100 Questions About Large Language Models

Saman Siadati

November 2024

100 Questions About Large Language Models

Edition 1.1
DOI: https://doi.org/10.5281/zenodo.15818027

Preface

The field of artificial intelligence (AI) is evolving at an extraordinary pace, and among its most transformative innovations are Large Language Models (LLMs). These models—powering chatbots, search engines, code generators, and more—are changing how we work, learn, and interact with technology. As LLMs become increasingly embedded in everyday applications, understanding how they work is no longer a luxury—it's a necessity.

Let me briefly share my own journey. I earned a Bachelor's degree in Applied Mathematics over two decades ago. Early in my career, I worked as a statistical data analyst on a range of software projects. My path eventually led me into data mining, and later into the field of data science. Over time, as I observed the rise of deep learning and language models, I realized that success in this space requires not only mathematical fluency but also a clear conceptual grasp of model architectures, training paradigms, and the practical challenges involved in deploying these systems responsibly.

This realization inspired me to write this book: a practical, accessible guide structured as 100 essential questions and answers about LLMs. Rather than a dense academic textbook, this book offers focused explanations designed to build intuition and clarity. Each question is crafted to explain a core idea, demystify a technique, or illuminate a challenge—whether you're new to the field or deepening your expertise.

You are welcome to use, share, or adapt any part of this book as you see fit. If you find it helpful, I only ask that you cite it—entirely at your discretion. This book is provided freely, with the hope of supporting your growth and curiosity in the ever-expanding world of large language models.

Saman Siadati
November 2024

Contents

Chapter 1

Core Concepts and Theory

Question 1:
What is a Large Language Model (LLM)?
A Large Language Model is a type of AI model trained to understand and generate human-like text. It learns statistical patterns in language by processing massive corpora of text data. LLMs are typically built using transformer architectures and have billions of parameters.

Question 2:
How do LLMs differ from traditional machine learning models?
LLMs are designed for unstructured textual data and can generalize across many language tasks, unlike traditional models which are often narrow, domain-specific, and require handcrafted features. LLMs also leverage deep neural networks with pretraining and fine-tuning paradigms.

Question 3:
What are transformers, and why are they important for LLMs?
Transformers are a type of neural network architecture that use self-attention mechanisms to process input sequences in parallel. Their scalability and ability to model context efficiently make them the foundation of most modern LLMs.

Question 4:
Explain the concept of self-attention in transformers.
Self-attention allows each word in a sequence to weigh and attend to all other words when creating contextual embeddings. This helps the model capture relationships regardless of distance in the sequence.

Question 5:
What is the difference between supervised, unsupervised, and self-supervised learning?

- **Supervised:** Learns from labeled data.
- **Unsupervised:** Finds patterns in unlabeled data.
- **Self-supervised:** Uses parts of the data to supervise other parts (e.g., predicting missing words), which is how LLMs are pretrained.

Question 6:
What is transfer learning, and how is it applied in LLMs?
Transfer learning involves pretraining a model on a large dataset and fine-tuning it on a smaller, task-specific dataset. LLMs use this to adapt to specific applications after general pretraining.

Question 7:
How does pretraining differ from fine-tuning?
Pretraining exposes the model to general text data to learn language understanding, while fine-tuning adapts the model to specific tasks (e.g., translation, summarization) using labeled examples.

Question 8:
What are embeddings and why are they crucial in LLMs?
Embeddings are vector representations of words or tokens. They capture semantic and syntactic properties and are fundamental for enabling models to process text numerically.

Question 9:
What is the role of positional encoding in transformers?
Since transformers process input in parallel, positional encoding provides the model with information about the order of tokens, enabling it to learn sequence structure.

Question 10:
How do LLMs handle out-of-vocabulary (OOV) words?

LLMs use subword tokenization techniques like byte pair encoding (BPE) or WordPiece to break unknown words into known subunits, ensuring any word can be represented.

Question 11:
Explain the concept of tokenization and its types (word, subword, byte-pair encoding).
Tokenization converts text into units (tokens):

- **Word-level:** each word is a token.
- **Subword-level:** splits words into smaller parts (e.g., un+believable).
- **Byte-pair encoding (BPE):** merges frequent pairs to create a flexible token set.

Question 12:
What are the challenges in scaling LLMs?
Challenges include computational cost, energy consumption, memory requirements, latency, data quality, and maintaining alignment with human values.

Question 13:
What is model capacity and how does it affect LLM performance?
Model capacity refers to the number of parameters and layers. Higher capacity allows learning more complex patterns, but increases resource needs and risk of overfitting.

Question 14:
How do LLMs represent context?
LLMs use self-attention mechanisms to build contextual representations, where each token's meaning is shaped by surrounding tokens, preserving semantics.

Question 15:
What is zero-shot and few-shot learning in LLMs?

- **Zero-shot:** Performing tasks without explicit examples.
- **Few-shot:** Performing tasks with only a few examples provided as part of the prompt.

Question 16:
What are the main sources of bias in language models?
Bias can stem from the training data (reflecting social stereotypes), model architecture, or deployment context. Even balanced data can produce skewed outputs.

Question 17:
How can you mitigate bias in LLMs?
Bias can be reduced through diverse datasets, debiasing algorithms, fairness constraints, and ongoing evaluation with fairness-focused benchmarks.

Question 18:
What are ethical concerns surrounding LLM deployment?
Concerns include misinformation, harmful content generation, surveillance, copyright violations, privacy breaches, and job displacement.

Question 19:
What is catastrophic forgetting and how is it addressed?
Catastrophic forgetting is when a model forgets previous knowledge when trained on new tasks. Techniques like elastic weight consolidation, rehearsal, or modular architectures help prevent this.

Question 20:
How do LLMs model long-range dependencies?
LLMs use multi-head self-attention to link tokens across long contexts. Attention spans are limited by input size, but techniques like memory layers and recurrence extensions help extend this range.

Chapter 2

Architecture and Algorithms

Question 1:
Describe the architecture of the Transformer model.
The Transformer architecture consists of layers that each include multi-head self-attention and feed-forward neural networks, followed by normalization and residual connections. The model processes input in parallel and captures context through self-attention.

Question 2:
What are the differences between encoder-only, decoder-only, and encoder-decoder transformer models?
Encoder-only models (e.g., BERT) focus on understanding tasks, decoder-only models (e.g., GPT) generate text, and encoder-decoder models (e.g., T5) are suited for tasks like translation by encoding input and decoding output.

Question 3:
How does the GPT architecture work?
GPT is a decoder-only transformer using causal self-attention to generate text autoregressively, predicting the next token based on all previous tokens.

Question 4:
Explain multi-head attention and why multiple heads are used.
Multi-head attention allows the model to focus on different parts of the input sequence simultaneously, capturing various relationships and patterns across positions.

Question 5:

What are feed-forward neural networks in the transformer?
Each transformer block contains a position-wise feed-forward network (FFN) applied independently to each token, adding nonlinear transformations and depth to the model.

Question 6:
How does layer normalization help training?
Layer normalization stabilizes training by normalizing inputs across features, leading to faster convergence and reduced internal covariate shift.

Question 7:
What is the difference between vanilla RNNs, LSTMs, and transformers?
Vanilla RNNs and LSTMs process sequences sequentially, while transformers use self-attention for parallel processing, enabling better handling of long-range dependencies and scalability.

Question 8:
What is beam search and how is it used in LLMs?
Beam search is a decoding algorithm that keeps multiple best sequences (beams) during generation, improving output quality by exploring several paths.

Question 9:
Explain temperature and top-k sampling in language generation.
Temperature controls randomness in predictions (lower is more deterministic), while top-k sampling restricts predictions to the top k most probable tokens.

Question 10:
What are the trade-offs between greedy decoding and sampling?
Greedy decoding is faster and deterministic but may yield repetitive outputs, while sampling introduces diversity at the cost of potential incoherence.

Question 11:
How does masked language modeling work?
Masked language modeling randomly masks some input tokens and trains the model to predict them, enabling bidirectional understanding of context (used in BERT).

Question 12:
What is causal language modeling?
Causal language modeling predicts each token using only past context, enforcing autoregressive generation (used in GPT).

Question 13:
How is dropout used in training transformers?
Dropout randomly deactivates neurons during training to prevent overfitting and improve model generalization.

Question 14:
What are position-wise feed-forward networks?
These are two-layer fully connected networks applied independently to each position, allowing token-wise transformation after attention.

Question 15:
Explain the concept of residual connections.
Residual connections skip layers and add the original input back to the output, helping gradients flow and stabilizing deep model training.

Question 16:
What is the role of attention masks during training and inference?
Attention masks control which tokens a model attends to. They prevent looking at future tokens during training or ignore padding during batch processing.

Question 17:
How do transformers handle variable length inputs?
Transformers handle variable input lengths using padding tokens and attention masks to ensure uniform processing and ignore irrelevant parts.

Question 18:
What is a transformer's receptive field?
The receptive field refers to the span of input tokens that influence a given output. In transformers, this is determined by the self-attention window and context length.

Question 19:
How do you implement gradient clipping? Why is it necessary?

Gradient clipping limits the magnitude of gradients during backpropagation to prevent exploding gradients, ensuring stable and effective training.

Question 20:
What are the pros and cons of model parallelism vs data parallelism in LLM training?
Model parallelism splits the model across devices, useful for large models, but increases communication overhead. Data parallelism replicates the model and splits data, enabling efficient scaling but constrained by model size.

Chapter 3

Training and Optimization

Question 1:
How do you train an LLM from scratch?
Training an LLM from scratch involves preparing a large, diverse dataset, tokenizing it, initializing model parameters, and optimizing the model using gradient descent over many iterations on powerful hardware.

Question 2:
What are common datasets used for training LLMs?
Common datasets include OpenWebText, Common Crawl, Wikipedia, BooksCorpus, and curated corpora like The Pile, which offer diverse and extensive language examples.

Question 3:
How do you handle tokenization during training?
Tokenization converts raw text into token IDs using methods like Byte Pair Encoding (BPE) or WordPiece. This step ensures uniform input format across the dataset.

Question 4:
What loss functions are used for training LLMs?
The cross-entropy loss is commonly used, measuring how well the predicted probability distribution aligns with the true token distribution.

Question 5:
What optimizers are commonly used (e.g., Adam, AdamW) and why?

Adam and AdamW are widely used due to their adaptive learning rates and stability. AdamW improves regularization by decoupling weight decay from gradient updates.

Question 6:
Explain learning rate scheduling for training transformers.
Learning rate scheduling involves adjusting the learning rate during training—commonly starting with a warm-up phase followed by a linear or cosine decay.

Question 7:
How do you handle extremely large datasets efficiently?
Efficient handling involves streaming data, using token buffers, distributed file systems, and preprocessing data into tokenized batches.

Question 8:
What techniques help speed up training (mixed precision, gradient checkpointing)?
Mixed precision reduces memory usage by using float16 operations, while gradient checkpointing saves memory by recomputing activations during backpropagation.

Question 9:
What is distributed training and how is it implemented?
Distributed training splits the workload across GPUs or nodes using data parallelism, model parallelism, or pipeline parallelism with tools like DeepSpeed or PyTorch DDP.

Question 10:
How do you detect and fix training instability?
Instability is monitored through loss spikes or divergence and mitigated via gradient clipping, smaller learning rates, or improved normalization techniques.

Question 11:
What are warm-up steps in training?
Warm-up steps gradually increase the learning rate from a low value to stabilize early training and prevent large, destabilizing gradient updates.

Question 12:
How do you monitor overfitting during training?
Overfitting is monitored by tracking performance on a validation set. A rising gap between training and validation loss often indicates overfitting.

Question 13:
What is early stopping?
Early stopping halts training when validation performance no longer improves, preventing overfitting and saving compute resources.

Question 14:
How can you reduce catastrophic forgetting during continual learning?
Techniques include elastic weight consolidation, rehearsal of old examples, or using modular network architectures that isolate learning.

Question 15:
How is validation done for LLMs?
Validation involves running the model on a hold-out dataset and computing metrics like loss, perplexity, or task-specific scores.

Question 16:
What is knowledge distillation and how is it used with LLMs?
Knowledge distillation transfers knowledge from a large model (teacher) to a smaller model (student), maintaining performance with reduced size.

Question 17:
Explain gradient accumulation and its use cases.
Gradient accumulation sums gradients over multiple mini-batches before updating weights, enabling training with large effective batch sizes under memory constraints.

Question 18:
What are common regularization techniques used for transformers?
Regularization methods include dropout, weight decay, attention dropout, and label smoothing to improve generalization.

Question 19:

How do you evaluate the quality of generated text during training?
Evaluation uses metrics like perplexity, BLEU, ROUGE, or human evaluation to assess fluency, coherence, and relevance of outputs.

Question 20:
What are the challenges of training LLMs on multi-lingual data?
Challenges include imbalanced data distribution across languages, token sharing issues, and varying syntactic/semantic structures that affect model alignment.

Chapter 4

Evaluation and Metrics

Question 1:
How do you evaluate the performance of an LLM?
Performance is typically evaluated using intrinsic metrics like perplexity and extrinsic task-specific metrics such as BLEU, ROUGE, or accuracy. Human evaluation is also used for assessing output quality.

Question 2:
What is perplexity? How is it calculated and interpreted?
Perplexity measures how well a language model predicts a sample. It is calculated as the exponential of the cross-entropy loss. Lower perplexity indicates better performance.

Question 3:
Explain BLEU, ROUGE, and METEOR metrics.
BLEU compares n-gram overlap between generated and reference texts. **ROUGE** measures recall of overlapping units like n-grams and sequences. **METEOR** considers semantic similarity, synonyms, and stemming.

Question 4:
What are the limitations of automated evaluation metrics for text generation?
Automated metrics often fail to capture semantic correctness, coherence, and creativity. They can be misleading if used without human validation.

Question 5:
How do you evaluate LLMs for tasks like summarization, translation,

or Q&A?
Task-specific metrics are used: ROUGE for summarization, BLEU for translation, and exact match or F1 for Q&A. Human evaluation complements automated scores.

Question 6:
What is human evaluation in the context of LLMs?
Human evaluation involves subjective judgment of output quality based on fluency, coherence, relevance, factual correctness, and user satisfaction.

Question 7:
How do you evaluate fairness and bias in LLMs?
Bias evaluation includes testing with benchmark datasets, measuring disparities in responses across demographics, and analyzing stereotype amplification.

Question 8:
What metrics assess factual consistency in generated text?
Metrics include FactCC, FEVER scores, entailment models, or using retrieval-based methods to compare outputs with knowledge sources.

Question 9:
How do you measure diversity in generated outputs?
Diversity is measured by metrics such as distinct-n (unique n-grams), entropy, or self-BLEU, which penalize repetitive or generic text.

Question 10:
What is zero-shot evaluation?
Zero-shot evaluation assesses a model's ability to perform a task without any task-specific training, relying purely on its pretrained knowledge and generalization capabilities.

Chapter 5

Applications

Question 1:
How can LLMs be used for chatbots and conversational AI?
LLMs generate coherent and contextually appropriate responses, enabling natural conversations. They support open-domain chatbots and task-specific dialogue systems, making interactions more human-like.

Question 2:
Explain how LLMs enable question answering systems.
LLMs use contextual understanding to match questions with relevant answers, supporting extractive and generative QA by retrieving or synthesizing responses from text.

Question 3:
How are LLMs used in text summarization?
LLMs perform abstractive summarization by generating concise representations of input documents, or extractive summarization by selecting key sentences.

Question 4:
What are the challenges of applying LLMs to machine translation?
Challenges include handling low-resource languages, maintaining context across long texts, and ensuring semantic and syntactic accuracy in the target language.

Question 5:
How can LLMs be used for code generation?
LLMs trained on code (e.g., Codex, CodeBERT) generate syntactically correct and semantically meaningful code snippets from natural language prompts or

partial code.

Question 6:
What are use cases of LLMs in content creation?
LLMs assist in writing articles, generating marketing copy, creating dialogue for games, drafting emails, and producing creative content like poetry or scripts.

Question 7:
How do you adapt LLMs for domain-specific tasks?
Adaptation can be done through fine-tuning on domain-specific data or prompt tuning to make LLMs perform better in specialized fields like law or medicine.

Question 8:
What are prompt engineering and few-shot prompting?
Prompt engineering crafts input prompts to guide model behavior. Few-shot prompting includes a few examples in the prompt to teach the task without retraining.

Question 9:
How can LLMs be used in information retrieval?
LLMs improve retrieval by reranking search results, generating summaries, or combining with retrieval-based systems (e.g., RAG) for better accuracy and relevance.

Question 10:
What are challenges in using LLMs for real-time applications?
Real-time use faces challenges in latency, memory usage, response consistency, cost, and ensuring that outputs meet safety and factuality standards.

Chapter 6

Practical and Engineering Considerations

Question 1:
How do you deploy LLMs efficiently?
Efficient deployment involves using model quantization, model distillation, batching requests, and serving models with optimized inference engines like ONNX Runtime, TensorRT, or FasterTransformer.

Question 2:
What hardware is needed for training and inference of LLMs?
Training large LLMs requires high-end GPUs or TPUs with substantial memory (e.g., A100, H100). Inference can be performed on smaller GPUs or even CPUs for lighter workloads with optimization.

Question 3:
How do you optimize inference latency?
Techniques include quantization, knowledge distillation, caching key/value pairs in transformers, efficient batching, and using optimized inference libraries.

Question 4:
What are model quantization and pruning? How do they help?
Quantization reduces model size by lowering numerical precision (e.g., FP32 to INT8), while pruning removes redundant weights. Both improve speed and reduce memory usage.

Question 5:
How do you secure LLM APIs?

Security involves API authentication, rate limiting, input validation, content filtering, and monitoring for abuse or prompt injection attacks.

Question 6:
What techniques exist for handling model updates and versioning?
Use semantic versioning, CI/CD pipelines for deployment, backward compatibility testing, and model registries to manage and track changes safely.

Question 7:
How do you handle prompt injection attacks?
Mitigation strategies include sanitizing inputs, using output filters, adversarial training, enforcing stricter prompting rules, and isolating sensitive operations.

Question 8:
What are best practices for logging and monitoring LLM performance in production?
Monitor latency, throughput, error rates, and response quality. Log prompts and outputs with care for privacy, and use alerts for anomalies or drift detection.

Question 9:
How do you handle multilingual support in deployed LLMs?
Multilingual support requires tokenizers that support Unicode, multilingual training data, evaluation in multiple languages, and user interface localization.

Question 10:
What are the challenges of scaling LLM services?
Challenges include high compute and memory demands, cost management, latency under load, system reliability, and maintaining consistency across distributed instances.

Chapter 7

Research and Advanced Topics

Question 1:
What are the latest advances in transformer architectures?
Recent advances include efficient transformer variants like sparse transformers, Linformers, and Performer, which reduce computational complexity; improvements in scaling laws; and innovations like adaptive attention spans and dynamic routing.

Question 2:
How do you fine-tune LLMs for specialized tasks?
Fine-tuning involves training a pretrained LLM on domain-specific labeled data with task-specific objectives, often with smaller learning rates and early stopping to avoid overfitting.

Question 3:
What is prompt tuning and prefix tuning?
Prompt tuning optimizes a small set of continuous prompt vectors prepended to inputs without updating the entire model. Prefix tuning is a related method that prepends learned vectors (prefixes) to each transformer layer's input to guide generation efficiently.

Question 4:
Explain retrieval-augmented generation (RAG).
RAG combines pretrained LLMs with external retrieval systems to fetch relevant documents during generation, improving factuality and grounding outputs in up-to-date knowledge.

Question 5:
What are foundation models?
Foundation models are large pretrained models that serve as a basis for many downstream tasks, characterized by their generality, scale, and adaptability.

Question 6:
How do generative adversarial networks (GANs) compare to transformers?
GANs use a generator-discriminator framework for generative tasks, mostly in images, while transformers rely on self-attention for sequence modeling. Transformers have largely overtaken GANs in NLP due to better scalability and training stability.

Question 7:
What are sparse transformers and how do they improve efficiency?
Sparse transformers reduce attention complexity by attending only to subsets of tokens rather than all pairs, lowering memory and compute costs, enabling longer context lengths.

Question 8:
Explain multimodal transformers.
Multimodal transformers process and integrate multiple data types (text, images, audio) within a unified architecture, enabling joint reasoning and generation across modalities.

Question 9:
How do you evaluate explainability in LLMs?
Explainability is assessed via techniques like attention visualization, feature importance analysis, probing classifiers, and user studies to understand model decision processes.

Question 10:
What is the future of LLMs in AI?
The future includes continued scaling, improved efficiency, better alignment with human values, multimodal integration, enhanced reasoning abilities, and broader adoption in real-world applications.

Glossary

Attention Mechanism: A component in transformers that allows models to focus on relevant parts of the input when producing an output.

Beam Search: A decoding strategy that keeps track of the top-k most likely sequences at each generation step.

Bias in LLMs: Systematic errors introduced by data, architecture, or training processes that affect model fairness and objectivity.

Byte-Pair Encoding (BPE): A subword tokenization technique that iteratively merges frequent pairs of bytes to handle OOV words.

Causal Language Modeling: Predicts the next token in a sequence using only past tokens.

Decoder: A transformer component that generates output sequences, often used in generative tasks.

Embedding: A vector representation of a word, token, or sentence that captures semantic information.

Encoder: The part of a transformer model that processes and encodes input sequences into context-rich representations.

Feed-Forward Network: A fully connected neural network layer applied to each position in the transformer independently.

Fine-Tuning: Adjusting a pretrained model on task-specific data to improve performance for a particular application.

Foundation Models: Large-scale models pretrained on broad data and designed for general-purpose downstream tasks.

Few-Shot Learning: A paradigm where models learn to perform tasks with only a few examples provided in the prompt.

Gradient Clipping: A technique to prevent exploding gradients by capping the gradient values during backpropagation.

Gradient Accumulation: A method to simulate larger batch sizes by summing gradients over multiple mini-batches before updating weights.

Layer Normalization: A technique to stabilize training by normalizing inputs across the features of each layer.

Long-Range Dependency: Relationships between distant tokens in a sequence; transformers handle these using self-attention.

Masked Language Modeling (MLM): A task where certain tokens are masked and the model learns to predict them (used in BERT).

Model Quantization: Reducing model size and computation by converting weights from high precision (e.g., FP32) to lower precision (e.g., INT8).

Model Pruning: Removing unnecessary weights from a model to improve efficiency with minimal performance loss.

Multi-Head Attention: Allows the model to attend to different parts of the sequence from multiple perspectives simultaneously.

Multimodal Transformers: Models that can handle input from multiple modalities like text, image, and audio.

Out-of-Vocabulary (OOV): Words not present in the training vocabulary; handled via subword methods like BPE.

Positional Encoding: Injects order information into input embeddings so that transformers can model sequence positions.

Prompt Engineering: Designing input prompts to guide LLM behavior for better task performance.

Prompt Injection: A security vulnerability where malicious input alters model behavior or outputs.

Prompt Tuning: A fine-tuning method where only prompt embeddings are trained while the base model remains unchanged.

Receptive Field: The range of input tokens that can influence a given output token in a model.

Residual Connection: Adds input to the output of a layer to improve training stability and gradient flow.

Retrieval-Augmented Generation (RAG): Combines LLMs with document retrieval systems to enhance response accuracy and grounding.

Self-Attention: Enables tokens to attend to each other within a sequence for rich context representation.

Self-Supervised Learning: Learning from unlabeled data by generating training signals from the data itself.

Sparse Transformer: A transformer variant that reduces computation by attending only to a subset of tokens.

Tokenization: Breaking down text into smaller units (tokens) for model processing.

Transformer: A neural network architecture based on attention mechanisms, fundamental to modern LLMs.

Transfer Learning: Leveraging knowledge from a model pretrained on one task or domain to improve performance on another.

Zero-Shot Learning: Performing tasks without seeing examples during training, relying solely on pretrained knowledge.

References

- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI blog, 1(8), 9.
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877-1901.
- Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805.*
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140), 1-67.
- So, D. R., Liang, C., & Le, Q. V. (2021). Primer: Searching for efficient transformers for language modeling. *arXiv preprint arXiv:2109.08668.*
- Liu, J., Wolf, T., & Keskar, N. S. (2023). A Survey of Large Language Models. *arXiv preprint arXiv:2303.18223.*
- Kaplan, J., McCandlish, S., Henighan, T., Brown, T., Chess, B., Child, R., ... & Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361.*

- Bommasani, R., Hudson, D., Adeli, E., Altman, R., Arora, S., von Arx, S., ... & Liang, P. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.
- Bender, E. M., & Friedman, B. (2018). Data statements for natural language processing: Toward mitigating system bias and enabling better science. *Transactions of the Association for Computational Linguistics*, 6, 587–604.
- Siadati, S. (2023). Transformers and Large Language Models. *Zenodo*. https://doi.org/10.5281/zenodo.15687613