements

**Deep synthesis regulations**

- Deepfake labeling requirements
- Consent for likeness use
- Traceability requirements

## 20.5 International frameworks

### 20.5.1 OECD AI Principles

Foundational international guidelines:

- Inclusive growth and sustainable development
- Human-centered values and fairness
- Transparency and explainability
- Robustness, security, and safety
- Accountability

### 20.5.2 G7 Hiroshima AI Process

International coordination on AI governance:

- Guiding principles for AI developers
- Code of conduct for advanced AI systems
- Risk-based approach emphasis

### 20.5.3 UNESCO AI Ethics

Global ethical framework:

- Human rights protection
- Environmental sustainability
- Cultural diversity
- Multi-stakeholder governance

## 20.6 Intellectual property issues

### 20.6.1 Training data copyright

**Legal debates**

- Fair use / fair dealing arguments
- Transformative use considerations
- Opt-out mechanisms
- Licensing requirements

### Litigation landscape

- Getty Images v. Stability AI
- New York Times v. OpenAI
- Sarah Silverman et al. v. OpenAI
- Class actions pending

### 20.6.2 Generated content ownership

- Authorship requirements for copyright
- US Copyright Office guidance
- Human authorship threshold
- Joint human-AI works

### 20.6.3 Patent considerations

- AI as inventor debates
- AI-assisted invention disclosure
- Patent eligibility for AI methods

## 20.7 Liability frameworks

### 20.7.1 Product liability

- AI systems as products
- Defect definitions for AI
- Manufacturer vs. deployer liability
- EU AI Liability Directive proposal

### 20.7.2 Professional liability

- AI in professional services
- Standard of care considerations
- Malpractice implications
- Insurance requirements

## 20.8 Privacy regulations

### 20.8.1 GDPR implications

- Personal data in training sets
- Right to erasure challenges
- Automated decision-making (Article 22)
- Data protection impact assessments
- Cross-border transfer restrictions

### 20.8.2 US privacy laws

- CCPA/CPRA in California
- State privacy law patchwork
- Proposed federal privacy legislation

## 20.9 Industry self-governance

### 20.9.1 Voluntary commitments

- White House voluntary commitments
- Frontier Model Forum
- Partnership on AI
- Model cards and documentation

### 20.9.2 Standards development

- NIST AI Risk Management Framework
- ISO/IEC AI standards
- IEEE ethical AI standards
- Industry-specific standards

## 20.10 Compliance strategies

### 20.10.1 For AI developers

- Documentation and record-keeping
- Safety testing protocols
- Red-teaming programs
- Responsible disclosure
- Third-party audits

### 20.10.2 For deployers

- Use case risk assessment
- Human oversight mechanisms
- Monitoring and evaluation
- User disclosure requirements
- Incident response plans

# Chapter 21

## Advanced topics and future directions

### 21.1 Efficient architectures and training

#### 21.1.1 Parameter-Efficient Models

##### Distillation

Transferring knowledge from large teacher models to smaller student models:

$$\mathcal{L}_{\text{distill}} = \alpha \mathcal{L}_{\text{CE}}(y, \hat{y}_s) + (1 - \alpha) \mathcal{L}_{\text{KL}}(\hat{y}_t, \hat{y}_s) \quad (21.1)$$

where  $\hat{y}_t$  is teacher output,  $\hat{y}_s$  is student output,  $y$  is ground truth.

##### Benefits:

- Reduced model size with minimal performance loss
- Faster inference
- Lower deployment costs
- Examples: DistilBERT, TinyBERT

##### Pruning

Removing unnecessary weights:

- Magnitude-based pruning
- Structured vs. unstructured
- Iterative pruning during training
- Can achieve significant compression with modest accuracy loss

##### Quantization

Reducing numerical precision:

- Post-training quantization (8-bit, 4-bit, even lower)
- Quantization-aware training
- Mixed precision strategies
- Enables deployment on resource-constrained devices

#### 21.1.2 Retrieval-Augmented Generation

Combining parametric knowledge with retrieved information:

**Architecture**

1. Query formulation from input
2. Retrieve relevant documents from corpus
3. Condition generation on retrieved context

**Examples:**

- REALM: Retrieve then predict
- RAG: Retrieval-Augmented Generation
- Atlas: Few-shot learning with retrieval

**Advantages:**

- Reduced hallucination through grounding
- Access to up-to-date information
- Attribution and citations
- Smaller parametric model sufficient
- Easier to update knowledge (update retrieval corpus)

**Challenges:**

- Retrieval quality critical
- Computational overhead
- Integration of retrieved and parametric knowledge

**21.1.3 Long Context and Efficient Attention****Approaches for Longer Context****Sparse attention patterns:**

- Longformer: Local + global attention
- BigBird: Random + window + global
- Reduced  $O(n^2)$  to  $O(n \log n)$  or  $O(n)$

**Recurrence and state tracking:**

- Transformer-XL: Segment-level recurrence
- Compressive Transformers: Compressed memory
- Maintaining state across segments

**Hierarchical processing:**

- Processing at multiple granularities
- Chunking and summarization

**Flash Attention and IO optimization:**

- Algorithm-level optimization for GPU efficiency
- Enables longer contexts with standard attention

### Context Length Progress

Evolution of context windows:

- GPT-3: 2k tokens
- GPT-4: 8k standard, 32k extended, 128k turbo
- Claude 2: 100k tokens
- Claude 3: 200k tokens
- Gemini 1.5: 1M tokens (experimental)

[Figure 21.1](#) visualizes the dramatic expansion of context window sizes across successive model generations.

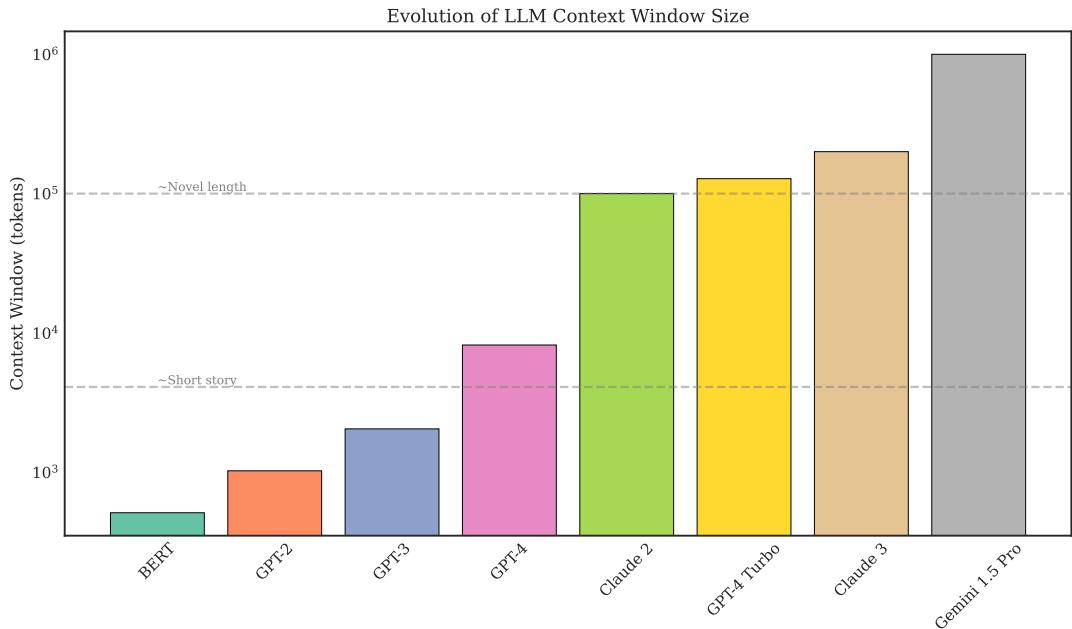


Figure 21.1: Evolution of context window sizes in large language models from 2018 to 2024. The plot shows exponential growth in supported context lengths, from 512 tokens in BERT [15] to over 1 million tokens in Gemini 1.5 Pro [156]. Data sourced from official model documentation: GPT-2 [16], GPT-3 [1], GPT-4 [30], and Claude 3 [19].

## 21.2 Multimodal Foundation Models

### 21.2.1 Vision-Language Models

#### Architecture Approaches

**Two-tower models:**

- Separate encoders for vision and language
- Aligned representation spaces
- CLIP: Contrastive Language-Image Pre-training

#### Cross-modal fusion:

- Early fusion: Combine modalities early in processing
- Late fusion: Process separately, combine for output

- Cross-attention between modalities

#### **Unified models:**

- Single model processing all modalities
- Flamingo: Visual language model with frozen language model
- GPT-4V, Claude 3: Native multimodal processing

#### **Training Objectives**

- Contrastive learning (image-text matching)
- Masked image modeling
- Image captioning
- Visual question answering
- Multimodal masked language modeling

### **21.2.2 Audio, Video, and Beyond**

#### **Audio-Language Models**

- Speech recognition and synthesis
- Audio captioning and understanding
- Music generation and analysis

#### **Video Understanding**

- Temporal modeling challenges
- Action recognition
- Video captioning and question answering
- Efficient processing of long videos

#### **Unified Multimodal Models**

Models processing arbitrary combinations of modalities:

- ImageBind: Binding six modalities in joint space
- Unified-IO: Single model for diverse modalities and tasks
- Future direction toward truly general-purpose models

## **21.3 Agents and Tool Use**

### **21.3.1 LLM Agents**

Agents that interact with environments to accomplish goals:

#### **Components**

- **Planning:** Decomposing tasks into steps
- **Memory:** Maintaining context and history
- **Tools:** Accessing external capabilities
- **Execution:** Taking actions in environment
- **Reflection:** Learning from outcomes

### Frameworks

- ReAct: Reasoning and Acting
- AutoGPT: Autonomous task completion
- LangChain: Composable agent components
- ChatGPT plugins and function calling

### 21.3.2 Tool Use and Function Calling

Extending LLM capabilities through tool access:

#### Tool Categories

- Information retrieval (search, databases)
- Computation (calculators, code execution)
- External services (APIs, web services)
- Data manipulation (file operations, transformations)
- Domain-specific tools (scientific instruments, simulation)

#### Challenges

- Tool selection and composition
- Error handling and recovery
- Security and sandboxing
- Reliability and verification

### 21.3.3 Multi-Agent Systems

Multiple agents collaborating:

- Specialized agents for different capabilities
- Debate and consensus mechanisms
- Coordination and communication protocols
- Emergent behaviors and collective intelligence

## 21.4 Embodied AI and Robotics

### 21.4.1 Foundation Models for Robotics

Applying LLM capabilities to robotics:

#### Applications

- High-level task planning from natural language
- Low-level control policy learning
- Human-robot interaction
- Sim-to-real transfer

### Challenges

- Grounding language in physical world
- Real-time constraints
- Safety in physical environments
- Generalization across environments and embodiments

### Approaches

- Vision-language-action models
- Code generation for robot control
- Reward specification from language
- Multimodal pre-training including robotic data

## 21.5 Reasoning and Planning

### 21.5.1 Advanced Reasoning Techniques

#### Tree-of-Thoughts

Exploring multiple reasoning paths:

- Branching and backtracking
- Evaluating intermediate states
- Search over reasoning space
- Improved performance on complex reasoning

#### Self-Consistency

- Sample multiple reasoning paths
- Select most consistent answer
- Improves robustness and accuracy

#### Decomposition Strategies

- Least-to-most prompting
- Recursive problem decomposition
- Subgoal generation

### 21.5.2 Integration with Symbolic Systems

Combining neural and symbolic approaches:

- **Neural-symbolic integration:** Using LLMs to interface with formal systems
- **Program synthesis:** Generating executable programs
- **Formal verification:** Ensuring correctness through formal methods
- **Knowledge graphs:** Structured knowledge integration

### 21.5.3 Mathematical and Scientific Reasoning

#### Theorem Proving

- Autoformalization: Translating natural language to formal mathematics
- Proof search and synthesis
- Integration with proof assistants (Lean, Coq, Isabelle)
- AlphaProof and similar systems

#### Scientific Discovery

- Hypothesis generation
- Experimental design
- Data analysis and interpretation
- Literature synthesis
- Challenges in novelty and verification

## 21.6 Interpretability and Mechanistic Understanding

### 21.6.1 Motivation

Understanding how LLMs work internally:

- Safety: Detecting deception, misalignment
- Capability assessment: Understanding what models can/cannot do
- Debugging: Fixing failures and biases
- Scientific understanding: Reverse-engineering intelligence

### 21.6.2 Approaches

#### Probing and Attribution

- Probing classifiers: What information is encoded?
- Attention analysis: What does the model attend to?
- Input attribution: Which inputs matter for outputs?
- Causal tracing: Causal paths through network

#### Mechanistic Interpretability

- Reverse-engineering circuits implementing behaviors
- Understanding individual neurons and layers
- Compositional understanding of computations
- Induction heads, indirect object identification, etc.

#### Representation Analysis

- Geometric structure of representation spaces
- Linear representations of concepts
- Compositional structure

### 21.6.3 Challenges

- **Scale:** Billions of parameters, complex interactions
- **Polysemy:** Individual neurons responding to multiple concepts
- **Distributed representations:** Information spread across many neurons
- **Superposition:** Representing more features than dimensions

## 21.7 Emerging Architectures and Paradigms

### 21.7.1 State Space Models

Alternative to Transformers using state space representations:

#### Structured State Space Models (S4)

- Continuous-time representations
- Efficient for long sequences
- Linear time complexity
- Strong performance on long-range tasks

#### Mamba

Recent state space architecture:

- Selective state spaces (data-dependent transitions)
- Linear scaling in sequence length
- Competitive with Transformers on language modeling
- More efficient inference

### 21.7.2 Hybrid Architectures

Combining different architectural elements:

- Transformer + RNN components
- Transformer + state space models
- Mixture of different attention patterns
- Adaptive computation (different amounts of computation per token)

### 21.7.3 Novel Training Paradigms

#### Continual Learning

Training models continuously on new data:

- Avoiding catastrophic forgetting
- Updating knowledge over time
- Balancing stability and plasticity

### Meta-Learning

Learning to learn:

- Few-shot learning through meta-training
- Task adaptation with minimal data
- Learning inductive biases

### Self-Supervised Learning Advances

New pre-training objectives beyond language modeling:

- Contrastive learning
- Masked prediction variants
- Multi-task self-supervision

## 21.8 Future Directions and Open Challenges

### 21.8.1 Scaling Frontiers

#### Continued Scaling

- Models with trillions of parameters
- Training on tens of trillions of tokens
- Implications of scaling laws
- Computational and environmental costs

#### Efficient Scaling

- Better algorithms reducing compute needs
- Improved data quality and curation
- Mixture-of-experts and sparse models
- Hardware co-design

### 21.8.2 Capabilities

#### Deeper Reasoning

- Extended multi-step reasoning
- Formal logical reasoning
- Causal reasoning
- Abstract conceptual reasoning

#### True Multimodality

- Seamless integration of all modalities
- Unified representations
- Cross-modal reasoning and generation

### Long-Term Memory and Coherence

- Maintaining consistency over extended interactions
- Episodic memory systems
- Personalization and user modeling

### Grounding and Factuality

- Reducing hallucination
- Better world models
- Integration with knowledge bases
- Uncertainty quantification

## 21.8.3 Safety and Alignment

### Scalable Oversight

- Aligning superhuman systems
- Recursive reward modeling
- Debate and amplification

### Robustness

- Adversarial robustness
- Out-of-distribution generalization
- Avoiding specification gaming

### Interpretability

- Understanding increasingly capable models
- Detecting deception
- Verifying alignment

## 21.8.4 Democratization and Access

- Open-source models approaching frontier performance
- Efficient models for edge deployment
- Reducing computational barriers
- Multilingual and multicultural representation

## 21.8.5 Scientific Understanding

### Theory of Deep Learning

- Understanding generalization
- Theoretical foundations for scaling laws
- Emergence and phase transitions

### Understanding Intelligence

- What do LLMs reveal about intelligence?
- Connections to cognitive science and neuroscience
- Necessary vs. sufficient components

#### 21.8.6 Societal Integration

- Governance frameworks
- Economic adaptation
- Educational transformation
- Ethical consensus building
- Equitable distribution of benefits

### 21.9 Conclusion

The field of large language models and foundation models represents one of the most rapidly advancing areas of artificial intelligence. From the foundational Transformer architecture to models with hundreds of billions of parameters exhibiting remarkable capabilities, the progress over the past few years has been extraordinary.

Key themes emerge:

- **Scaling:** Larger models trained on more data consistently improve, though efficiency advances are equally important
- **Generality:** Foundation models are increasingly general-purpose, handling diverse tasks and modalities
- **Emergence:** New capabilities appear at scale, often unexpectedly
- **Alignment:** Technical capability must be paired with alignment to human values
- **Impact:** These systems are transforming work, creativity, and information access

Significant challenges remain:

- Hallucination and factual accuracy
- Bias and fairness
- Interpretability and understanding
- Safety and robustness
- Equitable access and distribution of benefits
- Governance and regulation

The future promises continued rapid development. Models will become more capable, more efficient, and more integrated into society. Critical questions—both technical and societal—will require ongoing research, dialogue, and thoughtful decision-making.

This encyclopedia has aimed to provide a comprehensive snapshot of the field as of early 2026. While specific models and techniques will continue to evolve, the fundamental principles, challenges, and frameworks explored here provide a foundation for understanding and engaging with these transformative technologies.

The development of large language models and foundation models represents not just a technical achievement, but a profound development with implications for intelligence, creativity, knowledge, work, and society. Realizing their potential while mitigating risks will require continued collaboration across disciplines, stakeholders, and communities.



# Glossary

## **Activation function**

A non-linear function applied element-wise to neural network outputs, enabling the learning of complex patterns. Common examples include ReLU, GELU, and SiLU.

## **Alignment**

The process of ensuring AI systems behave according to human values and intentions, including safety, helpfulness, and harmlessness.

## **Attention mechanism**

A neural network component that allows models to focus on different parts of the input when producing outputs, enabling variable-length context processing.

## **Autoregressive model**

A model that generates outputs sequentially, where each output depends on previously generated outputs. GPT-style language models are autoregressive.

**Batch size** The number of training examples processed together before updating model weights.

## **BFloat16 (BF16)**

A 16-bit floating-point format with the same exponent range as float32 but reduced precision, widely used for training large models.

## **Causal language modeling**

Pre-training objective where the model predicts the next token given all previous tokens, also known as autoregressive language modeling.

## **Chain-of-thought (CoT)**

A prompting technique that encourages models to generate intermediate reasoning steps before producing final answers.

## **Checkpoint**

A saved snapshot of model parameters and training state, enabling resumption of training or model deployment.

## **Constitutional AI (CAI)**

An alignment approach that uses a set of principles to guide model behavior through self-critique and revision.

## **Context window**

The maximum number of tokens a model can process in a single forward pass, determining the effective memory of the model.

## **Cross-entropy loss**

The standard loss function for language modeling, measuring the difference between predicted and actual token distributions.

**Decoder** The component of a transformer that generates output sequences, using masked self-attention to prevent attending to future tokens.

## **Direct Preference Optimization (DPO)**

An alignment technique that directly optimizes models on preference data without training a separate reward model.

**Distillation**

The process of training a smaller student model to mimic a larger teacher model, transferring knowledge while reducing computational requirements.

**Embedding**

A dense vector representation of discrete tokens or concepts in a continuous vector space.

**Emergent capability**

An ability that appears in larger models but is absent in smaller ones, often arising unexpectedly at specific scale thresholds.

**Encoder** A transformer component that processes input sequences bidirectionally, creating contextualized representations.

**Encoder-decoder**

An architecture using both encoder and decoder components, typically for sequence-to-sequence tasks like translation.

**Few-shot learning**

The ability to perform tasks with only a few demonstration examples provided in the prompt.

**Fine-tuning**

Adapting a pre-trained model to specific tasks or domains through additional training on targeted data.

**FlashAttention**

An IO-aware attention algorithm that reduces memory usage and improves speed through efficient memory access patterns.

**Foundation model**

A large model trained on broad data that can be adapted to many downstream tasks.

**GELU** Gaussian Error Linear Unit, an activation function commonly used in transformer models.

**Gradient checkpointing**

A memory optimization technique that trades computation for memory by recomputing activations during backpropagation.

**Hallucination**

The generation of plausible-sounding but factually incorrect or nonsensical content by language models.

**Hidden dimension**

The size of intermediate representations in transformer feed-forward layers, typically 4x the model dimension.

**In-context learning (ICL)**

The ability of models to learn from examples provided in the prompt without updating weights.

**Inference** The process of using a trained model to generate predictions or outputs for new inputs.

**Instruction tuning**

Fine-tuning models on datasets of instructions paired with desired responses to improve instruction-following.

**Key-value (KV) cache**

Stored key and value projections from previous tokens during autoregressive generation, avoiding redundant computation.

**Layer normalization**

A normalization technique that standardizes activations across features, improving training stability.

**Learning rate**

A hyperparameter controlling the step size of parameter updates during optimization.

**LoRA**

Low-Rank Adaptation, a parameter-efficient fine-tuning method that adds trainable low-rank matrices to frozen model weights.

**Loss function**

A function measuring the difference between model predictions and target values, guiding the optimization process.

**Masked language modeling (MLM)**

A pre-training objective where models predict randomly masked tokens in the input, used in BERT-style models.

**Mixed precision training**

Training using lower precision (FP16/BF16) for most computations while maintaining FP32 for stability-critical operations.

**Model parallelism**

Distributing model parameters across multiple devices, enabling training of models too large for single devices.

**Multi-head attention**

Attention mechanism using multiple parallel attention heads, each learning different relationship patterns.

**Multi-query attention (MQA)**

An efficient attention variant where key-value projections are shared across attention heads.

**Next-token prediction**

The fundamental objective of autoregressive language models: predicting the probability distribution over the next token.

**Optimizer** An algorithm for updating model parameters based on computed gradients, such as Adam or AdamW.

**Parameter**

A learnable weight in a neural network, adjusted during training to minimize the loss function.

**Perplexity** A metric measuring how well a language model predicts a held-out dataset, calculated as the exponential of average cross-entropy.

**Pipeline parallelism**

Distributing model layers across devices and processing micro-batches in a pipeline fashion.

**Position encoding**

A mechanism for injecting sequence position information into transformer models, which otherwise lack positional awareness.

**Pre-training**

The initial training phase on large unlabeled datasets, learning general language understanding before task-specific fine-tuning.

**Prompt** The input text provided to a language model that conditions its generation or response.

**Prompt engineering**

The practice of designing effective prompts to elicit desired behaviors from language models.

**Quantization**

Reducing the numerical precision of model weights to decrease memory usage and improve inference speed.

**Reinforcement learning from human feedback (RLHF)**

Training models using human preference signals through reinforcement learning.

**Retrieval-augmented generation (RAG)**

Combining language models with external retrieval systems to ground responses in retrieved documents.

**RoPE** Rotary Position Embedding, a relative position encoding method that encodes positions through rotation of embeddings.

**Scaling laws**

Empirical relationships describing how model performance improves with increases in model size, data, and compute.

**Self-attention**

An attention mechanism where queries, keys, and values all come from the same sequence.

**Softmax** A function that converts a vector of real numbers into a probability distribution.

**Speculative decoding**

An inference acceleration technique that drafts multiple tokens with a smaller model and verifies them with the larger model.

**SwiGLU** An activation function combining swish activation with gated linear units, commonly used in modern transformers.

**Temperature**

A hyperparameter controlling the randomness of sampling during generation; higher values produce more diverse outputs.

**Tensor parallelism**

Partitioning individual layers across devices, requiring high-bandwidth interconnects between devices.

**Token** The basic unit of text processing in language models, typically a word or subword piece.

**Tokenizer** A component that converts raw text into sequences of tokens and vice versa.

**Top-k sampling**

A decoding strategy that samples from the k most probable next tokens.

**Top-p (nucleus) sampling**

A decoding strategy that samples from the smallest set of tokens whose cumulative probability exceeds p.

**Transformer**

The dominant neural network architecture for language models, based on self-attention mechanisms.

**Warmup** A training strategy that gradually increases the learning rate at the beginning of training for stability.

**Weight decay**

A regularization technique that adds a penalty on large weights to prevent overfitting.

**Zero-shot learning**

Performing tasks without any task-specific examples, relying solely on pre-trained knowledge and instruction understanding.

**ZeRO** Zero Redundancy Optimizer, a memory optimization technique that partitions optimizer states, gradients, and parameters across devices.

# Index of models

## **Anthropic**

- Claude 1, Claude 2, Claude 3 (Haiku, Sonnet, Opus), Claude 3.5 Sonnet

## **Google/DeepMind**

- BERT, T5, PaLM, PaLM 2, Gemini (Nano, Pro, Ultra), Gemini 1.5, Gemma

## **Meta**

- LLaMA, Llama 2, Llama 3, Code Llama

## **Microsoft**

- Phi-1, Phi-1.5, Phi-2, Phi-3, Orca, WizardLM

## **Mistral AI**

- Mistral 7B, Mixtral 8x7B, Mixtral 8x22B

## **OpenAI**

- GPT-1, GPT-2, GPT-3, GPT-3.5, GPT-4, GPT-4 Turbo, GPT-4o, Codex

## **Other Organizations**

- BLOOM (BigScience), Falcon (TII), Qwen (Alibaba), Yi (01.AI), DeepSeek, InternLM, StarCoder



# Bibliography

- [1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, *Advances in neural information processing systems* **33**, 1877 (2020).
- [2] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, *et al.*, arXiv preprint arXiv:2206.07682 [10.48550/arXiv.2206.07682](#) (2022).
- [3] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, *et al.*, arXiv preprint arXiv:2108.07258 [10.48550/arXiv.2108.07258](#) (2021).
- [4] S. F. Chen and J. Goodman, in *Proceedings of the 34th annual meeting on Association for Computational Linguistics* (1996) pp. 310–318.
- [5] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, in *Journal of machine learning research*, Vol. 3 (2003) pp. 1137–1155.
- [6] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur, *Interspeech* **2**, 1045 (2010).
- [7] S. Hochreiter and J. Schmidhuber, *Neural computation* **9**, 1735 (1997).
- [8] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, arXiv preprint arXiv:1406.1078 [10.48550/arXiv.1406.1078](#) (2014).
- [9] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, arXiv preprint arXiv:1602.02410 [10.48550/arXiv.1602.02410](#) (2016).
- [10] D. Bahdanau, K. Cho, and Y. Bengio, arXiv preprint arXiv:1409.0473 [10.48550/arXiv.1409.0473](#) (2014).
- [11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Advances in neural information processing systems* **30**, [10.48550/arXiv.1706.03762](#) (2017).
- [12] H. Hays, arXiv preprint arXiv:2601.03329 [10.48550/arXiv.2601.03329](#) (2025).
- [13] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, arXiv preprint arXiv:1802.05365 [10.48550/arXiv.1802.05365](#) (2018).
- [14] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, [\(2018\)](#).
- [15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, arXiv preprint arXiv:1810.04805 [10.48550/arXiv.1810.04805](#) (2018).
- [16] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, OpenAI blog **1**, 9 (2019).
- [17] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, *The Journal of Machine Learning Research* **21**, 5485 (2020).
- [18] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, arXiv preprint arXiv:2303.08774 [10.48550/arXiv.2303.08774](#) (2023).
- [19] Anthropic, Anthropic Technical Report (2024).
- [20] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Sorber, J. Schalkwyk, A. M. Dai, A. Hauth, *et al.*, arXiv preprint arXiv:2312.11805 [10.48550/arXiv.2312.11805](#) (2023).

- [21] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, arXiv preprint arXiv:2001.08361 [10.48550/arXiv.2001.08361](https://arxiv.org/abs/2001.08361) (2020).
- [22] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, *et al.*, arXiv preprint arXiv:2203.15556 [10.48550/arXiv.2203.15556](https://arxiv.org/abs/2203.15556) (2022).
- [23] I. Beltagy, M. E. Peters, and A. Cohan, arXiv preprint arXiv:2004.05150 [10.48550/arXiv.2004.05150](https://arxiv.org/abs/2004.05150) (2020).
- [24] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, *et al.*, *Advances in neural information processing systems* **33**, 17283 (2020).
- [25] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, *Advances in Neural Information Processing Systems* **35**, 16344 (2022).
- [26] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, arXiv preprint arXiv:2104.09864 [10.48550/arXiv.2104.09864](https://arxiv.org/abs/2104.09864) (2021).
- [27] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, arXiv preprint arXiv:2305.13245 [10.48550/arXiv.2305.13245](https://arxiv.org/abs/2305.13245) (2023).
- [28] T. Dao, arXiv preprint arXiv:2307.08691 [10.48550/arXiv.2307.08691](https://arxiv.org/abs/2307.08691) (2023).
- [29] R. Sennrich, B. Haddow, and A. Birch, arXiv preprint arXiv:1508.07909 [10.48550/arXiv.1508.07909](https://arxiv.org/abs/1508.07909) (2015).
- [30] OpenAI, arXiv preprint arXiv:2303.08774 [10.48550/arXiv.2303.08774](https://arxiv.org/abs/2303.08774) (2023).
- [31] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, arXiv preprint arXiv:1907.11692 [10.48550/arXiv.1907.11692](https://arxiv.org/abs/1907.11692) (2019).
- [32] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, arXiv preprint arXiv:1909.11942 [10.48550/arXiv.1909.11942](https://arxiv.org/abs/1909.11942) (2019).
- [33] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, arXiv preprint arXiv:2003.10555 [10.48550/arXiv.2003.10555](https://arxiv.org/abs/2003.10555) (2020).
- [34] P. He, X. Liu, J. Gao, and W. Chen, arXiv preprint arXiv:2006.03654 [10.48550/arXiv.2006.03654](https://arxiv.org/abs/2006.03654) (2020).
- [35] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, arXiv preprint arXiv:1910.13461 [10.48550/arXiv.1910.13461](https://arxiv.org/abs/1910.13461) (2019).
- [36] Y. Tay, M. Dehghani, V. Q. Tran, X. Garcia, J. Wei, X. Wang, H. W. Chung, D. Bahri, T. Schuster, H. S. Zheng, *et al.*, arXiv preprint arXiv:2205.05131 [10.48550/arXiv.2205.05131](https://arxiv.org/abs/2205.05131) (2022).
- [37] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et al.*, arXiv preprint arXiv:2302.13971 [10.48550/arXiv.2302.13971](https://arxiv.org/abs/2302.13971) (2023).
- [38] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, *et al.*, arXiv preprint arXiv:2307.09288 [10.48550/arXiv.2307.09288](https://arxiv.org/abs/2307.09288) (2023).
- [39] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, *et al.*, arXiv preprint arXiv:2204.02311 [10.48550/arXiv.2204.02311](https://arxiv.org/abs/2204.02311) (2022).
- [40] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittweiser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, *et al.*, *Science* **378**, 1092 (2022).
- [41] R. Taylor, M. Kardas, G. Cucurull, T. Scialom, A. Hartshorn, E. Saravia, A. Poulton, V. Kerkez, and R. Stojnic, arXiv preprint arXiv:2211.09085 [10.48550/arXiv.2211.09085](https://arxiv.org/abs/2211.09085) (2022).
- [42] H. Hays, Y. Yu, and W. J. Richardson, arXiv preprint arXiv:2512.00696 [10.48550/arXiv.2512.00696](https://arxiv.org/abs/2512.00696) (2024).
- [43] W. Fedus, B. Zoph, and N. Shazeer, *The Journal of Machine Learning Research* **23**, 5232 (2022).

- [44] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat, *et al.*, arXiv preprint arXiv:2112.06905 [10.48550/arXiv.2112.06905](https://arxiv.org/abs/2112.06905) (2021).
- [45] D. P. Kingma and J. Ba, arXiv preprint arXiv:1412.6980 [10.48550/arXiv.1412.6980](https://arxiv.org/abs/1412.6980) (2014).
- [46] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, **SC20: International Conference for High Performance Computing, Networking, Storage and Analysis**, 1 (2020).
- [47] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, arXiv preprint arXiv:2106.09685 [10.48550/arXiv.2106.09685](https://arxiv.org/abs/2106.09685) (2021).
- [48] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, arXiv preprint arXiv:2109.01652 [10.48550/arXiv.2109.01652](https://arxiv.org/abs/2109.01652) (2021).
- [49] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi, arXiv preprint arXiv:2212.10560 [10.48550/arXiv.2212.10560](https://arxiv.org/abs/2212.10560) (2022).
- [50] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et al.*, **Advances in Neural Information Processing Systems** **35**, 27730 (2022).
- [51] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon, *et al.*, arXiv preprint arXiv:2212.08073 [10.48550/arXiv.2212.08073](https://arxiv.org/abs/2212.08073) (2022).
- [52] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, arXiv preprint arXiv:2305.18290 [10.48550/arXiv.2305.18290](https://arxiv.org/abs/2305.18290) (2023).
- [53] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, **ACM Computing Surveys** **55**, 1 (2023).
- [54] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, arXiv preprint arXiv:2402.07927 [10.48550/arXiv.2402.07927](https://arxiv.org/abs/2402.07927) (2024).
- [55] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, and D. Zhou, **Advances in Neural Information Processing Systems** **35**, 24824 (2022).
- [56] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, **Advances in Neural Information Processing Systems** **35**, 22199 (2022).
- [57] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, arXiv preprint arXiv:2203.11171 [10.48550/arXiv.2203.11171](https://arxiv.org/abs/2203.11171) (2022).
- [58] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, **Advances in Neural Information Processing Systems** **36**, 10.48550/arXiv.2305.10601 (2024).
- [59] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. Le, *et al.*, arXiv preprint arXiv:2205.10625 [10.48550/arXiv.2205.10625](https://arxiv.org/abs/2205.10625) (2022).
- [60] T. Khot, H. Trivedi, M. Finlayson, Y. Fu, K. Richardson, P. Clark, and A. Sabharwal, arXiv preprint arXiv:2210.02406 [10.48550/arXiv.2210.02406](https://arxiv.org/abs/2210.02406) (2022).
- [61] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, *et al.*, **Advances in Neural Information Processing Systems** **36**, 10.48550/arXiv.2303.17651 (2024).
- [62] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, **Advances in Neural Information Processing Systems** **36**, 10.48550/arXiv.2303.11366 (2024).
- [63] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, arXiv preprint arXiv:1804.07461 [10.48550/arXiv.1804.07461](https://arxiv.org/abs/1804.07461) (2018).
- [64] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, **Advances in neural information processing systems** **32**, 10.48550/arXiv.1905.00537 (2019).
- [65] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, *et al.*, arXiv preprint arXiv:2110.14168 [10.48550/arXiv.2110.14168](https://arxiv.org/abs/2110.14168) (2021).

- [66] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, arXiv preprint arXiv:2103.03874 [10.48550/arXiv.2103.03874](https://arxiv.org/abs/2103.03874) (2021).
- [67] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, *et al.*, arXiv preprint arXiv:2407.21783 [10.48550/arXiv.2407.21783](https://arxiv.org/abs/2407.21783) (2024).
- [68] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, *et al.*, arXiv preprint arXiv:2107.03374 [10.48550/arXiv.2107.03374](https://arxiv.org/abs/2107.03374) (2021).
- [69] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, *et al.*, arXiv preprint arXiv:2308.12950 [10.48550/arXiv.2308.12950](https://arxiv.org/abs/2308.12950) (2023).
- [70] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li, *et al.*, arXiv preprint arXiv:2401.14196 [10.48550/arXiv.2401.14196](https://arxiv.org/abs/2401.14196) (2024).
- [71] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, arXiv preprint arXiv:2009.03300 [10.48550/arXiv.2009.03300](https://arxiv.org/abs/2009.03300) (2020).
- [72] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, *et al.*, arXiv preprint arXiv:2310.06825 [10.48550/arXiv.2310.06825](https://arxiv.org/abs/2310.06825) (2023).
- [73] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, *et al.*, arXiv preprint arXiv:2211.09110 [10.48550/arXiv.2211.09110](https://arxiv.org/abs/2211.09110) (2022).
- [74] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, arXiv preprint arXiv:1904.09675 [10.48550/arXiv.1904.09675](https://arxiv.org/abs/1904.09675) (2019).
- [75] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, arXiv preprint arXiv:1909.08053 [10.48550/arXiv.1909.08053](https://arxiv.org/abs/1909.08053) (2019).
- [76] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kasber, A. Pedram, J. Kim, *et al.*, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* , 1 (2021).
- [77] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* , 3505 (2020).
- [78] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borber, *et al.*, *ACM SIGARCH Computer Architecture News* **45**, 1 (2017).
- [79] J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobhahn, and P. Villalobos, arXiv preprint arXiv:2202.05924 [10.48550/arXiv.2202.05924](https://arxiv.org/abs/2202.05924) (2022).
- [80] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, *et al.*, *Advances in Neural Information Processing Systems* **32**, [10.48550/arXiv.1811.06965](https://arxiv.org/abs/1811.06965) (2019).
- [81] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, *Proceedings of the 27th ACM Symposium on Operating Systems Principles* , 1 (2019).
- [82] E. Strubell, A. Ganesh, and A. McCallum, arXiv preprint arXiv:1906.02243 [10.48550/arXiv.1906.02243](https://arxiv.org/abs/1906.02243) (2019).
- [83] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, arXiv preprint arXiv:2104.10350 [10.48550/arXiv.2104.10350](https://arxiv.org/abs/2104.10350) (2021).
- [84] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, *Journal of Machine Learning Research* **24**, 1 (2023).
- [85] K. Lee, D. Ippolito, A. Nystrom, C. Zhang, D. Eck, C. Callison-Burch, and N. Carlini, arXiv preprint arXiv:2107.06499 [10.48550/arXiv.2107.06499](https://arxiv.org/abs/2107.06499) (2022).
- [86] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi, arXiv preprint arXiv:2212.10560 [10.48550/arXiv.2212.10560](https://arxiv.org/abs/2212.10560) (2023).

- [87] O. Sainz, J. A. Campos, I. García-Ferrero, J. Etxaniz, O. L. de Lacalle, and E. Agirre, arXiv preprint arXiv:2310.18018 [10.48550/arXiv.2310.18018](https://arxiv.org/abs/2310.18018) (2023).
- [88] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, Z. Wang, R. Y. Y. Wong, Z. Chen, D. Arfeen, R. Abhyankar, and Z. Jia, arXiv preprint arXiv:2312.15234 [10.48550/arXiv.2312.15234](https://arxiv.org/abs/2312.15234) (2023).
- [89] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, *Proceedings of the 29th Symposium on Operating Systems Principles*, 611 (2023).
- [90] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, *et al.*, *Advances in Neural Information Processing Systems* **33**, 9459 (2020).
- [91] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, *et al.*, arXiv preprint arXiv:2311.05232 [10.48550/arXiv.2311.05232](https://arxiv.org/abs/2311.05232) (2023).
- [92] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang, arXiv preprint arXiv:2312.10997 [10.48550/arXiv.2312.10997](https://arxiv.org/abs/2312.10997) (2023).
- [93] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, arXiv preprint arXiv:2310.11511 [10.48550/arXiv.2310.11511](https://arxiv.org/abs/2310.11511) (2023).
- [94] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, arXiv preprint arXiv:2004.04906 [10.48550/arXiv.2004.04906](https://arxiv.org/abs/2004.04906) (2020).
- [95] G. Izacard, M. Caron, L. Hosseini, S. Riedel, P. Bojanowski, A. Joulin, and E. Grave, arXiv preprint arXiv:2112.09118 [10.48550/arXiv.2112.09118](https://arxiv.org/abs/2112.09118) (2021).
- [96] L. Wang, N. Yang, X. Huang, B. Jiao, L. Yang, D. Jiang, R. Majumder, and F. Wei, arXiv preprint arXiv:2212.03533 [10.48550/arXiv.2212.03533](https://arxiv.org/abs/2212.03533) (2022).
- [97] S. Xiao, Z. Liu, P. Zhang, and N. Muennighoff, arXiv preprint arXiv:2309.07597 [10.48550/arXiv.2309.07597](https://arxiv.org/abs/2309.07597) (2023).
- [98] M. Günther, J. Milliken, J. Ky, *et al.*, arXiv preprint arXiv:2310.19923 [10.48550/arXiv.2310.19923](https://arxiv.org/abs/2310.19923) (2023).
- [99] Z. Li, X. Zhang, Y. Zhang, D. Long, P. Xie, and M. Zhang, arXiv preprint arXiv:2308.03281 [10.48550/arXiv.2308.03281](https://arxiv.org/abs/2308.03281) (2023).
- [100] S. Robertson and H. Zaragoza, *Foundations and Trends in Information Retrieval* **3**, 333 (2009).
- [101] T. Formal, B. Piwowarski, and S. Clinchant, arXiv preprint arXiv:2107.05720 [10.48550/arXiv.2107.05720](https://arxiv.org/abs/2107.05720) (2021).
- [102] Y. A. Malkov and D. A. Yashunin, *IEEE transactions on pattern analysis and machine intelligence* **42**, 824 (2018).
- [103] J. Johnson, M. Douze, and H. Jégou, *IEEE Transactions on Big Data* **7**, 535 (2019).
- [104] J. Wang, X. Yi, R. Guo, H. Jin, P. Xu, S. Li, X. Wang, X. Guo, C. Li, X. Xu, *et al.*, *Proceedings of the 2021 International Conference on Management of Data*, 2614 (2021).
- [105] L. Gao, X. Ma, J. Lin, and J. Callan, arXiv preprint arXiv:2212.10496 [10.48550/arXiv.2212.10496](https://arxiv.org/abs/2212.10496) (2022).
- [106] W. Sun, L. Yan, X. Ma, P. Ren, D. Yin, and Z. Ren, arXiv preprint arXiv:2304.09542 [10.48550/arXiv.2304.09542](https://arxiv.org/abs/2304.09542) (2023).
- [107] O. Khattab and M. Zaharia, *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, 39 (2020).
- [108] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, *Transactions of the Association for Computational Linguistics* **12**, 157 (2024).
- [109] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, arXiv preprint arXiv:2309.15217 [10.48550/arXiv.2309.15217](https://arxiv.org/abs/2309.15217) (2023).

- [110] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson, arXiv preprint arXiv:2404.16130 [10.48550/arXiv.2404.16130](https://arxiv.org/abs/2404.16130) (2024).
- [111] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, *et al.*, *Frontiers of Computer Science* **18**, 186345 (2024).
- [112] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou, *et al.*, arXiv preprint arXiv:2309.07864 [10.48550/arXiv.2309.07864](https://arxiv.org/abs/2309.07864) (2023).
- [113] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, arXiv preprint arXiv:2210.03629 [10.48550/arXiv.2210.03629](https://arxiv.org/abs/2210.03629) (2022).
- [114] L. Wang, W. Xu, Y. Lan, Z. Hu, Y. Lan, R. K.-W. Lee, and E.-P. Lim, arXiv preprint arXiv:2305.04091 [10.48550/arXiv.2305.04091](https://arxiv.org/abs/2305.04091) (2023).
- [115] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, Advances in Neural Information Processing Systems **36**, [10.48550/arXiv.2303.11366](https://arxiv.org/abs/2303.11366) (2024).
- [116] A. Zhou, K. Yan, M. Shlapentokh-Rothman, H. Wang, and Y.-X. Wang, arXiv preprint arXiv:2310.04406 [10.48550/arXiv.2310.04406](https://arxiv.org/abs/2310.04406) (2023).
- [117] T. Sumers, S. Yao, K. Narasimhan, and T. Griffiths, arXiv preprint arXiv:2309.02427 [10.48550/arXiv.2309.02427](https://arxiv.org/abs/2309.02427) (2023).
- [118] C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica, and J. E. Gonzalez, arXiv preprint arXiv:2310.08560 [10.48550/arXiv.2310.08560](https://arxiv.org/abs/2310.08560) (2023).
- [119] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, Advances in Neural Information Processing Systems **36**, [10.48550/arXiv.2302.04761](https://arxiv.org/abs/2302.04761) (2024).
- [120] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, *Proceedings of the IEEE/CVF International Conference on Computer Vision* , 2998 (2023).
- [121] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, arXiv preprint arXiv:2305.14325 [10.48550/arXiv.2305.14325](https://arxiv.org/abs/2305.14325) (2023).
- [122] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, *et al.*, arXiv preprint arXiv:2308.08155 [10.48550/arXiv.2308.08155](https://arxiv.org/abs/2308.08155) (2023).
- [123] J. Yang, C. E. Jimenez, A. Wettig, K. Liber, S. Y. K. Narasimhan, and O. Press, arXiv preprint arXiv:2405.15793 [10.48550/arXiv.2405.15793](https://arxiv.org/abs/2405.15793) (2024).
- [124] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang, *et al.*, arXiv preprint arXiv:2308.03688 [10.48550/arXiv.2308.03688](https://arxiv.org/abs/2308.03688) (2023).
- [125] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, arXiv preprint arXiv:2310.06770 [10.48550/arXiv.2310.06770](https://arxiv.org/abs/2310.06770) (2023).
- [126] T. Räuker, A. Ho, S. Casper, and D. Hadfield-Menell, arXiv preprint arXiv:2207.13243 [10.48550/arXiv.2207.13243](https://arxiv.org/abs/2207.13243) (2023).
- [127] D. Marr, *Vision: A computational investigation into the human representation and processing of visual information* (MIT press, 1982).
- [128] C. Burns, H. Ye, D. Klein, and J. Steinhardt, arXiv preprint arXiv:2212.03827 [10.48550/arXiv.2212.03827](https://arxiv.org/abs/2212.03827) (2022).
- [129] J. Vig and Y. Belinkov, arXiv preprint arXiv:1906.04284 [10.48550/arXiv.1906.04284](https://arxiv.org/abs/1906.04284) (2019).
- [130] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, arXiv preprint arXiv:1906.04341 [10.48550/arXiv.1906.04341](https://arxiv.org/abs/1906.04341) (2019).
- [131] C. Olsson, N. Elhage, N. Nanda, N. Joseph, N. DasSarma, T. Henighan, B. Mann, A. Askell, Y. Bai, A. Chen, *et al.*, arXiv preprint arXiv:2209.11895 [10.48550/arXiv.2209.11895](https://arxiv.org/abs/2209.11895) (2022).

- [132] S. Jain and B. C. Wallace, arXiv preprint arXiv:1902.10186 [10.48550/arXiv.1902.10186](https://arxiv.org/abs/1902.10186) (2019).
- [133] N. Elhage, N. Nanda, C. Olsson, T. Henighan, N. Joseph, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, *et al.*, Transformer Circuits Thread (2021).
- [134] C. Olah, N. Cammarata, L. Schubert, G. Goh, M. Petrov, and S. Carter, Distill **5**, e00024 (2020).
- [135] H. Cunningham, A. Ewart, L. Riggs, R. Huben, and L. Sharkey, arXiv preprint arXiv:2309.08600 [10.48550/arXiv.2309.08600](https://arxiv.org/abs/2309.08600) (2023).
- [136] T. Bricken, A. Templeton, J. Batson, B. Chen, A. Jermyn, T. Conerly, N. Turner, C. Anil, C. Denison, A. Askell, *et al.*, Transformer Circuits Thread (2023).
- [137] K. Meng, D. Bau, A. Andonian, and Y. Belinkov, Advances in Neural Information Processing Systems **35**, 17359 (2022).
- [138] M. Geva, J. Bastings, K. Filippova, and A. Globerson, arXiv preprint arXiv:2304.14767 [10.48550/arXiv.2304.14767](https://arxiv.org/abs/2304.14767) (2023).
- [139] K. Meng, A. Sen Sharma, A. Andonian, Y. Belinkov, and D. Bau, arXiv preprint arXiv:2210.07229 [10.48550/arXiv.2210.07229](https://arxiv.org/abs/2210.07229) (2022).
- [140] N. Nanda and J. Bloom, GitHub repository (2022).
- [141] M. Awais, M. Naseer, S. Khan, R. M. Anwer, H. Cholakkal, M. Shah, M.-H. Yang, and F. S. Khan, arXiv preprint arXiv:2307.13721 [10.48550/arXiv.2307.13721](https://arxiv.org/abs/2307.13721) (2023).
- [142] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.*, International Conference on Machine Learning , 8748 (2021).
- [143] C. Jia, Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, Q. Le, Y.-H. Sung, Z. Li, and T. Duerig, International Conference on Machine Learning , 4904 (2021).
- [144] X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer, Proceedings of the IEEE/CVF International Conference on Computer Vision , 11975 (2023).
- [145] H. Liu, C. Li, Q. Wu, and Y. J. Lee, Advances in Neural Information Processing Systems **36**, [10.48550/arXiv.2304.08485](https://arxiv.org/abs/2304.08485) (2024).
- [146] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, arXiv preprint arXiv:2010.11929 [10.48550/arXiv.2010.11929](https://arxiv.org/abs/2010.11929) (2020).
- [147] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, Proceedings of the IEEE/CVF international conference on computer vision , 10012 (2021).
- [148] Y. Fang, W. Wang, B. Xie, Q. Sun, L. Wu, X. Wang, T. Huang, X. Wang, and Y. Cao, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition , 19358 (2023).
- [149] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, *et al.*, arXiv preprint arXiv:2304.07193 [10.48550/arXiv.2304.07193](https://arxiv.org/abs/2304.07193) (2023).
- [150] Y. Xu, M. Li, L. Cui, S. Huang, F. Wei, and M. Zhou, Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining , 1192 (2020).
- [151] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, International Conference on Machine Learning , 28492 (2023).
- [152] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, Advances in neural information processing systems **33**, 12449 (2020).
- [153] C. Wang, S. Chen, Y. Wu, Z. Zhang, L. Zhou, S. Liu, Z. Chen, Y. Liu, H. Wang, J. Li, *et al.*, arXiv preprint arXiv:2301.02111 [10.48550/arXiv.2301.02111](https://arxiv.org/abs/2301.02111) (2023).
- [154] Z. Tong, Y. Song, J. Wang, and L. Wang, Advances in neural information processing systems **35**, 10078 (2022).

- [155] H. Zhang, X. Li, and L. Bing, arXiv preprint arXiv:2306.02858 [10.48550/arXiv.2306.02858](https://arxiv.org/abs/2306.02858) (2023).
- [156] M. Reid, N. Savinov, D. Teber, A. Collins, *et al.*, arXiv preprint arXiv:2403.05530 [10.48550/arXiv.2403.05530](https://arxiv.org/abs/2403.05530) (2024).
- [157] J. Lu, C. Clark, R. Zellers, R. Mottaghi, and A. Kembhavi, arXiv preprint arXiv:2206.08916 [10.48550/arXiv.2206.08916](https://arxiv.org/abs/2206.08916) (2022).
- [158] Y. Zhang, K. Gong, K. Zhang, H. Li, Y. Qiao, W. Ouyang, and X. Yue, arXiv preprint arXiv:2307.10802 [10.48550/arXiv.2307.10802](https://arxiv.org/abs/2307.10802) (2023).
- [159] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10684 (2022).
- [160] S. Y. Gadre, G. Ilharco, A. Fang, J. Hayase, G. Smber, S. Maini, T. Thrush, R. Jha, S. Kornblith, M. Wortsman, *et al.*, Advances in Neural Information Processing Systems **36**, [10.48550/arXiv.2304.14108](https://arxiv.org/abs/2304.14108) (2024).
- [161] J. Li, D. Li, S. Savarese, and S. Hoi, *International conference on machine learning*, 19730 (2023).
- [162] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh, *Proceedings of the IEEE international conference on computer vision*, 2425 (2015).
- [163] M. Geva, R. Schuster, J. Berant, and O. Levy, arXiv preprint arXiv:2012.14913 [10.48550/arXiv.2012.14913](https://arxiv.org/abs/2012.14913) (2021).
- [164] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel, arXiv preprint arXiv:1909.01066 [10.48550/arXiv.1909.01066](https://arxiv.org/abs/1909.01066) (2019).
- [165] E. Mitchell, C. Lin, A. Bosselut, C. Finn, and C. D. Manning, arXiv preprint arXiv:2110.11309 (2021).
- [166] K. Li, A. K. Hopkins, D. Bau, F. Viégas, H. Pfister, and M. Wattenberg, arXiv preprint arXiv:2210.13382 [10.48550/arXiv.2210.13382](https://arxiv.org/abs/2210.13382) (2023).
- [167] G. I. Winata, A. Madotto, Z. Lin, R. Liu, J. Yosinski, and P. Fung, arXiv preprint arXiv:2109.07684 [10.48550/arXiv.2109.07684](https://arxiv.org/abs/2109.07684) (2021).
- [168] A. Conneau and G. Lample, Advances in neural information processing systems **32**, [10.48550/arXiv.1901.07291](https://arxiv.org/abs/1901.07291) (2019).
- [169] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov, arXiv preprint arXiv:1911.02116 [10.48550/arXiv.1911.02116](https://arxiv.org/abs/1911.02116) (2019).
- [170] L. Xue, N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua, and C. Raffel, arXiv preprint arXiv:2010.11934 [10.48550/arXiv.2010.11934](https://arxiv.org/abs/2010.11934) (2020).
- [171] T. L. Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé, *et al.*, arXiv preprint arXiv:2211.05100 (2022).
- [172] M. R. Costa-jussà, J. Cross, O. Çelebi, M. Elbayad, K. Heafield, K. Heffernan, E. Kalbassi, J. Lam, D. Licht, J. Maillard, *et al.*, arXiv preprint arXiv:2207.04672 [10.48550/arXiv.2207.04672](https://arxiv.org/abs/2207.04672) (2022).
- [173] J. Hu, S. Ruder, A. Siddhant, G. Neubig, O. Firat, and M. Johnson, *International Conference on Machine Learning*, 4411 (2020).
- [174] X. Zhu, J. Li, Y. Liu, C. Ma, and W. Wang, arXiv preprint arXiv:2308.07633 [10.48550/arXiv.2308.07633](https://arxiv.org/abs/2308.07633) (2023).
- [175] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, arXiv preprint arXiv:2210.17323 [10.48550/arXiv.2210.17323](https://arxiv.org/abs/2210.17323) (2022).
- [176] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, *Machine Learning and Systems* **6**, 87 (2024).

- [177] E. Frantar and D. Alistarh, [International Conference on Machine Learning](#) , 10323 (2023).
- [178] A. Gu and T. Dao, arXiv preprint arXiv:2312.00752 [10.48550/arXiv.2312.00752](https://arxiv.org/abs/2312.00752) (2023).
- [179] B. Peng, E. Alcaide, Q. Anthony, A. Alber, S. Arcadinho, H. Cao, X. Cheng, M. Chung, M. Grber, K. Grber, *et al.*, arXiv preprint arXiv:2305.13048 [10.48550/arXiv.2305.13048](https://arxiv.org/abs/2305.13048) (2023).
- [180] N. Shazeer, arXiv preprint arXiv:1911.02150 [10.48550/arXiv.1911.02150](https://arxiv.org/abs/1911.02150) (2019).
- [181] L. Floridi, J. Cowls, M. Beltrametti, R. Chatila, P. Chazerand, V. Dignum, C. Luetge, R. Madelin, U. Pagallo, F. Rossi, *et al.*, [Minds and machines](#) **28**, 689 (2018).
- [182] European Parliament and Council, Regulation (eu) 2024/1689 of the european parliament and of the council laying down harmonised rules on artificial intelligence (artificial intelligence act), Official Journal of the European Union (2024).
- [183] The White House, [Federal Register](#) (2023).
- [184] H. Roberts, J. Cowls, J. Morley, M. Taddeo, V. Wang, and L. Floridi, [AI & Society](#) **36**, 59 (2021).