ZERO TO HERO
LLM PRETRAINING GUIDE

PRETRAINING A LLM

Training Pipeline

Data Collection

Data Processing

Training LLM

LLM Evaluation

Finetuning LLM

Data Collection

Introduction to Data Collection

Definition: The process of gathering large volumes of diverse text data

Importance: Forms the foundation for training Large Language Models (LLMs)

Goal: Acquire a broad, representative sample of human knowledge and language use

Data Sources

Web crawling: Common Crawl, Custom Web Scraping

Books: Gutenberg, Google Books

Open Source Data Hub: HuggingFace, Kaggle

News: News APIs, Blogs

Domain Related:

- Scientic - Arxiv
- Legal - Govt. Report
- Medical - Pubmed

Considerations

Volume: Typically terabytes of text data

Diversity: Multiple Languages, Domains, Dialects

Quality: Removing Spam & Low Quality content, Data Source Reliability, etc...

Ethical considerations:

- Copyright/License
- Privacy
- Bias

Challenges

- Storage and Processing: Scalable and efficient pipeline
- Deduplication: Complex logic to remove duplicates
- Different File Formats: HTML, PDF, TXT, CSV
- Different File Encodings: UTF-8, ASCII
- Data Relevance: Datasets with updated information
- Legal & Ethics: Copyright laws & Anonymized Fair Usage

Data Preprocessing

Introduction to Data Preprocessing

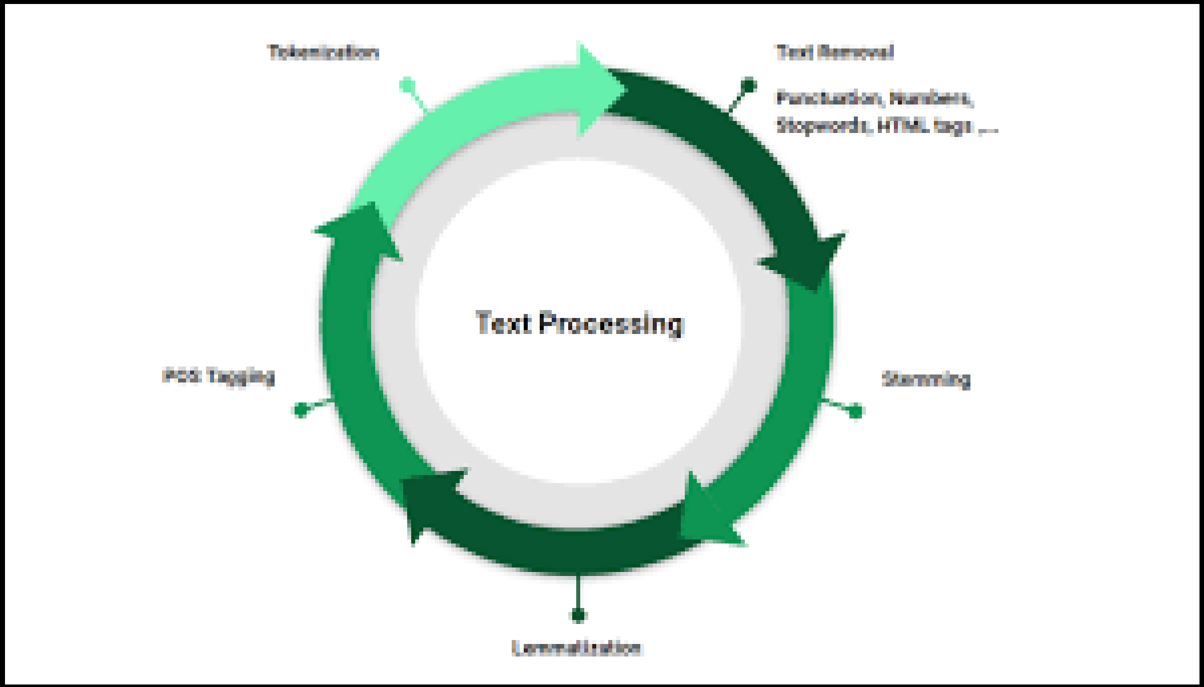
Definition: Transforming raw text data into a clean, structured format suitable for model training

Importance: Ensures data quality, consistency, and relevance

Goals:

- Remove noise & irrelevant information
- Standardize text format
- Prepare data for efficient processing by the model

Data Processing - Text Cleaning



Removal of HTML Tags & Stopwords, Handling special characters, Standardizing text, Deduplication of content

Vocabulary Creation

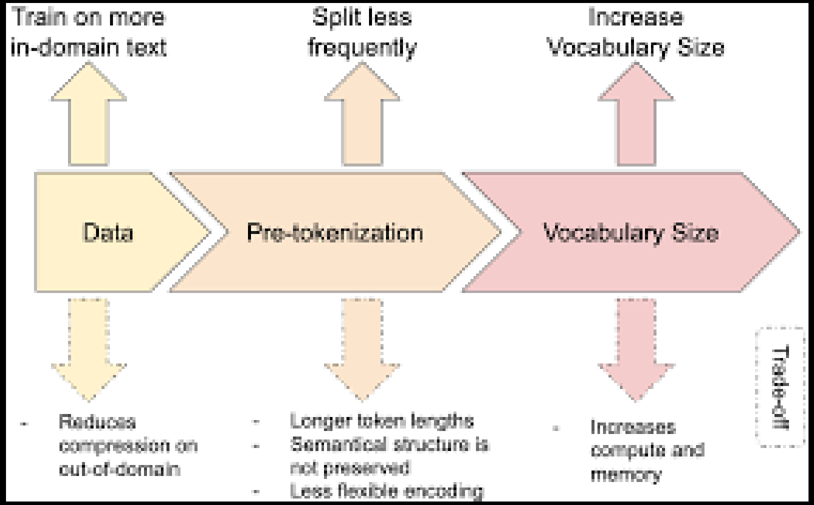
- Compiling unique tokens from the dataset
- Determining vocabulary size
- Handling out-of-vocabulary words
- Special tokens:
- [PAD]: Used for padding sequences to a fixed length
- [UNK]: Represents unknown or rare words
- [CLS]: Often used as a special classification token
- [SEP]: Used to separate different parts of the input

Tokenization

Definition: Breaking text into smaller units (tokens)

Types of tokenization:

- Word-based
- Subword
- Character-based


Train on more in-domain text
Split less frequently
Increase Vocabulary Size
Data
Pre-tokenization
Vocabulary Size
- Reduces compression on out-of-domain
- Longer token lengths
- Semantical structure is not preserved
- Less flexible encoding
- Increases compute and memory
Trade-off

Importance of tokenization in model performance:

- Balancing vocabulary size with token informativeness
- Handling out-of-vocabulary words and rare tokens

Implementing tokenization:

- Pretrained Tokenizers in NLTK, Spacy, HuggingFace
- Custom Tokenizer

Encoding Input

- Converting tokens to numerical IDs: Assigning unique integer IDs to each token in the vocabulary
- Creating input sequences for the model: Raw text to sequences of token IDs handling sentence and document boundaries
- Handling variable-length inputs: Padding, Truncation, Attention masking
- Implementing efficient encoding pipelines:
  - Parallelizing encoding processes for large datasets
  - Optimizing for memory usage and processing speed

Data Augmentation

- Synonym replacement: Using WordNet or word embeddings to find synonyms and randomly replacing words with synonyms increasing vocabulary diversity
- Back-translation: Translating text to another language and back to create paraphrases for generating diverse phrasings while maintaining semantic meaning
- Random insertion/deletion: Inserting random words from the vocabulary and deleting words randomly to simulate different writing styles
- Text generation using larger language models: Using existing large models or APIs to generate additional training data & Filtering generated text for quality and relevance
- Importance of data augmentation:
  - Increases dataset size and diversity
  - Helps in preventing overfitting
  - Improves model robustness to variations in input

Training The LLM

LLM Architecture Designing

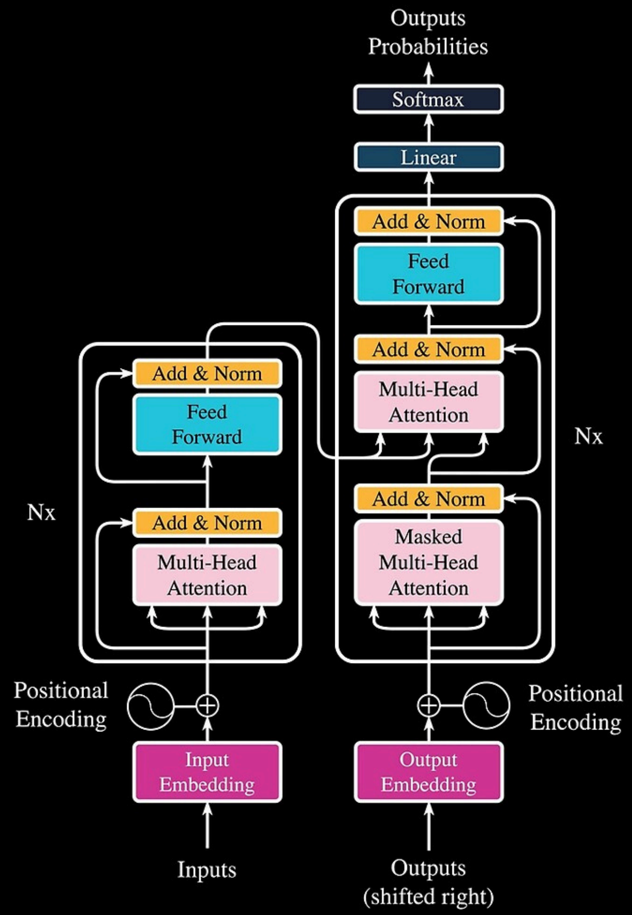
Introduction to LLM Architecture

- Evolution of language model architectures:
    - From n-gram models to neural network-based approaches
    - Breakthrough of the Transformer architecture in 2017
    - OS Models like Llama2, Mistral revolutionized LLMs
- Key design goals:
    - Capturing long-range dependencies in text
    - Efficient processing of large amounts of data
    - Scalability to billions of parameters

Transformer Architecture Overview

Key Components of Transformers:

- Embedding layers: Converting token IDs to dense vector representations
- Positional encodings: Incorporating sequence order information
- Multi-head attention mechanisms: Allowing the model to focus on different parts of the input
- Feed-forward neural networks: Processing the attention output
- Layer normalization: Stabilizing the learning process



Architectural Advancements

- Architecturally there have been numerous advancements to increase in performance and efficiency of architecture
  - Positional Encoding: Absolute, Relative, RoPE, ALiBi, Mixture
  - Attention: Multi Head, Multi Query, Grouped Query, Sliding Window, Disentangled Attention
  - Feed Forward: I: Activation Functions Optimization- GLU activation functions: GeGLU, SwiGLU II: FFN to MOE
- Choice of advancement can be customized as per requirement

Model Scaling

- Increasing model size:
  - Adding more layers to increase the depth
  - Increasing the hidden size to capture more information
  - Trade-offs between model size and computational requirements
- Efficient scaling techniques:
  - Sparse attention mechanisms (e.g., Longformer, BigBird)
  - Parameter sharing across layers
  - Mixture of Experts (MoE) models

LLM Training Process

Introduction to LLM (Pre)Training

Definition: Process of teaching the model to understand and generate language

Importance: Core step in creating a versatile language model

Goals:

- Develop a model with broad language understanding
- Enable zero-shot and few-shot learning capabilities
- Achieve good performance across various NLP tasks

Pretraining Tasks

- Masked Language Modeling (MLM):
  - Training model to predict randomly masked tokens in the input
- Next Sentence Prediction (NSP):
  - Predicting whether two sentences follow each other in the original text
- Causal Language Modeling:
  - Predicting the next token given the previous tokens
  - Used in autoregressive models like GPT

Causal Language Modeling

- What is Causal Language Modeling?
  - A task where the model predicts the next token in a sequence using only preceding tokens.
  - Example: Given "The sky is", predict the next word like "blue."
- Why is it Important?
  - Core to many NLP tasks like text generation, chatbots, and autocompletion.
  - Helps models learn language structure, grammar, and context.

Causal Language Modeling

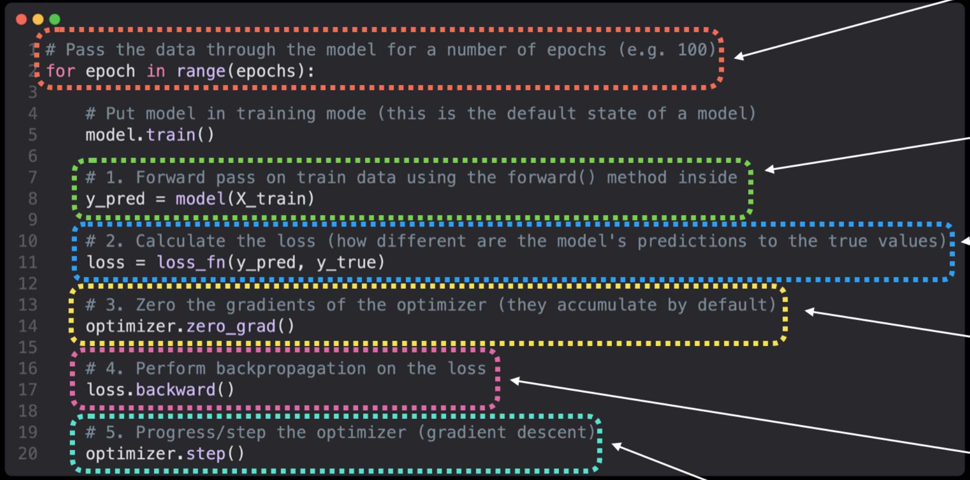
- How Does It Work?
  - Autoregressive process: predicts tokens one-by-one.
  - Trained to minimize cross-entropy loss.
  - Only uses past tokens, with no access to future ones.
- Applications
  - Text generation: creating coherent sequences.
  - Language tools: like autocompletion, summarization
- Challenges: Limited Context and Bias

Loss Functions and Optimizers

- Cross-entropy loss:
  - Standard loss function for classification tasks
  - Applied token-wise in language modeling
- Optimization algorithms:
  - Adam optimizer: Adaptive learning rate method
  - AdamW: Adam with decoupled weight decay
- Learning rate scheduling:
  - Warm-up period: Gradually increasing learning rate
  - Decay strategies: Linear, cosine, or step decay

Training Process Details

PyTorch training loop



```python
# Pass the data through the model for a number of epochs (e.g. 100)
for epoch in range(epochs):

    # Put model in training mode (this is the default state of a model)
    model.train()

    # 1. Forward pass on train data using the forward() method inside
    y_pred = model(X_train)

    # 2. Calculate the loss (how different are the model's predictions to the true values)
    loss = loss_fn(y_pred, y_true)

    # 3. Zero the gradients of the optimizer (they accumulate by default)
    optimizer.zero_grad()

    # 4. Perform backpropagation on the loss
    loss.backward()

    # 5. Progress/step the optimizer (gradient descent)
    optimizer.step()
```

Note: all of this can be turned into a function

Pass the data through the model for a number of epochs (e.g. 100 for 100 passes of the data)

Pass the data through the model, this will perform the `forward()` method located within the model object

Calculate the loss value (how wrong the model's predictions are)

Zero the optimizer gradients (they accumulate every epoch, zero them to start fresh each forward pass)

Perform backpropagation on the loss function (compute the gradient of every parameter with `requires_grad=True`)

Step the optimizer to update the model's parameters with respect to the gradients calculated by `loss.backward()`

Training Process Details

- Batching and input preparation: Dataset and DataLoader:
  - Creating mini-batches of sequences
  - Applying padding and attention masks
- Forward pass:
  - Propagating input through the model
  - Computing model predictions
- Loss calculation:
  - Comparing predictions with ground truth
  - Aggregating loss across the batch

Training Process Details

- Backpropagation:
  - Computing gradients of the loss with respect to model parameters
  - Using automatic differentiation frameworks (e.g., PyTorch, TensorFlow)
- Parameter updates:
  - Applying gradients to update model weights
  - Implementing gradient clipping to prevent exploding gradients

Distributed Training Strategies

- Hardware requirements:
  - High-performance GPUs (e.g., NVIDIA A100) or TPUs
  - Fast interconnects for distributed training (e.g., NVLink, InfiniBand)
  - Large amounts of RAM and fast storage (SSDs)
- Software stack:
  - Deep learning frameworks: PyTorch, TensorFlow
  - Distributed training libraries: DeepSpeed, Megatron-LM
  - Experiment tracking tools: MLflow, Weights & Biases, Tensorboard
- Cloud vs. on-premises infrastructure:
  - Trade-offs between flexibility, cost, and control
  - Leveraging cloud GPUs for scalability

Monitoring and Debugging

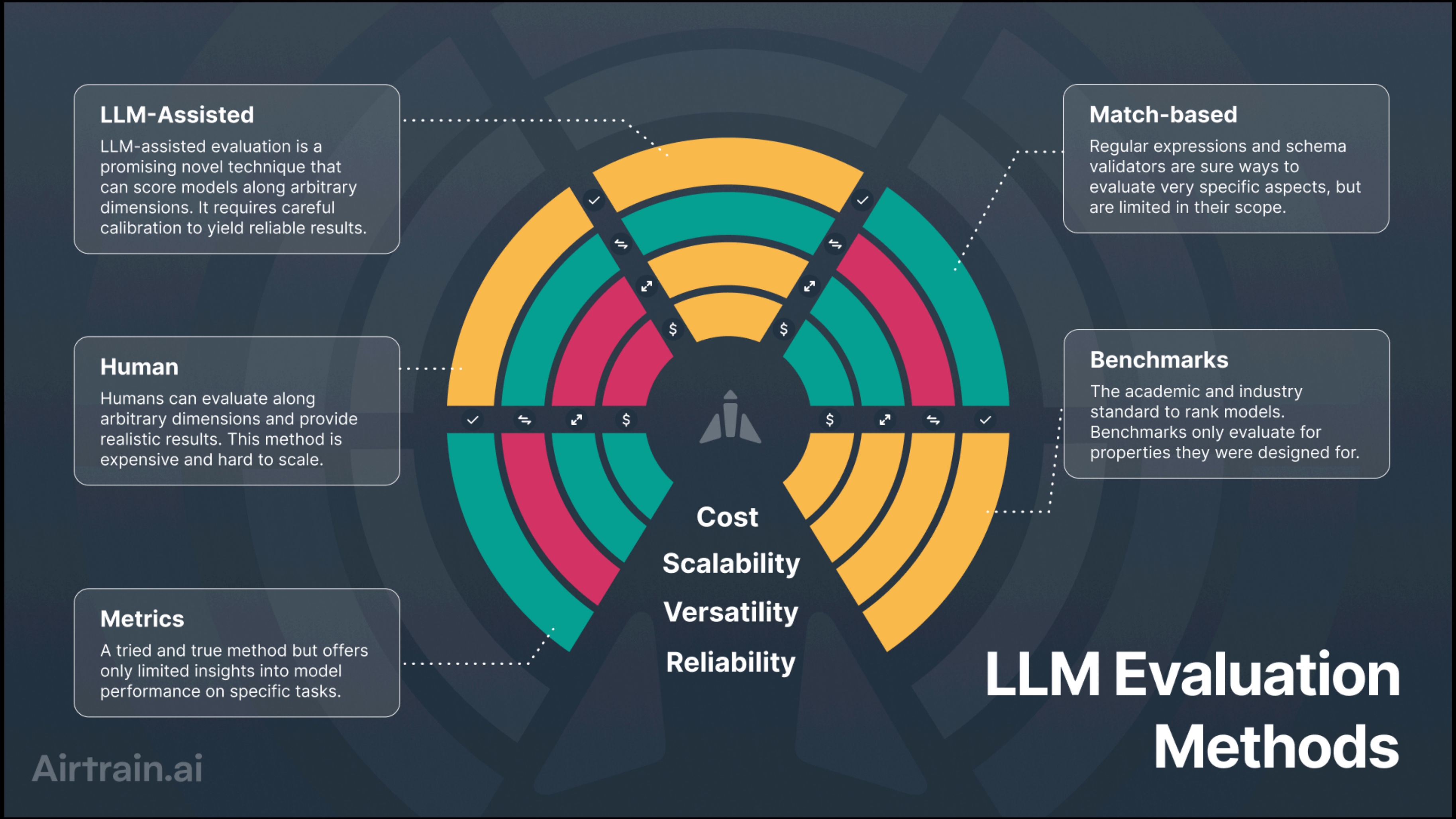
- Training metrics to track:
  - Loss values (training and validation)
  - Perplexity
  - Learning rates
  - Gradient statistics (norm, variance)
- Debugging techniques:
  - Gradient checking
  - Layer-wise relevance propagation
  - Visualization of attention weights
- Handling training instabilities:
  - Gradient accumulation
  - Mixed-precision training
  - Careful initialization of model parameters

Checkpointing and Resuming Training

- Regular model checkpointing:
  - Saving model weights and optimizer states
  - Frequency considerations (balancing safety and I/O overhead)
- Efficient checkpoint formats:
  - Sharded checkpoints for large models
  - Compression techniques to reduce storage requirements
- Resuming training from checkpoints:
  - Handling learning rate schedules and data iterators
  - Ensuring reproducibility of results

LLM Evaluation

LLM Evaluation Methods



LLM-Assisted
LLM-assisted evaluation is a promising novel technique that can score models along arbitrary dimensions. It requires careful calibration to yield reliable results.

Human
Humans can evaluate along arbitrary dimensions and provide realistic results. This method is expensive and hard to scale.

Metrics
A tried and true method but offers only limited insights into model performance on specific tasks.

Match-based
Regular expressions and schema validators are sure ways to evaluate very specific aspects, but are limited in their scope.

Benchmarks
The academic and industry standard to rank models. Benchmarks only evaluate for properties they were designed for.

Importance of LLM Evaluation

- Assessing model performance and capabilities
- Identifying strengths and weaknesses
- Guiding further improvements and iterations
- Ensuring responsible AI development

Perplexity and Loss Metrics

- Perplexity:
  - Definition: Exponential of the cross-entropy loss
  - Interpretation: Lower perplexity indicates better model performance
  - Limitations: Not always correlated with downstream task performance
- Training and validation loss:
  - Monitoring loss curves to detect overfitting
  - Using moving averages for smoother loss tracking

Benchmark Datasets and Leaderboards

- GLUE & SuperGLUE:
  - Suite of 9 tasks testing natural language understanding
  - Includes tasks like sentiment analysis, textual entailment, and question answering
  - SuperGLUE - More challenging extension of GLUE
  - Includes more complex reasoning tasks
- SQuAD (Stanford Question Answering Dataset):
  - Evaluating reading comprehension and question answering capabilities

Benchmark Datasets and Leaderboards

- LAMBADA (Language Modeling Benchmark):
  - Testing the model's ability to understand long-range dependencies
- Multilingual benchmarks:
  - XTREME and XGLUE for cross-lingual understanding
- Emerging benchmarks:
  - BIG-bench: Collaborative benchmark for language model capabilities
  - HELM: Holistic evaluation framework for language models

Task-Specific Evaluation

- Question answering:
  - Metrics: Exact Match (EM) and FI score
  - Datasets: SQuAD, Natural Questions, TriviaQA
  - Evaluating both answer correctness and relevance
- Text summarization:
  - Metrics: ROUGE (Recall-Oriented Understudy for Gisting Evaluation)
  - Assessing content selection, compression, and fluency
  - Datasets: CNN/Daily Mail, XSum, MLSUM (multilingual)

Task-Specific Evaluation

- Machine translation:
  - Metrics: BLEU (Bilingual Evaluation Understudy), METEOR, chrF
  - Evaluating translation quality and preservation of meaning
  - Datasets: WMT (Workshop on Machine Translation) datasets
- Named entity recognition:
  - Metrics: Precision, Recall, FI score
  - Evaluating the model's ability to identify and classify named entities
  - Datasets: CoNLL-2003, OntoNotes 5.0, WNUT

Human Evaluation

- Importance of human judgment:
  - Capturing nuances that automated metrics might miss
  - Assessing overall quality and coherence of model outputs
- Designing evaluation protocols:
  - Defining clear evaluation criteria and guidelines
  - Training evaluators for consistency
- Types of human evaluation:
  - Direct assessment: Rating model outputs on various criteria
  - Comparative evaluation: Ranking outputs from different models
  - Error analysis: Identifying and categorizing model mistakes
- Challenges in human evaluation:
  - Subjectivity and potential bias of human judges
  - Cost and time-intensive nature of the process
  - Scaling human evaluation to large datasets

Ethical Considerations

- Bias detection and mitigation:
  - Evaluating model outputs for gender, racial, or other biases
  - Using datasets designed to probe for biases (e.g., WinoBias, CrowS-Pairs)
- Fairness across different demographic groups:
  - Assessing model performance across various subpopulations
  - Implementing techniques like demographically balanced datasets

Ethical Considerations

- Privacy and security implications:
  - Evaluating the model's tendency to memorize and potentially leak training data
  - Assessing vulnerability to adversarial attacks or prompt injections
- Truthfulness and factual accuracy:
  - Developing benchmarks for factual knowledge (e.g., TruthfulQA)
  - Assessing the model's tendency to generate false or misleading information

Iterative Improvement Process

- Analyzing evaluation results:
    - Identifying patterns in model errors and weaknesses
    - Prioritizing areas for improvement based on project goals
- Data-centric improvements:
    - Augmenting training data in areas of poor performance
    - Cleaning or filtering data to address identified biases

Iterative Improvement Process

- Model-centric improvements:
  - Fine-tuning on specific tasks or domains
  - Experimenting with architectural modifications
- Hyperparameter optimization:
  - Balancing model performance with computational efficiency
- Continuous evaluation:
  - Implementing automated evaluation pipelines
  - Regularly reassessing model performance as it evolves

Finetuning LLM

Introduction to Fine-tuning

- Definition: Adapting a pretrained LLM for a specific task
- Advantages of fine-tuning over training from scratch:
  - Leveraging general language understanding
  - Requiring less task-specific data
  - Faster convergence and potentially better performance
- Overview of task:
  - Goal: Determine the sentiment (positive, negative, neutral) of a given text
  - Applications: Customer feedback analysis, social media, etc...

Preparing Sentiment Dataset

- Collecting labeled sentiment data:
  - Sources: Product reviews, social media posts, movie reviews
  - Popular datasets: IMDB Movie Reviews, Stanford Sentiment Treebank, Amazon Product Reviews
- Data preprocessing for sentiment analysis:
  - Cleaning and normalizing text (similar to pretraining data preparation)
  - Handling domain-specific terms and expressions
- Balancing positive, negative, and neutral samples:
  - Importance of a balanced dataset for unbiased classification
  - Techniques: Oversampling, undersampling, or synthetic data generation (e.g., SMOTE)
- Creating train/validation/test splits:
  - Typically 70-80% train, 10-15% validation, 10-15% test
  - Ensuring representative distribution across splits

Finetuning Process

- Modifying the LLM architecture for classification:
  - Adding a classification head (usually a feed-forward neural network)
  - Utilizing the [CLS] token or pooled representation of the input
- Freezing vs. unfreezing layers:
  - Strategies: Freezing all but the last few layers, gradual unfreezing
  - Trade-offs between adaptation and preserving general knowledge
- Choosing appropriate learning rates:
  - Typically lower learning rates than in pretraining
  - Differential learning rates: Lower for pretrained layers, higher for new layers
- Selecting batch size and number of epochs:
  - Generally smaller batch sizes than in pretraining
  - Monitoring validation performance to determine optimal training duration

Training the Sentiment Classifier

- Loss functions for sentiment classification:
    - Cross-entropy loss for multi-class classification, BCE for binary class classification
    - Focal loss for handling class imbalance
- Handling class imbalance:
    - Weighted loss functions
    - Adjusting class weights in the final layer
- Monitoring training progress:
    - Tracking training and validation loss
    - Monitoring classification accuracy and other relevant metrics
- Early stopping and model selection:
    - Implementing patience-based early stopping
    - Saving the best model based on validation performance

Advanced Fine-tuning Techniques

- Gradual unfreezing:
  - Starting with only the classification head unfrozen
  - Gradually unfreezing earlier layers of the model
- Discriminative fine-tuning:
  - Using different learning rates for different layers of the model
- Mixed precision training:
  - Utilizing lower precision (e.g., float16) for faster training and reduced memory usage
- Knowledge distillation:
  - Training a smaller model to mimic the fine-tuned LLM

Evaluating the Sentiment Classifier

- Metrics for sentiment classification:
  - Accuracy: Overall correctness of predictions
  - Precision, Recall, FI-score: Per-class and macro/micro averaged
  - Confusion matrix: Visualizing classification errors
- Cross-validation techniques:
  - K-fold cross-validation for robust performance estimation
  - Stratified sampling to maintain class distribution
- Error analysis and model interpretation:
  - Analyzing misclassified examples
  - Attention visualization to understand model focus

Deployment Considerations

- Model compression techniques:
    - Pruning: Removing less important weights
    - Quantization: Reducing numerical precision of weights
    - Knowledge distillation: Creating a smaller, deployable model
- Inference optimization:
    - Optimizing for latency vs. throughput
    - Leveraging hardware acceleration (e.g., NVIDIA TensorRT, ONNX Runtime)

Deployment Considerations

- Integration with existing systems:
    - API development for model serving
    - Batch processing vs. real-time inference
- Monitoring and updating the model in production:
    - Implementing logging and monitoring for inference requests
    - Detecting concept drift and performance degradation
    - Strategies for continuous model updating and A/B testing

Challenges &
Future Directions

Challenges in LLM Development

- Computational resources and energy consumption:
  - Addressing the environmental impact of large-scale training
  - Developing more efficient training techniques
- Bias and fairness:
  - Mitigating biases present in training data
  - Ensuring equitable performance across different groups
- Interpretability and explainability:
  - Developing methods to understand model decisions
  - Balancing performance with transparency
- Factual accuracy and hallucination:
  - Reducing the generation of false or misleading information
  - Improving models' grounding in factual knowledge

Trends & Future Directions

- Continual learning:
  - Developing models that can learn new information without forgetting
  - Addressing the challenge of catastrophic forgetting
- Few-shot and zero-shot learning:
  - Improving models' ability to perform new tasks with minimal examples
  - Developing more generalizable language understanding
- Ethical AI and responsible development:
  - Implementing safeguards against misuse
  - Developing frameworks for ethical decision-making in AI systems

Conclusion

Conclusion of Pretraining Guide

- Recap of the LLM pipeline:
  - From data collection to fine-tuned sentiment classifier
- Key takeaways:
  - Importance of high-quality, diverse data
  - Crucial role of careful preprocessing and tokenization
  - Complexity of model architecture and training process
  - Significance of thorough evaluation and iterative improvement
- Future of LLMs:
  - Potential impact on various industries and applications
  - Ongoing research challenges and opportunities