Muppet Models

Natural Language Processing

University of Maryland

Midterm Review

Adapted from Hao Hoang

Scenario: Flight-Booking Assistant

You are building an assistant that helps customers:

- book flights,
- change existing reservations,
- cancel bookings.

To make the model perform well, you must use prompting strategies to correctly classify user intent and produce reliable outputs.

Subquestions:

A. Why is **prompt engineering** crucial?

B. Why does **zero-shot prompting** work?

C. What is **few-shot prompting**, and why is it beneficial?

D. What is **Chain-of-Thought prompting**, and how does it help reasoning?

A. Prompt Engineering

Subquestions:

A. Why is prompt engineering crucial?

B. Why does zero-shot prompting work?

C. What is few-shot prompting, and why is it beneficial?

D. What is Chain-of-Thought prompting, and how does it help reasoning?

A. Prompt Engineering

Subquestions:

A. Why is prompt engineering crucial?

B. Why does zero-shot prompting work?

C. What is few-shot prompting, and why is it beneficial?

D. What is Chain-of-Thought prompting, and how does it help reasoning?

Answer (A): Prompt engineering involves crafting clear, structured instructions that guide the model toward the desired behavior. Well-designed prompts reduce ambiguity, increase task adherence, and improve output quality—especially in a flight-booking assistant, where the model must reliably identify whether the user wants to book, change, or cancel a flight. Effective prompt engineering includes iterative refinement, testing alternative phrasings, and adding contextual constraints or formatting requirements so the model produces consistent, controllable responses.

B. Zero-Shot Prompting

Subquestions:

A. Why is prompt engineering crucial?

B. Why does zero-shot prompting work?

C. What is few-shot prompting, and why is it beneficial?

D. What is Chain-of-Thought prompting, and how does it help reasoning?

B. Zero-Shot Prompting

Subquestions:

A. Why is prompt engineering crucial?

B. Why does zero-shot prompting work?

C. What is few-shot prompting, and why is it beneficial?

D. What is Chain-of-Thought prompting, and how does it help reasoning?

Answer (B): Zero-shot learning allows LLMs to perform untrained tasks using general knowledge from pretraining. For example, if prompted with “Identify whether this customer wants to book, change, or cancel a flight,” the model can infer the correct intent without being explicitly trained on airline-domain data. This works because LLMs internalize broad semantic knowledge and communication patterns during pretraining, enabling them to generalize to new tasks based solely on a well-constructed instruction.

C. Few-Shot Prompting

Subquestions:

A. Why is prompt engineering crucial?

B. Why does zero-shot prompting work?

C. What is few-shot prompting, and why is it beneficial?

D. What is Chain-of-Thought prompting, and how does it help reasoning?

C. Few-Shot Prompting

Subquestions:

A. Why is prompt engineering crucial?

B. Why does zero-shot prompting work?

C. What is few-shot prompting, and why is it beneficial?

D. What is Chain-of-Thought prompting, and how does it help reasoning?

Answer (C): Few-shot learning enables LLMs to perform tasks with minimal examples, leveraging pretrained knowledge. Providing a few examples—such as customer messages paired with the correct intent label (book, change, cancel)—helps the model recognize the structure of the task, reduces ambiguity, and improves robustness to different phrasings. Few-shot prompting offers faster adaptation and lower cost than fine-tuning, making it ideal for domain-specific tasks like airline intent classification.

D. Chain-of-Thought Prompting

Subquestions:

A. Why is prompt engineering crucial?

B. Why does zero-shot prompting work?

C. What is few-shot prompting, and why is it beneficial?

D. What is Chain-of-Thought prompting, and how does it help reasoning?

D. Chain-of-Thought Prompting

Subquestions:

A. Why is prompt engineering crucial?

B. Why does zero-shot prompting work?

C. What is few-shot prompting, and why is it beneficial?

D. What is Chain-of-Thought prompting, and how does it help reasoning?

Answer (D): Chain-of-Thought (CoT) prompting guides LLMs to solve problems step-by-step, mimicking human reasoning. For example, if a customer writes: “I’m flying to Denver next week, but I need to switch to an earlier flight,” CoT helps the model break down the request—identifying relevant clauses, determining intent, and reasoning about the implied modification. This improves accuracy on multi-step queries, transparency of reasoning, and interpretability in domain tasks requiring careful analysis.

Scenario: Scaling a Math-Help Muppet Model

You work at a startup building a Muppet Model (LLM) that helps students solve math problems. The model works well, but you are struggling with:

- computational cost,
- latency under load,
- memory limits for fine-tuning.

You meet with your boss to discuss options.

Subquestions:

A. She suggests using **Mixture of Experts (MoE)** — how might that help?

B. She suggests **distillation** — how would you implement it?

C. You need fine-tuning but cannot update all parameters — **should you use LoRA or QLoRA, and why?**

D. She worries that LoRA relies on low-rank structure — **what does it mean if the fine-tuning update does not actually have low-rank eigenstructure, and how would you test this?**

A. Why might MoE help?

Subquestion A:

Your boss suggests using Mixture of Experts (MoE). How might that help scalability?

A. Why might MoE help?

Subquestion A:

> *Your boss suggests using Mixture of Experts (MoE). How might that help scalability?*

Answer (A): MoE enhances scalability by using a gating network to activate only a small subset of expert sub-networks for each input. Instead of running the entire model for every math question, the gating function might activate only 10% of the total parameters. This reduces computational load and allows you to deploy a much larger model—one with many specialized experts (e.g., algebra expert, geometry expert, word-problem expert)—without increasing per-query cost. This makes it possible to scale to billion-parameter models while maintaining efficiency and strong performance.

B. How would you implement distillation?

Subquestion B:

She also suggests distillation. How would you implement it for the math muppet model?

B. How would you implement distillation?

Subquestion B:

> *She also suggests distillation. How would you implement it for the math muppet model?*

Answer (B): Model distillation trains a smaller “student” model to imitate the output behavior of a larger “teacher” model. You would run the teacher model on a large set of math problems—possibly synthetic ones—and store the soft probability distributions over answers. The student is then trained to match these soft targets rather than hard labels, allowing it to learn richer decision boundaries. The result is a lightweight model suitable for deployment on laptops, tablets, or school Chromebooks, retaining near-teacher performance while using far less memory and compute.

C. Should you use LoRA or QLoRA?

Subquestion C:

You need fine-tuning but can't update all parameters. Should you use LoRA or QLoRA?

C. Should you use LoRA or QLoRA?

Subquestion C:

> *You need fine-tuning but can't update all parameters. Should you use LoRA or QLoRA?*

Answer (C): LoRA adds low-rank adaptation matrices to model layers, enabling efficient fine-tuning with minimal memory overhead. QLoRA extends LoRA by quantizing the base model (e.g., to 4-bit precision), which drastically reduces memory usage while preserving accuracy through carefully designed quantization and dequantization mechanisms. If you are memory-constrained—for example, trying to fine-tune a 70B muppet model on a single GPU—QLoRA is the better choice. If you have more GPU memory and prefer simpler debugging and fewer quantization artifacts, vanilla LoRA is sufficient.

D. What if the fine-tuning update isn't low-rank?

Subquestion D:

Your boss notes (as a math PhD) that LoRA assumes the fine-tuning update has low-rank structure. What does it mean if the update does not have low-rank eigenstructure, and how would you test this?

D. What if the fine-tuning update isn't low-rank?

Subquestion D:

> *Your boss notes (as a math PhD) that LoRA assumes the fine-tuning update has low-rank structure. What does it mean if the update does not have low-rank eigenstructure, and how would you test this?*

Answer (D): LoRA relies on the assumption that the necessary parameter update matrix during fine-tuning has only a few significant eigenvectors—that is, the update lies mostly within a low-dimensional subspace. If the update truly has many large eigenvalues, then the low-rank LoRA update cannot fully capture the transformation needed for your math-domain tasks. You would test this by:

- computing the empirical update matrix from a baseline fine-tuning step,
- performing an eigenvalue or singular value decomposition (SVD),
- inspecting how rapidly singular values decay.

If the spectrum decays slowly, the update is not low-rank, meaning LoRA may be insufficient; you might then increase LoRA rank or switch to a different modular architecture.

Scenario

You're talking with a frustrated student who says:

"Why do we learn all this old math in CMSC 470? None of it matters for modern Muppet Models (LLMs)."

You're asked to explain how each of these topics is fundamental to how modern attention-based LLMs actually work.

Topics to Explain

Why each of the following concepts matters for modern LLMs:

- Softmax
- Cross-Entropy Loss
- KL Divergence
- ReLU Derivative
- Chain Rule
- Multi-Head Attention
- Hyperparameters

Softmax

Softmax

Softmax is central to modern attention mechanisms. It transforms attention logits into a valid probability distribution:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

In an LLM, the raw similarity scores (query–key dot products) must be normalized so the model can decide how much to attend to each token. Softmax ensures the model focuses on contextually relevant words by amplifying higher scores and suppressing lower ones, making attention stable and interpretable.

Cross-Entropy Loss

Cross-Entropy Loss

Cross-entropy measures divergence between predicted token probabilities and the true next-token distribution:

$$L = -\sum_i y_i \log(\hat{y}_i)$$

It penalizes incorrect next-token predictions. In modern Muppet Models, next-token training is the foundation of everything—reasoning, generation, coding, math tutoring. Cross-entropy ensures the model assigns higher probability to correct continuations, directly optimizing predictive accuracy.

KL Divergence

KL Divergence

KL divergence measures how one probability distribution differs from another:

$$D_{\mathrm{KL}}(P \,\|\, Q) = \sum_{x} P(x) \log \frac{P(x)}{Q(x)}$$

Modern LLM uses include:

- Aligning model output distributions during fine-tuning
- RLHF reward modeling
- Distillation from teacher to student models

KL is the mathematical backbone of “matching” behaviors between models or datasets.

ReLU Derivative

ReLU Derivative

The ReLU activation:

$$f(x) = \max(0, x)$$

has derivative

$$f'(x) = \begin{cases} 1 & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Its sparse, piecewise-linear derivative prevents vanishing gradients and keeps computation efficient. Even though transformers use GELU, ReLU is foundational for understanding why nonlinearities are required—and why deep LLMs can actually train.

Chain Rule

Chain Rule

The chain rule for composite functions:

$$\frac{d}{dx} f(g(x)) = f'(g(x)) \cdot g'(x)$$

is the basic machinery behind backpropagation. Every weight update in an LLM—from attention to embeddings to output projection—depends on repeatedly applying the chain rule across dozens or hundreds of layers. Without the chain rule, there is no gradient descent, and no modern LLMs.

Multi-Head Attention

Multi-Head Attention

Multi-head attention splits queries, keys, and values into multiple subspaces. Each head learns to focus on different aspects of the input: syntax, semantics, long-distance structure, or task-specific reasoning. This diversity allows LLMs to capture complex patterns simultaneously, enabling coherent math explanations, reasoning traces, or multi-step problem solving.

Hyperparameters

Hyperparameters

Hyperparameters are preset training controls—learning rate, batch size, dropout, weight decay, number of attention heads. They determine convergence behavior, training stability, and final performance. Modern muppet models are massively sensitive to these settings, and poor hyperparameter tuning can derail training regardless of dataset quality or architecture.

Scenario

You've joined a company whose ML stack is stuck in the 2000s.

Your boss is technical but hasn't kept up with modern AI.

In your first onboarding meeting, he asks you to walk him through "what all this new LLM stuff really is, and how it differs from what we used to do."

Topics to Explain

- Generative vs. Discriminative Models
- What are Sequence-to-Sequence Models?
- How Transformers Improve on Older Seq2Seq Approaches
- Next Sentence Prediction (and which models use it)
- Autoregressive vs. Masked Models
- What Defines a Muppet / Foundation / Frontier Model

Generative vs. Discriminative Models

Generative vs. Discriminative Models

Generative models (e.g., GPT) model the joint distribution of data, enabling them to generate new content such as text, images, or completions. In language tasks, this means producing coherent sentences token by token.
Discriminative models (e.g., BERT used for classification) model conditional probabilities to distinguish classes—like sentiment categories or spam detection.
Generative models are ideal for creation and open-ended tasks; discriminative models excel at structured prediction and classification.

Sequence-to-Sequence Models

Sequence-to-Sequence Models

Sequence-to-sequence (Seq2Seq) architectures transform an input sequence into an output sequence—often of different length.
They use an **encoder** (to process input) and a **decoder** (to produce outputs), enabling:

- Machine translation (e.g., English $\rightarrow$ Spanish)
- Text summarization
- Chatbots and dialogue systems

This encoder–decoder paradigm is the predecessor of modern transformer-based translation and reasoning systems.

How Transformers Improve on Classic Seq2Seq Models

How Transformers Improve on Classic Seq2Seq Models

Transformers solved the major limitations of RNN-based Seq2Seq systems by introducing:

- **Parallel Processing:** Self-attention allows all tokens to be processed simultaneously instead of sequentially.
- **Long-Range Dependencies:** Attention easily captures relationships across distant parts of a sentence.
- **Positional Encodings:** Preserve sequence order without recurrence.

These innovations dramatically improved scalability, training speed, and performance for tasks like translation, summarization, and document modeling.

Next Sentence Prediction (NSP)

Next Sentence Prediction (NSP)

Next Sentence Prediction (NSP) is a pretraining task used in **models like BERT**. The model is trained to determine whether two sentences are consecutive (50% of the time) or randomly paired (50%).
NSP helps the model learn:

- Sentence-to-sentence coherence
- Document-level reasoning
- Dialogue and summarization context

This objective is essential for tasks requiring understanding relationships between sentences, not for generative models like GPT.

Autoregressive vs. Masked Models

Autoregressive vs. Masked Models

Autoregressive models (e.g., GPT):

- Predict the next token based only on prior tokens.
- Excel at generative tasks: text continuation, story writing, coding.

Masked models (e.g., BERT):

- Predict masked tokens using bidirectional context.
- Excel at understanding tasks: classification, sentiment, retrieval.

Their pretraining objectives directly shape strengths: generation (autoregressive) vs. comprehension (masked).

What Defines a Muppet / Foundation / Frontier Model

What Defines a Muppet / Foundation / Frontier Model

Modern “Muppet Models,” “Foundation Models,” or “Frontier Models” are:

- Large-scale AI systems trained on vast corpora.
- Typically transformers with billions of parameters.
- Capable of broad generalization: translation, summarization, reasoning, Q&A.
- Built through unsupervised or self-supervised pretraining on massive datasets.

They differ from traditional statistical language models (e.g., N-grams) by using:

- Attention architectures
- Contextual embeddings
- Long-distance dependency modeling
- Large-scale foundation training

These models form the backbone of modern AI applications and product ecosystems.

Scenario: Producing Diverse Translations

You're working on a machine translation (MT) system based on an autoregressive model. Your goal: produce diverse but high-quality translations for the same source sentence.
You need to decide how to configure generation to balance coherence and variety.

Techniques to Discuss

- Beam Search
- Top-k Sampling
- Top-p (Nucleus) Sampling
- Temperature

Beam Search

Beam Search

Beam search explores multiple candidate sequences at each generation step instead of selecting only the single most probable token (as in greedy decoding).

How it helps:

- Keeps top-k sequences ("beams"), balancing probability and diversity.
- Ensures coherent translations while considering alternative plausible word choices.

Usage: Use a moderate beam width (e.g., $k = 5$) to explore different translation paths without generating incoherent outputs.

Top-k Sampling

Top-k Sampling

Top-k sampling randomly selects the next token from the k most probable options.

How it helps:

- Introduces randomness while limiting to high-probability words.
- Produces diverse outputs while avoiding extremely unlikely or incoherent tokens.

Usage: Set k (e.g., 20) to control the trade-off between diversity and quality. Higher $k \rightarrow$ more varied translations.

Top-p (Nucleus) Sampling

Top-p (Nucleus) Sampling

Top-p (nucleus) sampling chooses the next token from the smallest set whose cumulative probability exceeds threshold p (e.g., 0.95).

How it helps:

- Dynamically adapts to the context, selecting more options when the model is uncertain.
- Produces varied but contextually coherent translations.

Usage: Use $p \in [0.9, 0.95]$ to allow flexibility in MT while avoiding very low-probability tokens that could break fluency.

Temperature

Temperature

Temperature adjusts the “sharpness” of the probability distribution over tokens:

- Low temperature (~ 0.3) $\rightarrow$ favors high-probability tokens $\rightarrow$ conservative, predictable translations.
- High temperature (~ 1.5) $\rightarrow$ flattens distribution $\rightarrow$ more diverse outputs.

How it helps: Increases variability in translation while keeping the model guided by learned probabilities.
Usage: Set moderate temperature (e.g., 0.8) to balance coherence and diversity for MT outputs.

Scenario: Handling Client-Specific Inputs

You're provisioning a bespoke Muppet Model for a client company. The client is concerned that their specialized inputs (e.g., agricultural terms) aren't processed correctly.
Your goal: explain how the model handles inputs while keeping them confident in the system.

Client Questions on Input Handling

- How do you decide how to process the Unicode stream, and can it be customized for our data?
- Token embedding initialization
- How do you initialize token embeddings?
- We're a tractor company; many agriculture words aren't in the vocabulary. Is this a problem?
- What are positional embeddings, and how do they relate to segment embeddings in older BERT-based models?
- What is the context window, and how do I know if increasing it is worth the cost?

Processing the Unicode Stream

Question: How do you decide how to process the Unicode stream, and can it be customized for our data?

Processing the Unicode Stream

Question: How do you decide how to process the Unicode stream, and can it be customized for our data?

Answer:

- The raw text (Unicode) is normalized and mapped to tokens using a tokenizer (splitting into words, subwords, or characters).
- Tokenization ensures the model sees a consistent numerical representation.
- For specialized domains like agriculture, we can extend the tokenizer with custom vocab or pre-tokenize common domain-specific terms.
- This allows the model to handle client-specific language without retraining from scratch.

Token Embedding Initialization

Question: How do you initialize token embeddings?

Token Embedding Initialization

Question: How do you initialize token embeddings?
Answer:

- Embeddings are dense vectors representing tokens in a continuous space.
- They are typically initialized randomly or with pretrained embeddings (e.g., GloVe, FastText).
- During fine-tuning, embeddings are adjusted to capture semantic and syntactic relationships relevant to your domain.
- For your tractor/agriculture vocabulary, embeddings for new tokens will evolve based on context during training, ensuring accurate representation.

Handling Out-of-Vocabulary Words

Question: We're a tractor company; many agriculture words aren't in the vocabulary. Is this a problem?

Handling Out-of-Vocabulary Words

Question: We're a tractor company; many agriculture words aren't in the vocabulary. Is this a problem?

Answer:

- LLMs use subword tokenization (e.g., BPE) to split unknown words into smaller known units.
- Example: "harvester" $\rightarrow$ "harv" + "ester".
- This ensures even unseen or domain-specific words are understood, though semantic precision improves with fine-tuning on your text.
- Optionally, add common agriculture terms to the vocabulary to improve efficiency and interpretability.

Positional Embeddings

Question: What are positional embeddings, and how do they relate to segment embeddings in older BERT-based models?

Positional Embeddings

Question: What are positional embeddings, and how do they relate to segment embeddings in older BERT-based models?

Answer:

- Transformers are inherently order-agnostic; positional embeddings encode token position information.
- Sinusoidal or learned vectors are added to token embeddings so the model knows sequence order (important for sentences like "tractor pulls plow" vs. "plow pulls tractor").
- Segment embeddings (from BERT) distinguished sentence pairs; positional embeddings complement token embeddings to give both order and segment context.
- In practice, this ensures correct understanding of token relationships in your domain-specific text.

Context Window

Question: What is the context window, and how do I know if increasing it is worth the cost?

Context Window

Question: What is the context window, and how do I know if increasing it is worth the cost? **Answer:**

- The context window is the maximum number of tokens the model can process in a single input.
- Larger windows allow the model to consider more context (e.g., entire instructions or manuals), improving coherence and reasoning.
- Cost scales roughly linearly with window size; doubling the window can double compute requirements.
- Decision depends on task needs: for short queries, current window suffices; for long documents or multi-step reasoning, increasing the window can improve output quality.

Scenario: Common Sense QA System

Your team is building a system to answer tricky "common sense" questions. The goal is to improve your model's performance in a competition setting.
Consider: How do you handle Cranfield-style evaluation, integrate external knowledge, and improve ranking metrics?

Subquestions for QA System Improvement

- What are Cranfield Paradigm questions and how does that influence expectations?
- Evaluation metrics beyond exact match
- Using RAG for QA
- Knowledge graphs: benefits and usage

Cranfield Paradigm in QA

Question: What does it mean that questions are Cranfield Paradigm, and how does it change expectations?

Cranfield Paradigm in QA

Question: What does it mean that questions are Cranfield Paradigm, and how does it change expectations?

Answer:

- Cranfield Paradigm comes from information retrieval evaluation: a fixed document collection, a set of queries, and relevance judgments.
- Key point: These are information-seeking questions — the asker does not already know the answer.
- Implications for QA:
  - The model cannot assume prior knowledge; it must retrieve or reason based on evidence.
  - Must detect and correct false presuppositions in the question.
  - May need to resolve ambiguity by clarifying or disambiguating the query context.
  - Evaluation is typically based on relevance or correctness of the retrieved/generated answer.
- Your system design should focus on combining retrieval, reasoning, and clear presentation of supported answers.

Evaluation Metrics for QA

Question: The organizers of the competition suggest exact match accuracy. What alternatives could you propose?

Evaluation Metrics for QA

Question: The organizers of the competition suggest exact match accuracy. What alternatives could you propose?
Answer:

- **F1-score:** Considers partial overlap between predicted and gold answers; helpful for multi-word answers.
- **Mean Reciprocal Rank (MRR):** Useful if multiple candidate answers are ranked.
- **BLEU** / **ROUGE:** Measures n-gram overlap; more flexible for free-form responses.
- Rationale: Exact match is strict and may penalize semantically correct but phrased differently answers; ranking-based or partial-match metrics better reflect model utility.
- Ranking: Item Response Theory would help identify problematic questions.

Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG)

Question: What is RAG and would it help here? How would you use it?

Retrieval-Augmented Generation (RAG)

Question: What is RAG and would it help here? How would you use it?
Answer:

- RAG combines retrieval with generation: relevant documents are retrieved and provided as context for an LLM to generate answers.
- Helpful for common-sense QA when the model's internal knowledge is insufficient or outdated.
- Implementation:
  1. Index a knowledge corpus relevant to the questions.
  2. Use a retriever (dense or sparse embeddings) to fetch top-k documents per query.
  3. Feed retrieved documents along with the query to the generative model.
- This improves answer accuracy and provides explainable context.

Knowledge Graphs for QA

Knowledge Graphs for QA

Question: Would a knowledge graph help? What is it, and how would you use it?

Knowledge Graphs for QA

Question: Would a knowledge graph help? What is it, and how would you use it?

Answer:

- A knowledge graph (KG) represents entities and their relationships in a structured graph.
- Helps QA by enabling reasoning over explicit relationships, e.g., "tractor $\rightarrow$ used for $\rightarrow$ plowing."
- Use cases in QA:
  - Augment retrieval by finding paths connecting query entities.
  - Support logical reasoning or validation of candidate answers.
  - Combine with LLMs: feed KG facts as structured context or precompute embeddings for retrieval.
- For Cranfield-style QA, KG can improve both relevance and consistency of answers.