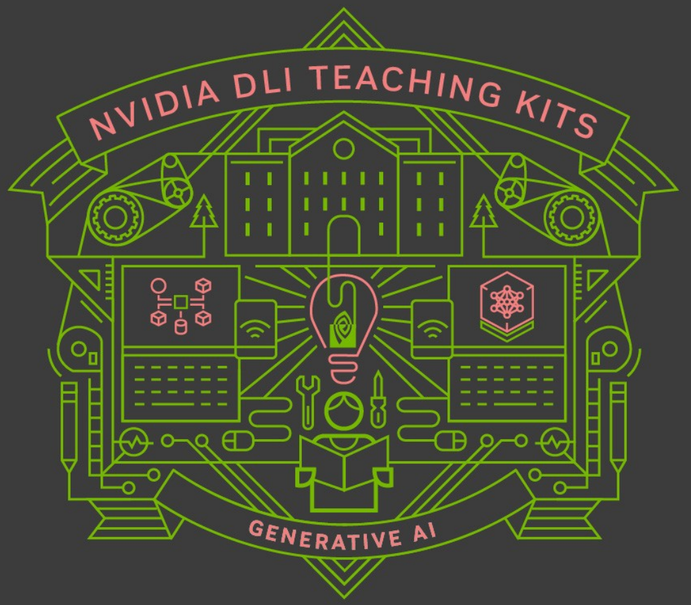
Lecture 3.3 - Autoregressive Training Transformer LLMs

Generative AI Teaching Kit



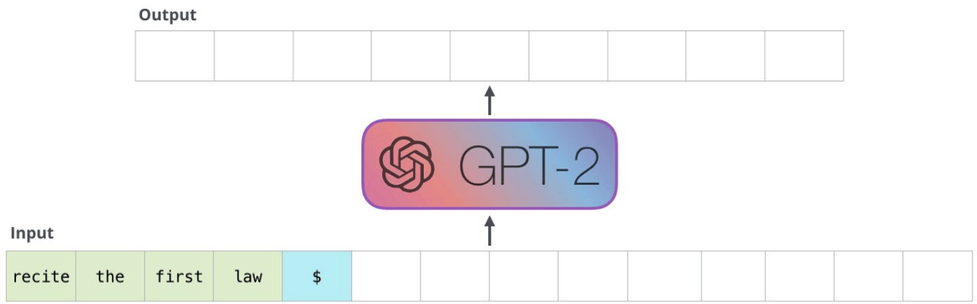
This lecture

- Autoregressive Modeling
- Training Pipeline
- Input Context Considerations
- Challenging in Autoregressive Training

Autoregressive Modeling

Autoregressive Modeling in Language

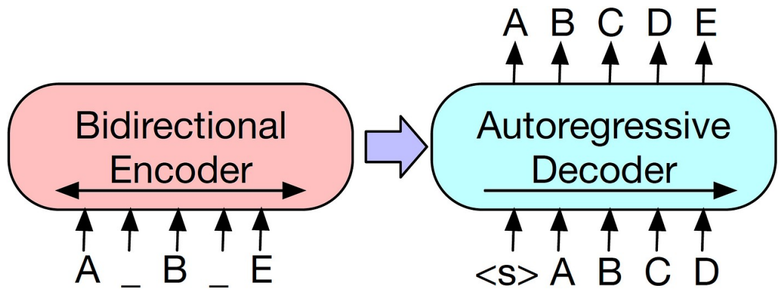
In the last lecture the original transformer was introduced as a machine translation tool. We will see, in the next module, that the Transformer mod can be converted into what is known as an autoregressive model which is used to predict the next token (e.g. ChatGPT)



Autoregressive Modeling**:** Predict the next token in a sequence given all previous tokens.

Transformers are also used in non-autoregressive models, such as bidirectional encoders

These models processes tokens in parallel (e.g., BERT predicts missing words all at once).



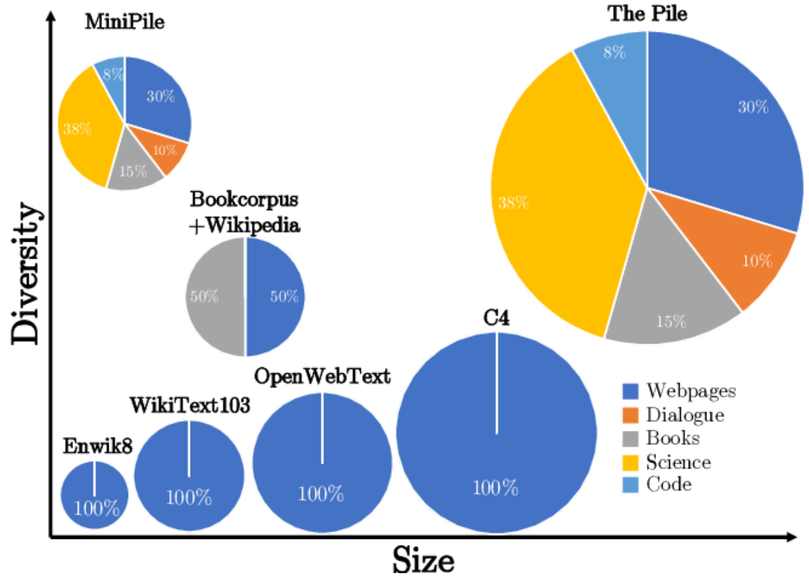
Autoregressive Modeling Challenges

Training Objectives

- Difficulty in balancing long-term dependencies with training efficiency.
- Exposure bias: ensuring the model is only exposed to correct prefixes during training, causing errors to compound during inference.
- Limited by the autoregressive nature, leading to slower token-by-token prediction.

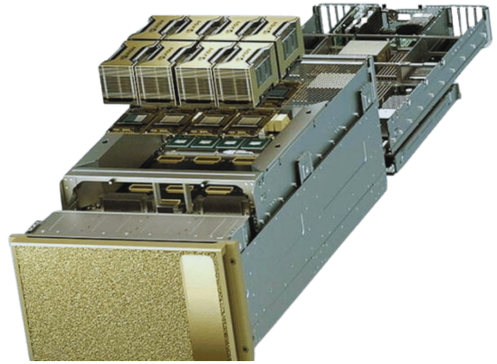
Data Requirements

- Requires extensive, high-quality, and diverse datasets to generalize well.
- Preprocessing and tokenization challenges for multiple languages and domains.
- Bias and representation issues due to skewed data distributions.



Hardware Requirements

- High computational cost due to sequential nature during training and inference.
- Demands for large-scale distributed systems to handle memory and compute requirements.
- Hardware accelerators (e.g., GPUs, TPUs) necessary for efficiency and scalability.



Training Pipeline

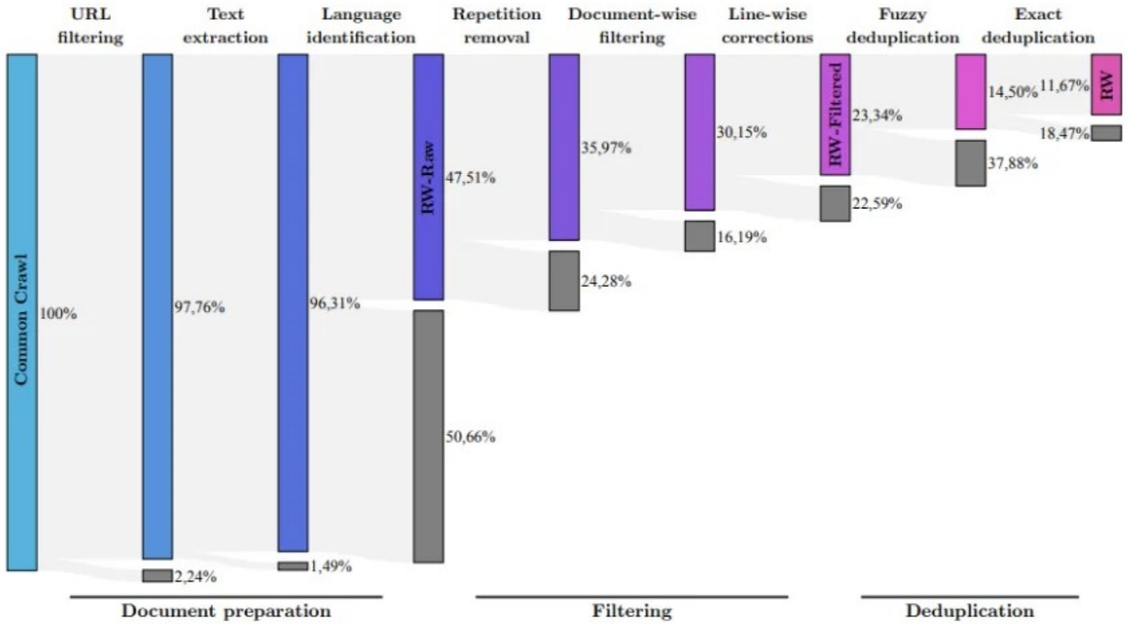
Training for AR Modeling

Data Preprocessing

1. **Tokenization**: Breaking text into tokens (subwords, words, or characters) for sequential modeling.
2. **Normalization**: Standardizing text (e.g., lowercase, removing special characters).
3. **Dataset Splitting**: Creating training, validation, and test sets for robust evaluation.
4. **Sequence Preparation**

Generate input-output pairs:

- Input: Sequence
- Target: next token prediction
  - Padding/truncation for consistent sequence lengths in batches.



Training Objective

- **Next Token Prediction**:
  - Train the model to maximize: $P(x_t|x_1, x_2, \ldots, x_{t-1})$
- **Loss Function**: Negative Log-Likelihood (NLL): Focus on minimizing errors in long-term sequence dependencies.

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \log P(x_t^{(i)}|x_1^{(i)}, x_2^{(i)}, \ldots, x_{t-1}^{(i)})$$

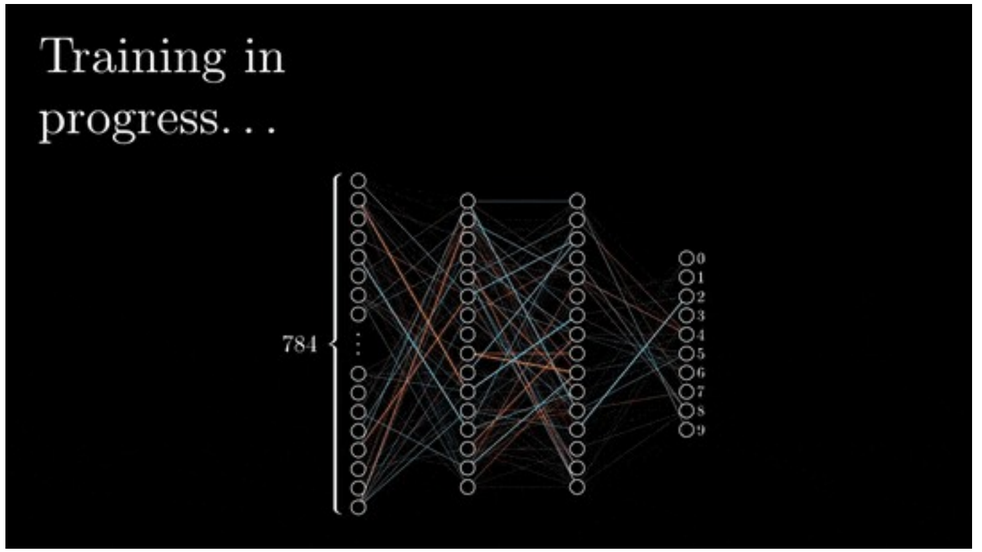
Backpropagation for AR Modeling

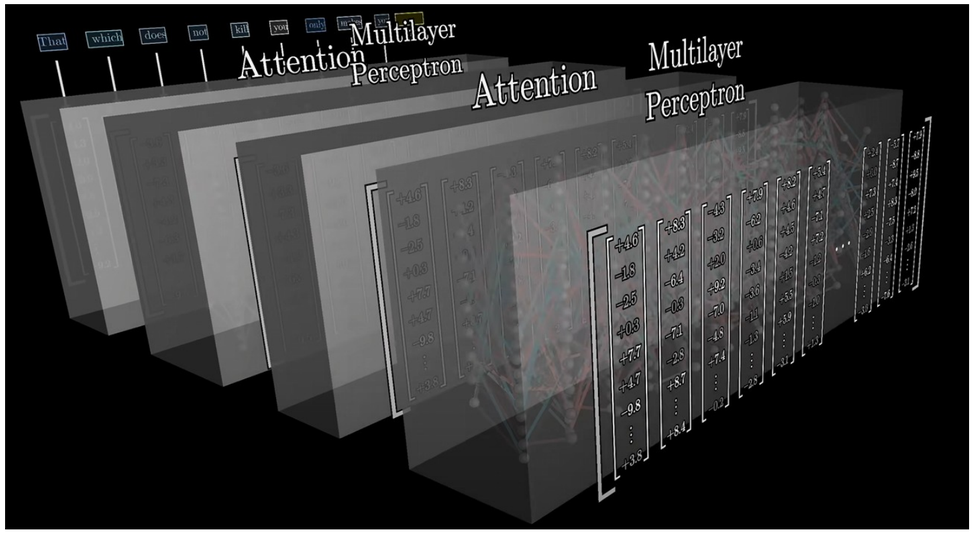
Forward Pass: Token-by-Token Prediction

- Input: Token sequence
- Model Output: Predicted probabilities for the next token
- Loss computed using Negative Log-Likelihood (NLL)

Backward Pass: Gradient Computation

- Gradients are calculated layer-by-layer using chain rule:
- Starting from the loss at the output layer and propagating back through the model.
- Attention layers contribute gradients for both:
  - Key, Query, Value matrices.
  - Multi-Headed Attention weights.





Loss Functions for AR Modeling

Training autoregressive models requires a loss function that measures how well the model predicts the next token in a sequence.
The choice of loss function directly impacts the model's ability to generalize, handle uncertainty, and produce coherent outputs.

While **Cross-Entropy Loss** is the standard choice, enhancements like **Label Smoothing** are often used to address challenges like overconfidence and overfitting.

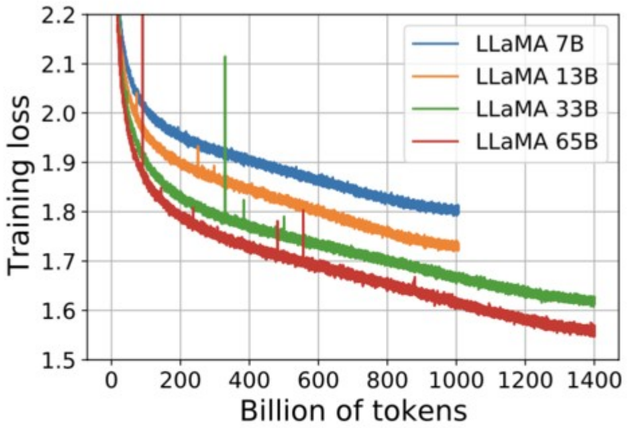


Figure 1: **Training loss over train tokens for the 7B, 13B, 33B, and 65 models.** LLaMA-33B and LLaMA-65B were trained on 1.4T tokens. The smaller models were trained on 1.0T tokens. All models are trained with a batch size of 4M tokens.

Cross-Entropy Loss

- **Purpose**: Measures the difference between predicted and true token distributions.
- **Key Benefit**: Maximizes likelihood of correct token predictions.
- **Challenge**: Can lead to overconfident predictions.

$$\mathcal{L}_{CE} = -\sum_{t=1}^{T} \log P(x_t | x_{<t})$$

Label Smoothing

- **Purpose**: Reduces overconfidence by distributing some probability mass to incorrect tokens.
- **Key Benefit**: Improves generalization and reduces overfitting.

$$\mathcal{L}_{LS} = -\sum_{t=1}^{T} \sum_{i} q_i \log P(x_t = i | x_{<t})$$

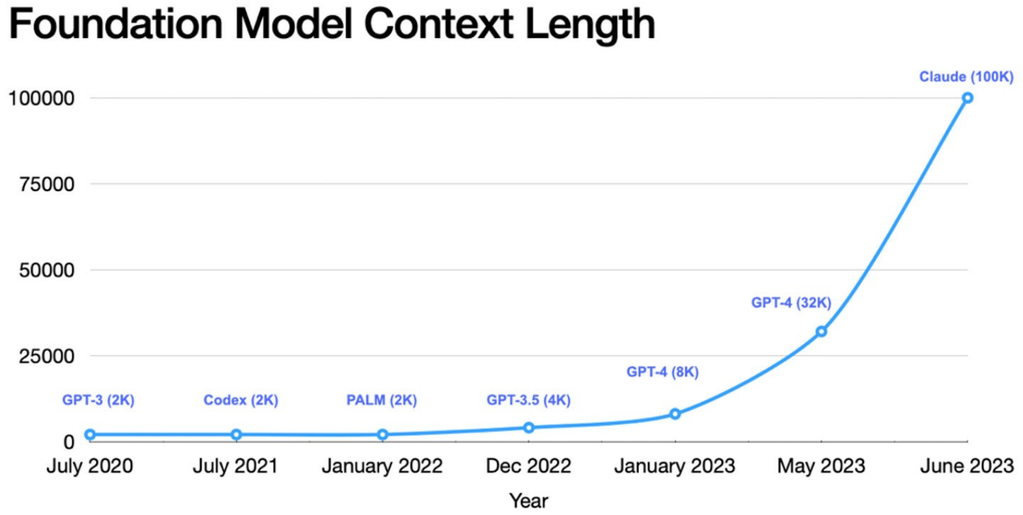
q_i: Smoothed label distribution (e.g., 0.9 for correct token, 0.1 spread across others).

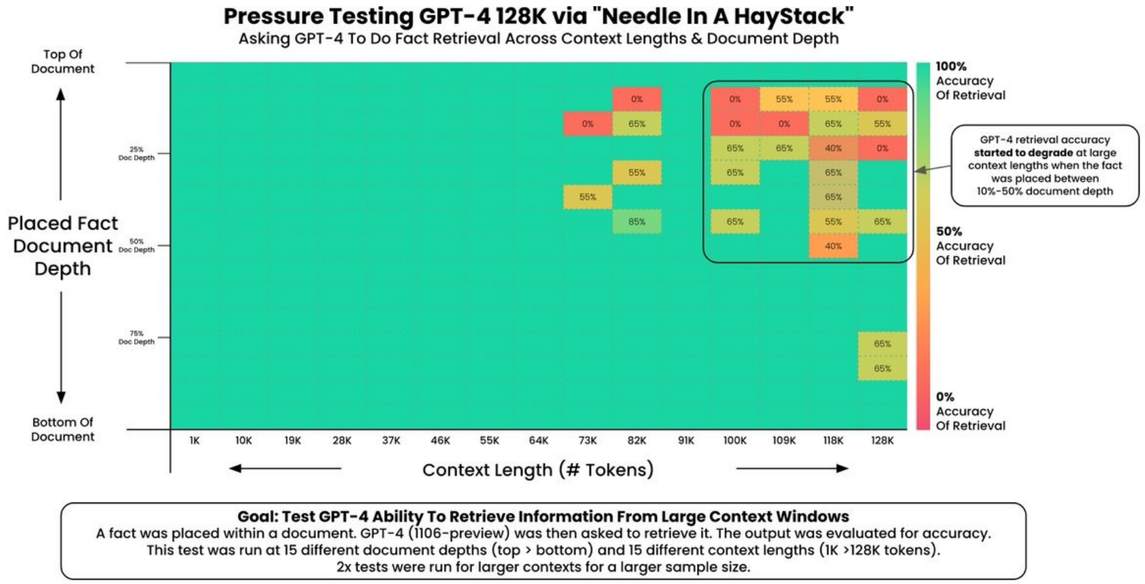
Input Context Considerations

Sequence Length Considerations – Accuracy

Impact of Sequence Length on Accuracy

- Longer Sequences:
  - Capture more context, improving predictions for tasks requiring long-term dependencies (e.g., summarization or story generation).
  - Higher accuracy for generating coherent and contextually relevant outputs.
  - Challenge: Increased computational complexity and risk of vanishing gradients.
- Shorter Sequences:
  - Faster training and inference due to reduced computational demand.
  - Limited context window leads to poorer predictions for context-heavy tasks.





Sequence Length Considerations - Memory

Longer Sequences:

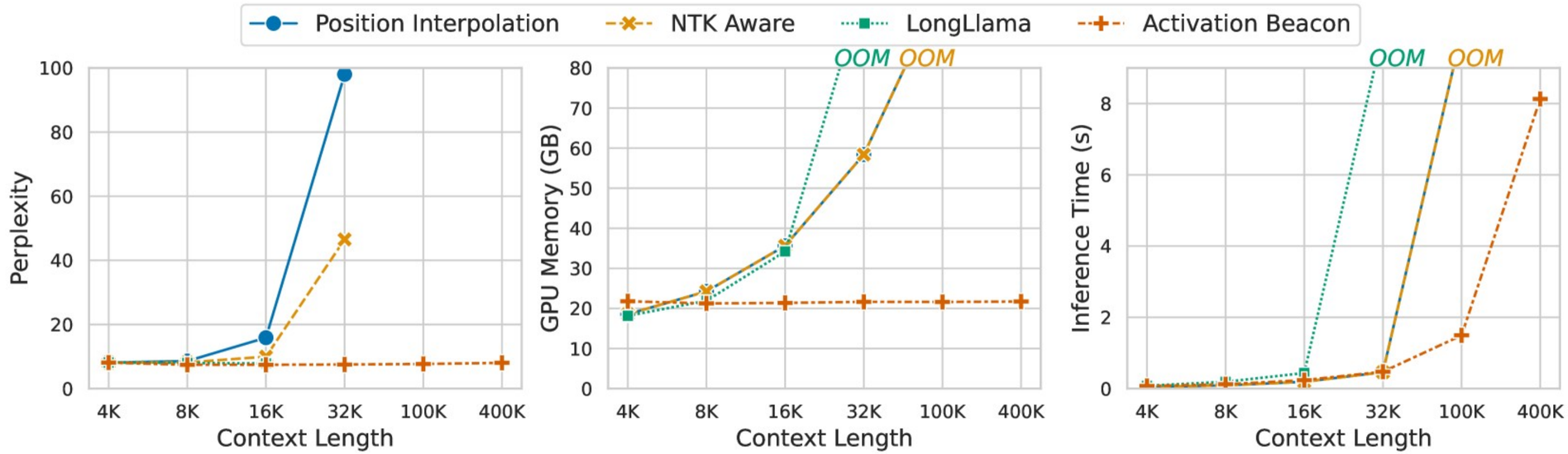
- Quadratic Growth in Attention: Memory usage scales as O(*n*^2), where *n* is the sequence length.
- Increases storage requirements for activations, attention weights, and gradients during training.

Shorter Sequences:

- Reduce memory consumption, enabling faster training and inference.
- May sacrifice context, impacting accuracy for tasks requiring long-term dependencies.

Memory Bottlenecks

- Self-Attention Mechanism: Stores attention matrices for all tokens, consuming significant memory for long sequences.
- Batch Size Trade-Off: Longer sequences force smaller batch sizes to fit within hardware limits.
- GPU/TPU Constraints: Limited VRAM in GPUs can bottleneck training for long sequences.

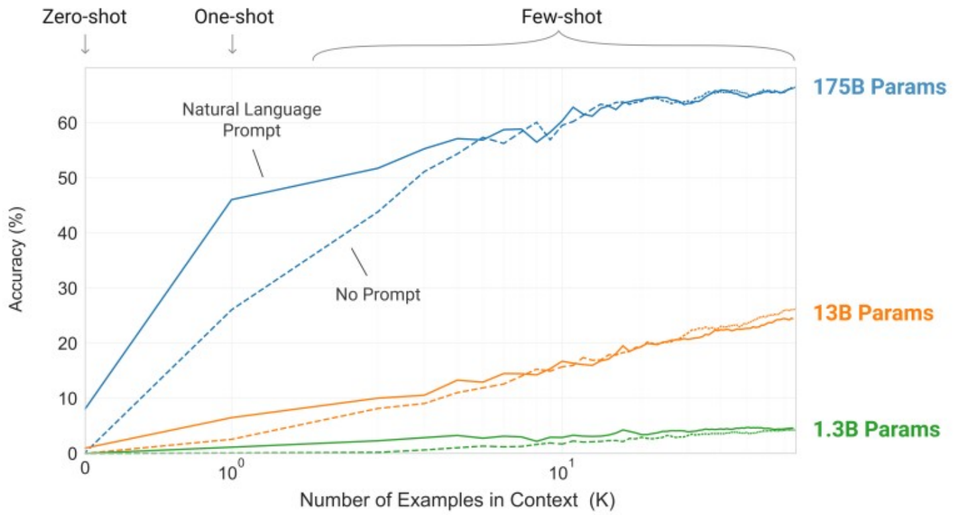


In-Context Learning (a look ahead)

In-Context Learning (ICL) allows language models to learn and perform tasks simply by being provided examples in their input, without any additional fine-tuning. Instead of updating model weights, the model uses its existing knowledge to infer patterns and generate appropriate outputs.

GPT-3 demonstrated remarkable generalization across tasks with zero-shot, one-shot, and few-shot learning using a single model.

- In ICL, all examples and task instructions must fit into the context window.
- A larger window allows the model to incorporate more task-specific information or handle more complex prompts.
- However, longer windows also increase computational costs and memory requirements, highlighting the trade-off between accuracy and efficiency.



Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:    ← task description
cheese =>                       ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:    ← task description
sea otter => loutre de mer      ← example
cheese =>                       ← prompt
```

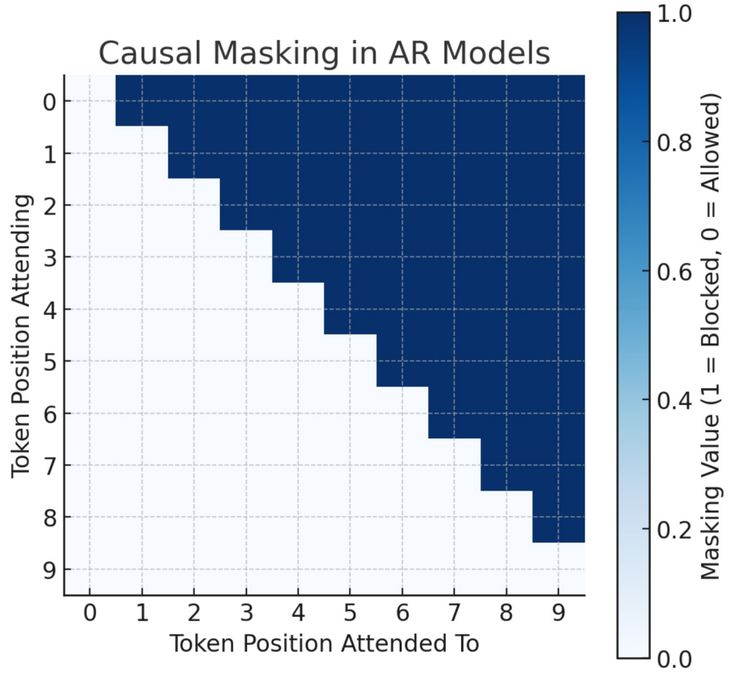
Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:    ← task description
sea otter => loutre de mer      ← examples
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>                       ← prompt
```

Challenging in Autoregressive Training

Causal Masking for AR Model Training

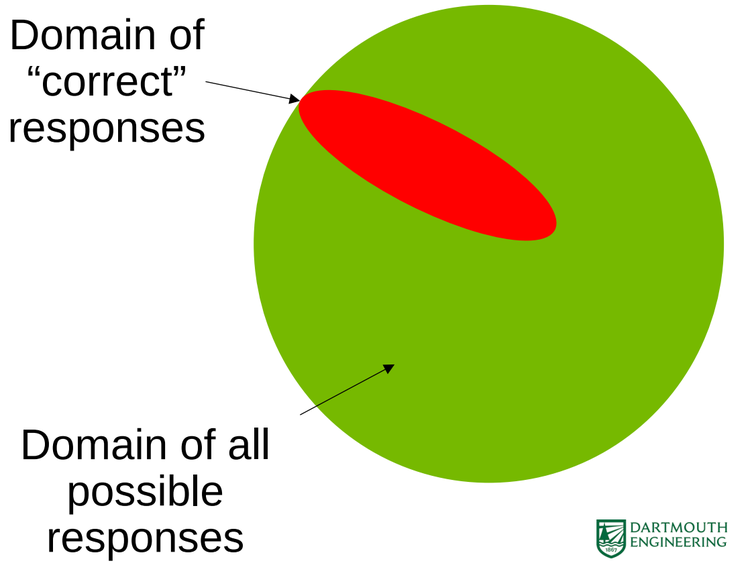


Causal Masking

- Ensures that each token can only attend to previous tokens in the sequence.
- Used during training to prevent "cheating" by looking at future tokens.
- Implemented via a triangular attention mask where:
  - Upper triangular elements are set to -infinity (ignored in softmax).
- Lower triangular elements allow attention to past tokens.

Exposure Bias

- Training Phase (Teacher Forcing): The model is fed the ground-truth previous tokens to predict the next token.
- Inference Phase: The model generates its own tokens iteratively, which may contain errors.
- Errors propagate because future predictions depend on past outputs.
- Leads to cascading errors, especially for long text generation.



Training / Inference Parallelism for AR Models – Beam Search

Instead of greedily selecting the single best next token at each step (greedy decoding), **Beam Search** keeps track of the top k (beam size) candidate sequences at each timestep. Each partial sequence is expanded by all possible next tokens, and only the top k overall are retained for the next iteration.

Motivation

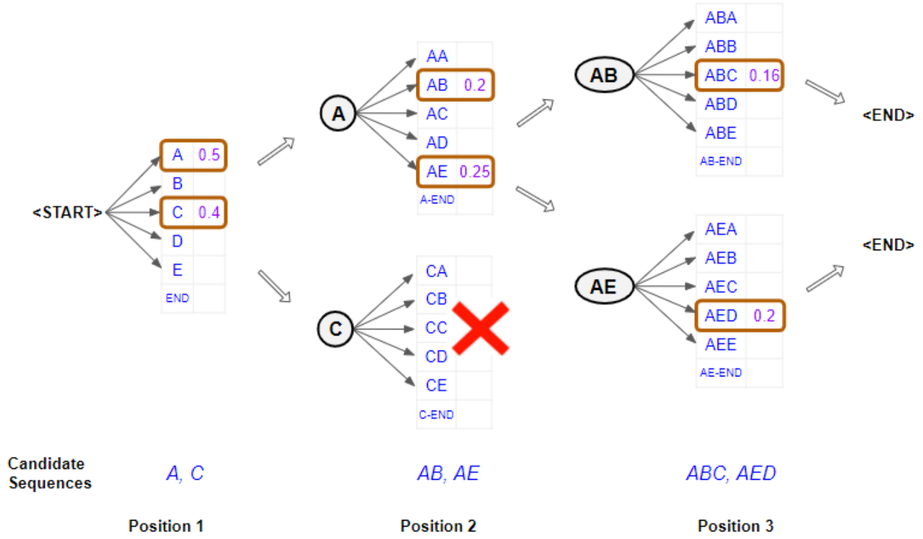
- **Balanced Exploration**: A middle ground between purely greedy decoding (fast but prone to errors) and exhaustive search (exponential time).
- **Quality Improvement**: Better chance of finding a high-probability sequence (e.g., for translation) compared to greedy search.

Advantages

- Higher-likelihood sequences compared to greedy sampling.
- Reduces the likelihood of local optima (short-sighted choices).

Limitations

- **Computationally Expensive**: Larger beam sizes increase computation.
- **Diversity Trade-off**: Can converge to similar or repetitive sequences if the model's probability mass is heavily skewed.
- Not guaranteed to find the true global optimum but often works well in practice.



Usage in LLMs

Commonly used in tasks demanding correctness or coherence (machine translation, summarization).
May reduce the creative “variety” found in purely stochastic sampling methods.

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
tokenizer = AutoTokenizer.from_pretrained("t5-base")
model = AutoModelForSeq2SeqLM.from_pretrained("t5-base")
encoder_input_str = "translate English to German: How old are
you?"
input_ids = tokenizer(encoder_input_str,
return_tensors="pt").input_ids
outputs = model.generate(
    input_ids,
    num_beams=10,
    num_return_sequences=1,
)
```

Training / Inference Parallelism for AR Models – Speculative Decoding

Speculative Decoding uses a smaller “draft” model to predict multiple tokens at once, then validates (or corrects) these tokens with the larger LLM.
If the large model confirms the predicted tokens, they are accepted in bulk; if not, the incorrect tokens are revised with a follow-up query to the large model.

Motivation

- Reduces the number of expensive calls to the large model by amortizing the “draft” process.
- Aimed at speeding up generation without sacrificing (too much) quality.

Mechanics

1. **Drafting**: The smaller model proposes a candidate sequence for the next few tokens.
2. **Verification**: The large model “checks” these tokens.
3. **Acceptance or Correction**: If tokens match the larger model’s probability distribution, they are accepted; otherwise, the mismatched portion is corrected by sampling from the large model directly.

Advantages

- Significantly faster generation in practice, especially for long sequences.
- Works well if the smaller model’s predictions are frequently accurate.

Challenges

- Designing the smaller model or “draft” strategy to align closely with the large model’s distribution.
- Managing mismatch: frequent corrections can reduce speed gains and potentially lead to generation artifacts.

Wrap Up

Autoregressive Training Transformers LLMs

- Today we introduced autoregressive modeling, where the model predicts the next token given all previous tokens, and discussed how it differs from non-autoregressive models.
- We examined the training pipeline, covering tokenization, data preprocessing, and training objectives like next-token prediction using Negative Log-Likelihood (NLL).
- The challenges of autoregressive modeling were highlighted, including exposure bias, balancing long-term dependencies, and hardware constraints due to sequential computation.
- We discussed input context considerations, including the impact of sequence length on accuracy and memory usage, as well as trade-offs in model efficiency.
- The lecture introduced beam search as a decoding strategy, comparing it with greedy sampling, and covered speculative decoding, which accelerates inference using a smaller draft model

Thank you!