## Introduction

- AI Engineering:
  - Focuses less on modeling and training, and more on model adaption and evaluation
  - Works with models that are bigger and consume more compute resources, and incur higher latency than traditional ML models
- Token:
  - Basic unit of a language model (can be a word, character, or part of a word)
  - Allow models to break words into meaningful components
  - There are fewer unique tokens than unique words
  - Number of tokens in a model's dataset isn't the same as its number of training tokens → the number of training tokens measures the tokens that the model is trained on
- Masked language model: Trained to predict a token anywhere in a sequence, using the context from both before and after the missing token → fill in the blank
- Autoregressive language model:
  - Trained to predict the next token in a sequence, using only the preceding tokens
  - Can continually generate one token after another
- Generative models can generate open-ended outputs
- Classical ML models are closed-ended since they only can outputs that are among predefined values
- Completions are predictions based on probabilities, and not guaranteed to be correct
- Self-supervision:
  - Model infers labels from the input data
  - Language modeling is self-supervised because each input sequence provides both the labels (tokens to be predicted) and the contexts the model can use to predict these labels → labels are inferred from the input data
- Foundation model: Can be built upon for different needs
- Multimodal model: Models that can work with more than one data modality
- Agents: AI that can plan and use tools
- Three layers of AI stack:
  - Application development: Provide a model with good prompts and necessary context
  - Model development: Tooling for developing models, including frameworks for modeling, training, finetuning, and inference optimization
  - Infrastructure: Tooling for model infrastructure, including tooling for model serving, managing data and compute, and monitoring
- Inference optimization: Make models faster and cheaper
- Prompt engineering: Get AI models to express the desirable behaviors from the input alone, without changing the model weights

## Foundation Models

### Sampling

- Process of constructing output
- How does a model chooses an output from all possible options
- Makes an AI's output probabilistic
- To generate the next token, the model first computes the probability distribution over all tokens in the vocabulary

- Greedy sampling: Always pick the most likely outcome → creates boring outputs since the most common words are responded
- Instead of picking the most likely token, the model can sample the next tokens according to the probability distribution
- Sampling strategies:
  - Temperature: A higher temperature reduces the probabilities of common tokens and increases the probability of rarer tokens
  - Top-k: Pick top-k logits to perform softmax over these → smaler k makes the text more predictable but less interesting
  - Top-p:
    - Allows for a more dynamic selection of values to be sampled from than top-k
    - Model sums the probabilities of the most likely next values in descending order and stops when the sum reaches p → only the values within the cumulative probability are considered
    - Focuses only on the set of most relevant values for each context and allows outputs to be more contextually appropriate
    - Typically range from 0.95 to 0.99
- Logit vector:
  - Corresponds to one possible value → one token in the model's vocabulary
  - Logit vector size is the size of the vocabulary
  - Do not represent probabilities
  - To convert logits to probabilities, a softmax layer is often used
- Sparsity allows for more efficient data storage and computation (large percentage of zero-value parameters)
- FLOP: Unit for a model's compute requirement → measures the number of floating point operations performed for a certain task
- Utilization: Measures how much of the maximum compute capacity one can use
- Post-Training:
  - Issues of a pre-trained model:
    - Self-supervision optimizes the model for text completion, not conversations
    - Outputs can be biased or wrong
  - Solution:
    - Supervised finetuning (SFT): finetune the pre-trained model on high-quality instruction data to optimize models for conversations instead of completion
    - Preference finetuning: Finetune the model to output responses that aligh with human preferences
  - Preference finetuning is typically done with reinforcement learning or reinforcement learning from human feedback (RLHF)
  - Pre-training optimizes token-level quality
  - Post-training optimizes the model to generate responses that users prefer
- RLHF:
  - Train a reward model that scores the foundation model's output
  - Optimize the foundation model to generate responses for which the reward model will give maximal scores
- Reward model:
  - Outputs a score for how good the response is
  - Generate multiple responses for each prompt and the resulting data is comparison

data with the format (prompt, winning_response, losing_response)

- Robustness: A model is robust if it doesn't dramatically change its outputs with small variations in the input
- Inconsistency: Generating very different responses for the same or slightly different prompts
- Hallucination:
  - Response that isn't grounded in facts
  - Cannot differentiate between the data it's given and the data it generates
  - Snowball hallucinations: Model continues ti justify the initial wrong assumption
  - Also caused by the mismatch between the model's internal knowledge and the labeler's internal knowledge

## Evaluation Methodology

### Metrics

- Most autoregressive LMs are trained using cross entropy or perplexity (predictive accuracy metrics $\rightarrow$ the more accurately, the lower the metrics)
- Entropy:
  - Measures how much information, on average, a token carries $\rightarrow$ the higher the entropy, the more information
  - Measures how difficult it is to predict what comes next in a language $\rightarrow$ the lower the language's entropy, the more predictable that language
- Cross Entropy on a dataset:
  - When training a LM on a dataset, the goal is to get the model learn the distribution of this training data
  - Measures how difficult it is for the LM to predict what comes next in this dataset
  - An LM is trained to minimize its cross entropy with respect to the training data
  - A model's cross entropy is its approximation of the entropy of its training data
- Bits-per-byte (BPB): Number of bits an LM needs to represent one byte of the original training data
- Perplexity:
  - Exponential of entropy and cross entropy
  - Rules:
    - More structured data gives lower expected perplexity
    - The bigger the vocabulary, the higher the perplexity
    - The longer the context, the lower the perplexity

### Exact Evaluation

- Judgment without ambiguity
- Approaches:
  - Functional Correctness: Evaluating a system based on whether it performs the intended functionality
  - Similarity Measurements Against Reference Data: Evaluate AI's output against reference data
- Ground truth: Reference responses
- Similarity measurement: Asking an evaluator, exact match, lexical similarity, semantic similarity

- Exact match: Generated response matches one of the reference responses exactly
- Lexical similarity:
  - Measures how much two texts overlap → Counting how many tokens two texts have in common
  - Approximate string matching (fuzzy matching): Measures similarity between two texts by counting how many edits it'd need to convert from one text to another (edit distance)
- Semantic similarity: Aims to compute the similarity in semantics, requiring embeddings
- Embeddings: Numerical representation (vector) that aims to capture the meaning of the original data

### AI as Judge

- Using AI to evaluate AI
- Approaches:
  - Evaluate data by itself
  - Compare a generated response to a reference response
  - Compare two generated responses and determine which one is better
- Challenges:
  - AI as a judge criteria aren't standardized
  - AIs are subjective → Evaluation results can change based on the judge model and prompt
  - Probabilistic nature of AI makes it seem too unreliable to act as evaluator
  - Inconsistency: Same judge, on the same input, can output different scores if prompted differently
  - Criteria ambiguity: Misinterpret and misuse judge metrics since they are not standardized
  - Evaluation should be fixed, but AI judges also change over time
  - Self-bias: Model favors own response over response from other models
  - first-position-bias: AI judge my favor the first answer in a pairwise comparison
  - Verbosity bias: Favoring lengthier answers, regardless of their quality
- Judge models:
  - Reward model: Takes in (prompt, response) pair and scores how good the response is given the prompt
  - Reference-based judge: Evaluates the generated response with respect to one or more references
  - Preference model: Takes in (prompt, response 1, response 2) as input and outputs which of the two responses is better for the given prompt

## Evaluate AI Systems

### Evaluation Criteria

- Evaluation-driven development: Define evaluation criteria before building the application
- Evaluation criteria:
  - Domain-specific capability
  - Generation capability
  - Instruction-following capability

  - Cost and latency

### Domain-Specific Capability

- Domain-specific capabilities are commonly evaluated using exact evaluation
- Coding-related capabilities are evaluated using functional correctness
- Efficiency can be evaluated by measuring runtime or memory usage
- Non-coding domain capabilities are often evaluated with close-ended tasks (MCQs) $\rightarrow$ how many questions the model gets right
- Classification metrics: F1 Score, precision, and recall
- MCQs are best suited for evaluating knowledge and reasoning, but not ideal for evaluating generation capabilities

### Generation Capability

- Fluency: Measures whether the text is grammatically correct and natural-sounding (became less important)
- Coherence: How well-structured is the whole text (became less important)
- Faithfulness (for translation task): How faithful is the generated translation to the original sentence?
- Relevance: Does the summary focus on the most important aspects of the source document?
- Factual consistency:
  - Local factual consistency: Output is evaluated against a context and is considered factually consistent if it's supported by the given context
  - Global factual consistency: Output is evaluated against open knowledge
- Most straightforward evaluation approach is AI as a judge:
  - Self-verification: If a model generates multiple outputs that disagree with one anther, the original output is likely hallucinated
  - Knowledge-augmented verification
- Textual entailment: Determine the relationship between two statements

# Prompt Engineering

- Instruction given to a model to perform a task
- Guides a model's behavior without changing the model's weights
- Parts:
  - Task description: What you want the model to do
  - Examples of how to do the task
  - Concrete task
- Chain-of-Thought (CoT): Asking the model to think step by step, reducing hallucinations
- Self-critique (self-eval): Asking the model to check its own outputs

## In-Context Learning: Zero-Shot and Few-Shot

- In-Context Learning:
  - Teaching models what to do via prompts
  - Language models can learn behavior from examples in the prompt
  - Allows a model to incorporate new information continually to make decisions
- Shot: Examples provided in the prompt
- Few-Shot: Teaching a model to learn from examples in the prompt
- Zero-Shot: No example is provided in the prompt

## System Prompt and User Prompt

- System prompt:
  - Task description
  - Contain the instructions provided by the application developers
- User prompt:
  - Task
  - Instructions provided by the user
- System and user prompt are combined by the model into a single prompt, following a template

## Defensive Prompt Engineering

- Main prompt attacks:
  - Prompt extraction: Extracting the application's prompt, including system prompt
  - Jailbreaking and prompt injection
  - Information extraction: Getting the model to reveal its training data
- Reverse prompt engineering:
  - Process of deducing the system prompt used for certain applications
  - Done by analyzing outputs or by tricking the model into repeating its entire prompt, including system prompt

### Jailbreaking and Prompt Injection

- Jailbreaking: Subvert a model's safety features
- Prompt injection: Injection of malicious instructions are injected into the user prompt
- Direct manual prompt hacking: Trick a model into dropping its safety filters (obfuscation, role playing)

### Defenses Against Prompt Attacks

- Model-level defense:
  - Model is unable to differentiate between system instructions and malicious instructions
  - Levels of priority:
    1. System prompt
    2. User prompt
    3. Model outputs
    4. Tool outputs
  - Not only recognize malicious prompts but also generate safe responses for borderline requests (invoke both safe and unsafe responses)
- Prompt-level defense:
  - Be explicit about what the model isn't supposed to do
  - Repeat system prompt twice, both before and after the user prompt
- System-level defense:
  - Isolation
  - Placing guardrails both to the inputs and outputs

## RAG and Agents