reading guide week 2.docx

1. Basic Definitions in Layman’s Terms

ANN (Artificial Neural Network)

Think of an ANN as a simplified digital brain that learns patterns from data. It consists of layers of artificial neurons that take input, process it through mathematical operations, and produce an output, improving over time by adjusting its internal settings.

LSTM (Long Short-Term Memory)

LSTMs are a special type of neural network that remembers past data for long periods. Imagine it as a notebook where you can write important notes while reading a book, so you don’t forget previous chapters while reading the next.

RNN (Recurrent Neural Network)

RNNs are designed to process sequential data by remembering past inputs. They are like a conveyor belt, where each step influences the next. However, they struggle with long-term memory, which is why LSTMs were developed.

Tokenization

Tokenization splits text into smaller units (tokens) like words, subwords, or characters, making it easier for a computer to process language. Think of it like chopping a paragraph into pieces, so an AI can understand each word separately.

Embeddings

Embeddings convert words into numbers in a way that captures meaning. Imagine a map, where words with similar meanings (e.g., "dog" and "puppy") are placed closer together, making it easier for AI to understand relationships.

Backpropagation

Backpropagation is how AI learns from mistakes. It compares the predicted answer to the correct one, calculates the error, and adjusts the internal connections to improve accuracy—like practicing a basketball shot and adjusting your aim after every miss.

Bayes’ Theorem

Bayes’ theorem helps AI update its predictions based on new evidence. If you think there's a 30% chance of rain, but then you see dark clouds, you update your belief to a higher probability.

Prior Probability, Posterior Probability, and Likelihood

Prior Probability: The initial assumption before seeing new data (e.g., "I believe there’s a 20% chance my favorite team will win").

Likelihood: How well the new evidence supports the assumption (e.g., "The team’s star player is injured").

Posterior Probability: The updated belief after incorporating new evidence (e.g., "Now, I think they only have a 10% chance of winning").

Activation Function

An activation function decides whether a neuron should activate. Think of it as a filter, allowing only important information to pass through. Examples include ReLU (most common) and Softmax (used for probabilities).

2. What Constitutes the Input and Output Layers of an LLM?

Input Layer: Takes in text, converts it into tokens, then transforms those into word embeddings (numerical vectors).

Output Layer: Predicts the next token based on probabilities, which are converted back into words.

3. What is "Attention" in Transformers?

Attention is how AI focuses on important words in a sentence. Imagine reading a book—you don’t remember every word, but you focus on key sentences that help you understand the story. Attention helps LLMs decide which words matter most in a sentence.

4. What are the Two Main Components of a Transformer?

Encoder (like a reader) → Understands the input by looking at all words simultaneously.

Decoder (like a writer) → Uses that understanding to generate the next words in a response.

5. Core Building Blocks of the Encoding Part of the Transformer

Self-Attention → Helps the model weigh the importance of different words.

Feedforward Layers → Processes the data further, like applying grammar rules.

Positional Encoding → Helps AI understand word order (since transformers process all words at once).

6. Core Building Blocks of the Decoding Part of the Transformer

Masked Self-Attention → Looks at previous words only (to predict the next word in sequence).

Cross-Attention → Connects with the encoder's understanding of the input.

Softmax Layer → Converts raw scores into probabilities for choosing the next word.

7. Two Factors That Determine the Quality and Diversity of LLM Output

Foundation model: LLM Evaluation Metrics

8. Main Steps in the Training Process of an LLM

Pretraining → AI reads massive amounts of text and learns general language patterns.

Fine-tuning → Adjusts the AI for specific tasks (e.g., legal documents, medical advice).

Evaluation → Tests accuracy using benchmarks.

Deployment → The model is made available via an API.

9. How Does Reinforcement Learning from Human Feedback (RLHF) Work?

Human reviewers rank AI responses.

A reward model is trained based on human preferences.

AI fine-tunes itself using this ranking to improve alignment with human expectations.

10. LLM Evaluation Frameworks in Layman's Terms

GLUE & SuperGLUE → Tests general language understanding (SuperGLUE is harder).

MMLU → Measures AI's knowledge across multiple subjects (math, law, science).

HellaSwag → Evaluates common sense reasoning by predicting the most logical next sentence.

TruthfulQA → Tests whether AI can avoid false or misleading statements.

AI2 Reasoning Challenges (ARC) → Focuses on scientific and logical reasoning.

11. Three Ways to Customize a Pre-Trained LLM

1. Extending Non-Parametric Knowledge

Instead of relying only on what it learned in training, the AI fetches real-time knowledge from external sources (e.g., Wikipedia, company databases). This helps keep answers accurate and up-to-date.

2. Few-Shot Learning

The AI learns a new task on the fly without retraining. If you give it a few examples in the prompt (e.g., "Translate 'Hello' to French" → "Translate 'Goodbye' to French"), it quickly adapts without requiring retraining.

3. Fine-Tuning

The model is retrained on specialized data to perform better in specific domains (e.g., a law firm fine-tuning AI on legal documents to improve legal advice accuracy).

Based on the tone and style of your reflection document, here are your answers as a fun AI expert, maintaining an analytical, structured, and slightly reflective style:

1. What does inverse scaling mean when training foundation models?

Inverse scaling refers to a phenomenon where larger models perform worse on certain tasks compared to smaller models. Typically, we expect that increasing model size improves performance, but in some cases—especially tasks requiring memorization or handling strong priors—larger models might generalize poorly or amplify biases. This issue was highlighted by research efforts like the Inverse Scaling Prize, where researchers found that as models grew, they sometimes failed on specific benchmarks.

Lesson: More parameters don't always mean better performance. Bigger models can sometimes amplify errors, biases, or unintended behaviors.

2. How do training data impact the performance of foundation models?

Training data is the foundation of any model's performance. The size, diversity, and quality of training data determine what a model learns, how it generalizes, and whether it hallucinates or provides accurate outputs. For example, a model trained on high-quality, human-verified text will be more reliable than one trained on noisy or biased internet data. The challenge is that large-scale models require vast amounts of data, leading to concerns about data scarcity and AI models inadvertently learning from AI-generated content.

Lesson: Garbage in, garbage out. No matter how powerful a model is, its output is only as good as its training data.

3. The book provided a few domain-specific foundation models. Select one and describe it in your own language.

One notable domain-specific model is Google's Flamingo, which specializes in multimodal understanding—combining images and text. Think of Flamingo as a model that can "see" and "read" at the same time. If you show it a picture of a cat wearing sunglasses, it can describe the image and even answer questions like, "Why is the cat wearing sunglasses?" This makes it useful for applications like visual question answering, caption generation, and content moderation.

Lesson: Specialized models are often more effective than general-purpose models for domain-specific tasks.

4. How would you describe a foundation model's size? Select a foundation model and use it as an example.

A foundation model's size is typically described in terms of:

Number of parameters: The more parameters, the more complex patterns it can learn.

Number of training tokens: How much data it has been exposed to.

Compute required for training (FLOPs): The processing power needed.

Example: Llama 3-70B has 70 billion parameters, a context length of 128K tokens, and was trained on 15 trillion tokens. While larger than Llama 2, it was optimized for efficiency, making it easier to deploy despite its scale.

Lesson: Bigger isn't always better, but it often means more capability—at the cost of higher computational demand.

5. Describe the “Chinchilla scaling law” when training compute-optimal large language models. Do you think it is still valid?

The Chinchilla scaling law suggests that to train an optimal model given a fixed compute budget, the number of training tokens should be 20 times the number of parameters. This means that instead of endlessly increasing model size, researchers should also focus on increasing the amount of high-quality training data.

Is it still valid? Mostly, yes. However, recent advancements in sparse models, mixture-of-experts (MoE), and retrieval-augmented generation (RAG) challenge this rule by introducing new ways to scale models efficiently without increasing data requirements proportionally.

Lesson: Scaling is not just about size—it’s about balance. Efficient training strategies matter.

6. What are the differences between pre-training and post-training of an LLM?

Pre-training: The model learns general knowledge by predicting the next token on large-scale, unsupervised internet data.

Post-training: The model is refined through techniques like Supervised Finetuning (SFT) and Reinforcement Learning from Human Feedback (RLHF) to align its responses with human preferences.

Lesson: Pre-training builds raw intelligence, while post-training refines it for user interaction.

7. Describe Figure 2-10, the overall training flow with pre-training, SFT, and RLHF, in your own language.

Imagine building a robot assistant:

Pre-training: You teach the robot by making it read everything ever written.

Supervised Finetuning (SFT): You give it specific examples of how it should respond.

RLHF: You show it two different answers and reward the better one, refining its ability to give good responses.

It’s like first teaching a student general knowledge, then giving them structured lessons, and finally training them through real-world exams.

Lesson: AI models need layers of refinement to be useful and safe.

8. How does supervised fine-tuning work?

Supervised finetuning (SFT) involves training a model using (prompt, response) pairs where humans provide high-quality examples of desired outputs. This process helps shift the model from text completion mode (predicting the next word) to conversation mode (answering coherently).

Lesson: If pre-training is “reading books,” supervised finetuning is “learning from teachers.”

9. How does preference fine-tuning work?

Preference finetuning involves:

Training a reward model to rank AI-generated responses.

Using reinforcement learning to make the AI favor responses that align with human preferences (RLHF).

Example: If you train an AI to answer moral dilemmas, preference finetuning ensures it avoids offensive or misleading responses.

Lesson: AI doesn’t inherently know what’s “good” or “bad.” Humans have to teach it.

10. What does temperature mean in LLM? How would you set the “temperature” of a specific LLM based on your application?

Temperature controls randomness.

Low temperature (0-0.3): More deterministic, good for factual answers.

Medium (0.5-0.7): Balanced creativity and coherence.

High (0.8-1.2): More randomness, useful for storytelling or idea generation.

Lesson: Adjust temperature based on whether you want “precise” or “creative” responses.

11. Discuss the differences and similarities between the top-k versus the top-p sampling strategy.

Top-k: Keeps only the k most likely tokens.

Top-p: Picks tokens until cumulative probability reaches p%.

Similarities: Both reduce randomness.

Differences: Top-k limits count, top-p limits probability.

Lesson: Top-k is like picking from a shortlist, top-p is like picking until enough options add up.

12. What is a stop condition for an LLM to generate text?

A stop condition is when the model knows to stop generating output. This can be:

Reaching a maximum token limit.

Detecting a stop word (e.g., "END").

Using semantic rules (e.g., ensuring JSON outputs are properly formatted).

13. What is “test time compute”? How would you select the best output under this strategy?

Test time compute refers to running multiple inferences and picking the best one. Selection methods include:

Picking the highest probability response.

Using a reward model to rank outputs.

Majority voting (useful for classification tasks).

14. What are different strategies for generating structured outputs from an LLM?

Prompting: Ask the model directly.

Post-processing: Clean up errors manually.

Constrained sampling: Ensure outputs follow strict rules.

Finetuning: Train the model to generate structured outputs reliably.

15. Why do we say “GenAI models are probabilistic?”

Because they don’t return the same answer every time. Instead, they sample from a probability distribution over possible outputs.

16. What is “inconsistency” in a GenAI model’s output? How may you address this phenomenon?

Definition: The model gives different responses to the same prompt.

Fix: Lower temperature, use seed values, or cache responses.

17. What is “Hallucination” in a GenAI model’s output? How may you address this phenomenon?

Definition: The model generates false or misleading information.

Fix: Improve training data, use retrieval-augmented generation (RAG), and verify responses.

Let me know if you want refinements! 🚀

Pure Reinforcement Learning (RL) is a training approach where an AI agent learns by interacting with an environment, receiving feedback in the form of rewards or penalties based on its actions, and adjusting its behavior to maximize cumulative rewards. This method doesn't rely on labeled datasets; instead, the agent explores various strategies, learning from the outcomes of its actions to improve performance over time.

In contrast, Supervised Fine-Tuning (SFT) involves training a model on a labeled dataset where each input is paired with the correct output. The model learns to map inputs to desired outputs by minimizing the difference between its predictions and the actual labels during training. This process requires a substantial amount of labeled data and focuses on learning from examples provided by human annotators.

The key differences between Pure RL and SFT are:

Learning Process: Pure RL learns through trial and error interactions with the environment, adjusting actions based on feedback, whereas SFT learns from a predefined set of input-output pairs.

Data Requirements: Pure RL doesn't require labeled data but needs a well-defined environment to interact with. SFT requires a large, labeled dataset for training.

Application Scenarios: Pure RL is suitable for tasks where an agent must make a sequence of decisions to achieve a goal, such as game playing or robotic control. SFT is ideal for tasks where the correct output can be directly provided, like image classification or language translation.

An example of applying Pure RL in training language models is DeepSeek's R1 model, which employs a streamlined variant of reinforcement learning, significantly reducing training complexity and data collection costs.

Another post-training strategy not described in the book is Self-Training with Verifiable Rewards (RLVR). This approach involves generating samples from the model, filtering them using binary feedback (correct or incorrect), fine-tuning the model on these samples, and repeating the process. This method allows the model to improve its performance by learning from its own generated data, reducing dependence on human-generated data

reading guide week 3.docx

Alto, V. 2024. Chapter 3. Choosing an LLM for your application

1. Differences Between Encoder-Decoder and Decoder-Only Transformer-Based Architectures

Encoder-decoder and decoder-only architectures both stem from the Transformer model, but they function differently:

Encoder-Decoder (e.g., T5, BART):

How it works: The encoder processes the input sequence, converting it into a fixed-length representation, which the decoder then uses to generate the output sequence.

Best for: Tasks requiring input transformation, such as machine translation, text summarization, and question answering.

Example: A translation model where the encoder encodes an English sentence, and the decoder generates the corresponding French sentence.

Decoder-Only (e.g., GPT-4, LLaMA-2):

How it works: The model generates text autoregressively, predicting each next token based only on previously generated tokens.

Best for: Text generation, code completion, and chatbot applications, where the model doesn't need to transform structured inputs into structured outputs.

Example: ChatGPT, where a user prompt serves as context, and the model generates relevant responses.

👉 Key takeaway: Encoder-decoder models are better suited for structured tasks like translation, while decoder-only models excel in generating fluent and coherent text-based responses.

2. How RLHF (Reinforcement Learning from Human Feedback) Works

RLHF is a method used to align LLMs with human intent by optimizing them based on human feedback. The process follows two key steps:

Training a Reward Model

Human labelers rank multiple responses generated by an LLM based on helpfulness, accuracy, and safety.

These rankings are used to train a reward model that predicts human preferences.

Optimizing the LLM via Reinforcement Learning

The model generates text and receives a reward score from the reward model.

Using reinforcement learning (e.g., Proximal Policy Optimization, PPO), the model adjusts itself to maximize human-aligned responses.

👉 Example: ChatGPT uses RLHF to improve conversational quality by learning from user feedback and refining responses to be less biased, more informative, and more engaging.

3. What is MoE (Mixture-of-Expert) Transformer Architecture?

MoE is a scalable AI model architecture that divides tasks among multiple specialized sub-models, or "experts."

Instead of a single large model handling everything, different experts specialize in different types of data or tasks.

A router mechanism determines which expert should process a given input, reducing computation cost while maintaining performance.

👉 Example: Google's Gemini 1.5 uses MoE to improve efficiency and reduce computational load, making it more scalable.

4. What is CAI (Constitutional AI)?

CAI is a method developed by Anthropic to ensure that AI models are helpful, honest, and harmless by following a set of ethical principles.

Instead of relying solely on RLHF, CAI guides the model's behavior using predefined rules (e.g., UN Human Rights principles).

The model critiques and revises its own responses based on these principles, reducing the chances of generating biased, misleading, or harmful outputs.

👉 Example: Claude 2 uses CAI to improve safety, ensuring that it doesn't engage in harmful behaviors or spread misinformation.

5. Differences Between a Base Model and a "Chat" Model

Base Model:

Trained on a large dataset to predict the next token.

Has broad knowledge but lacks specific instruction-following capabilities.

Example: LLaMA-2, which can generate text but is not optimized for conversation.

Chat Model:

A base model fine-tuned with instruction-following data and RLHF.

Optimized for conversations, with reduced hallucinations and improved contextual understanding.

Example: LLaMA-2-Chat, specifically trained to act as an assistant.

👉 Key takeaway: Base models excel at general text generation, while chat models are optimized for conversational applications.

6. What Makes Falcon LLM Unique?

Falcon LLM is different from other models because it prioritizes training data quality over sheer model size:

Optimized Dataset: Uses RefinedWeb, a high-quality, filtered web dataset, making it more efficient than larger models trained on noisy data.

Smaller but Powerful: With only 40 billion parameters, it competes with much larger models like GPT-3 and PaLM-62B.

Apache 2.0 License: Fully open-source and commercially available, making it a strong alternative to proprietary models.

👉 Key takeaway: Falcon LLM is an efficient, high-performance model that achieves state-of-the-art results while using less computing power.

7. Grouped-Query Attention (GQA) and Sliding-Window Attention (SWA)

Grouped-Query Attention (GQA)

Reduces inference time by grouping attention heads together.

Makes processing faster and lowers memory usage, ideal for low-latency applications.

Sliding-Window Attention (SWA)

Allows the model to attend to a fixed-size sliding window instead of all tokens, making it more efficient for long documents.

Helps models handle longer contexts without dramatically increasing computation.

👉 Example: Mistral-7B uses GQA and SWA to optimize efficiency, outperforming many larger models in benchmarks.

8. Key Factors in Choosing a Foundation Model for Your Application

Choosing the right LLM depends on multiple trade-offs:

1. Performance vs. Cost

Larger models (e.g., GPT-4, Claude 2) → Better performance, but higher cost and latency.

Smaller models (e.g., Falcon, Mistral) → Cheaper and faster, but less powerful for complex reasoning tasks.

2. Open-Source vs. Proprietary

Proprietary (GPT-4, Claude 2): Easy to use, strong support, but expensive and non-transparent.

Open-Source (LLaMA-2, Falcon): Customizable, free, but requires more infrastructure.

3. Fine-tuning Needs

Open-source models (e.g., Falcon, Mistral) allow fine-tuning for domain-specific tasks.

Proprietary models often limit fine-tuning options (GPT-4 fine-tuning is restricted).

4. Domain-Specific Capabilities

MMLU → General knowledge (GPT-4, Claude 2).

TruthfulQA → Truthfulness (Claude 2 excels).

HumanEval → Coding (Claude 2 > GPT-4 > LLaMA).

Finance → BloombergGPT

Medical → BioMedLM

5. Deployment Strategy

Cloud-based (e.g., GPT-4, Claude 2) → No infrastructure required.

On-premises (e.g., Falcon, Mistral) → More control but needs computing resources.

Final Thoughts

Selecting the right foundation model depends on performance, cost, customization, and domain specificity. If scalability and multimodal support are priorities, GPT-4 is a strong choice. If customization and cost-effectiveness matter more, then LLaMA-2 or Falcon may be better.

👉 Bottom line: There is no one-size-fits-all solution. The best model depends on your specific application needs.

Here are the latest news, events, and knowledge updates in the field of Large Language Models (LLMs):

1. The Application of Mixture-of-Experts (MoE) in LLMs

MoE is an innovative technique that integrates multiple specialized sub-models (or "experts") within a large language model, significantly improving performance and efficiency. Recently, Mistral AI released the open-source Mixtral 8x7B model, which adopts an MoE architecture and demonstrates the potential of MoE in LLMs.

2. Advances in Reinforcement Learning from Human Feedback (RLHF)

RLHF is a machine learning technique that uses direct human feedback to train a reward model, which is then used to optimize AI models through reinforcement learning. However, one major challenge in RLHF is the efficiency of collecting human feedback. To address this, OpenAI has introduced a Rule-Based Rewards (RBR) mechanism, aimed at improving the ability of AI models to comply with safety policies.

3. The Development of Constitutional AI (CAI)

Constitutional AI (CAI) is a technique where a pre-trained language model acts as a feedback model, replacing human annotators in ranking responses to ensure AI safety. This approach is known as Reinforcement Learning from AI Feedback (RLAIF) and reduces dependence on human supervision.

4. The Release of DeepSeek-V3

In January 2025, the Chinese AI company DeepSeek launched DeepSeek-V3, a large-scale language model based on Mixture-of-Experts (MoE) architecture. This new model aims to improve computational efficiency while maintaining high performance, positioning itself as a strong competitor in the LLM landscape.

These updates indicate a shift toward more efficient and scalable architectures (MoE), alternative approaches to RLHF, and the emergence of powerful new models in the AI field. 🚀

Huyen 2025: Chapter 3: Evaluation Methodology

1. What are the challenges of evaluating foundation models?

Evaluating foundation models is much harder than evaluating traditional machine learning models because of their open-ended nature and complexity. Here are the key challenges:

Increased Complexity – The more advanced a model is, the harder it is to determine whether it's producing correct outputs. A basic math solution is easy to check, but a PhD-level solution? Not so much.

No Ground Truth – Traditional models have clear right and wrong answers (like classification tasks), but foundation models often generate multiple valid responses, making standard evaluation methods unreliable.

Black Box Problem – Many foundation models operate as black boxes, meaning we don't have full visibility into their architecture, training data, or reasoning processes.

Benchmark Saturation – Public benchmarks quickly become outdated as models improve, making it hard to measure real progress.

Scope Expansion – Unlike task-specific models, foundation models can perform unexpected tasks, requiring evaluation methods that go beyond predefined benchmarks.

The key takeaway? Evaluating foundation models isn't just about checking if they work—it's about continuously updating evaluation methods to keep up with their evolving capabilities.

2. Describe cross-entropy, bits-per-character, and perplexity, and explain their relationships.

These are all metrics used to evaluate language models:

Cross-Entropy – Measures how well a model predicts the next token in a sequence. A lower value means the model is better at predicting text.

Bits-Per-Character (BPC) – A variation of cross-entropy that expresses how many bits the model needs to encode each character. Lower BPC = better compression and efficiency.

Perplexity (PPL) – The exponential of cross-entropy. It measures how "confused" a model is when predicting the next token. Lower perplexity means better predictions.

In short: cross-entropy and perplexity are directly related (perplexity is just an exponentiation of cross-entropy). BPC is another way to express cross-entropy in terms of characters rather than tokens.

3. What is functional correctness? How would you measure a foundation model using this metric?

Functional correctness evaluates whether a model's output actually works in practice. It's common in code generation and structured tasks.

For example, in AI-generated code:

You can check if the code compiles.

You can run unit tests to verify correctness.

You can measure pass@k, which evaluates how often the model generates correct solutions.

For SQL generation tasks:

The generated query can be run on a database to check if it returns the expected results.

Functional correctness is one of the most reliable evaluation methods, but it's limited to tasks with clear, objective correctness criteria—which many foundation model applications don't have.

4. How can similarity measurements be used to evaluate a foundation model?

Similarity measurements compare model-generated outputs to a set of reference answers to determine how close they are. They are useful in:

Machine Translation – Checking if an AI-generated translation aligns with a human reference.

Text Summarization – Measuring how closely an AI-generated summary resembles a human-written one.

There are two main types:

Lexical Similarity – Measures how similar the words are.

Semantic Similarity – Measures how similar the meaning is.

Similarity metrics are widely used but have limitations—they assume that reference answers are perfect, which is rarely the case.

5. Describe lexical similarity and semantic similarity. What are the differences between them?

Lexical Similarity – Measures overlap in exact words or characters (e.g., BLEU, ROUGE scores).

Semantic Similarity – Measures meaning (e.g., BERTScore, cosine similarity of embeddings).

Example:

Lexically different, but semantically similar:
"How are you?" vs. "What's up?" → Low lexical similarity, high semantic similarity.

Lexically similar, but semantically different:
"Let's eat, grandma." vs. "Let's eat grandma." → High lexical similarity, but very different meaning.

Lexical similarity is easier to compute but less flexible, while semantic similarity captures meaning better but requires more computation.

6. Why would you use an AI judge for evaluating foundation models? How would you implement this and what are the limitations of this approach?

AI judges automate evaluation by having one AI model evaluate another. Why use them?

Faster and cheaper than humans – AI judges can evaluate thousands of responses instantly.

Flexible criteria – You can ask them to judge correctness, fluency, toxicity, etc.

Scalable – AI judges work well for large-scale evaluations where human evaluation isn't practical.

Implementation:

Use a strong AI model (e.g., GPT-4) as the judge.

Provide clear instructions and scoring rubrics.

Compare AI-generated responses against reference answers or other model responses.

Limitations:

Inconsistency – AI judges sometimes give different scores for the same input.

Bias – AI judges tend to favor their own model's outputs.

Cost – Using powerful AI models as judges can be expensive.

Lack of standardization – Different AI judges use different scoring systems, making results hard to compare.

Bottom line? AI judges are useful, but they should be supplemented with human evaluation and exact metrics.

7. Describe the three specialized judges: reward models, reference-based judges, and preference models.

There are different types of AI judges, each suited for specific tasks:

Reward Models

Assign scores based on how well a response meets predefined criteria.

Used in Reinforcement Learning from Human Feedback (RLHF).

Example: Google's Cappy model (scores responses from 0 to 1).

Reference-Based Judges

Compare responses against reference answers.

Example: BLEURT, which assigns similarity scores to text.

Problem: If the reference answers are flawed, the evaluation is flawed.

Preference Models

Rank responses based on human-like preferences.

Example: PandaLM, which predicts which response users will prefer.

Very useful for ranking chatbots or summarization models.

Each judge type has trade-offs—reference-based judges need good reference data, preference models require user feedback, and reward models need fine-tuning.

8. How would you conduct a comparative evaluation of foundation models? What are the challenges and advantages associated with this approach?

Comparative evaluation ranks models by directly comparing their responses rather than scoring them independently.

How to do it?

Select multiple models and generate responses for the same set of prompts.

Have human evaluators or AI judges pick the better response in each comparison.

Use ranking algorithms like Elo or Bradley-Terry to compute rankings.

Advantages:

Easier for subjective tasks – People find it easier to compare two responses than to assign scores.

Captures real-world preferences – Models are judged based on user preferences.

Harder to game – Unlike benchmarks, models can't just train on test data.

Challenges:

Scalability – The number of comparisons grows quadratically as more models are added.

No absolute performance – Tells you which model is better, but not whether it's good enough.

Bias in user comparisons – Some users prefer longer responses or specific writing styles, which may skew rankings.

Despite these challenges, comparative evaluation is becoming the standard for ranking AI models, especially in leaderboards like LMSYS Chatbot Arena.

Final Thoughts

Foundation model evaluation is not just about accuracy—it’s about finding the best model for your use case. Whether using functional correctness, similarity measurements, or AI judges, the key is to combine multiple evaluation methods for reliable and fair assessments.

Want a perfect evaluation method? Doesn’t exist. But by mixing different approaches, we can get closer to a meaningful ranking that actually helps us build better AI systems.

reading guide week 4.docx

Alto 2024: Chapter 4 – Key Takeaways & Insights

1. What is prompt engineering? What is a prompt? Describe a simple prompt that you often use for learning in this class.

Prompt engineering is about designing the right input to get the best response from an AI model. It’s like giving clear directions to a GPS—if the input is vague, the output won’t be useful.

A prompt is the actual instruction or question given to the AI model. It tells the model what to do and how to respond.

A simple prompt I use:

In this class, I often use something like:

Explain [concept] in simple terms with an example.

This helps break down complex ideas into something more digestible.

2. Explain the importance of “providing clear instructions” in prompt engineering. Support your answer with an example of your own.

If the instructions are unclear, the AI model might misinterpret the request or provide incomplete information.

Example:

Unclear prompt: "Tell me about databases."

Clear prompt: "Explain relational databases, how they differ from NoSQL, and give an example of each."

The second prompt guides the AI to focus on key areas, making the response more useful.

3. Explain the importance of “splitting complex tasks into subtasks” in prompt engineering. Support your answer with an example of your own.

AI models handle simple tasks better than complex, multi-step ones. Breaking down a task helps ensure each part gets the right attention.

Example:

Instead of saying, "Summarize this article in bullet points and give me a tweet-length version," I could split it into:

"Extract key points from this article."

"Rewrite the key points in bullet form."

"Create a tweet-length summary based on the bullet points."

This improves accuracy and ensures nothing important is skipped.

4. Explain the importance of “asking for justification” in prompt engineering. Support your answer with an example of your own.

Asking the AI to explain its reasoning improves reliability and transparency. It forces the model to step through its logic instead of just guessing an answer.

Example:

Instead of just asking, "What is the best cloud provider?" I can ask:

Compare AWS, Azure, and Google Cloud based on pricing, scalability, and ease of use. Justify your ranking.

This prevents vague answers and ensures the model considers different factors.

5. Explain the importance of “generating multiple responses and selecting the best one” in prompt engineering. Support your answer with an example of your own.

Sometimes the first response isn’t the best. Asking for multiple responses allows better comparisons.

Example:

Instead of just saying, "Suggest a name for my tech startup," I can prompt:

Give me five potential names for a tech startup focused on AI automation. Explain why each one works.

This gives me more options and reasoning, making it easier to choose the best one.

6. Explain why you may want to “repeat instruction at the end.”

Sometimes AI models “forget” earlier instructions in long prompts. Repeating instructions at the end reinforces the main request.

Example: If I say:

Summarize this article in 3 bullet points. Do not exceed 10 words per bullet point.

Adding at the end:

Remember, keep each bullet point under 10 words.

This acts as a reminder to the model, reducing errors.

7. How to use delimiters in prompt engineering? Discuss quotation marks, triple backticks, brackets, and dashes. What are the benefits of using delimiters?

Delimiters organize prompts and clearly define different sections.

Common delimiters and their use cases:

Quotation marks (" " or ' '): Highlight specific phrases or text inputs.

Triple backticks (python or text): Format code snippets or structured data.

Brackets ([ ], { }, ( )): Indicate placeholders or optional inputs.

Dashes (--- or ====): Separate different sections in a prompt.

Benefits:
✅ Avoids confusion between instructions and examples.
✅ Helps the model understand where different elements start and end.
✅ Improves formatting for structured responses.

8. What is few-shot learning? Discuss how it works in prompt engineering.

Few-shot learning means giving the model a few examples to guide its response.

How it works: Instead of saying, "Classify this review as positive or negative," I give it a few labeled examples:

Text: "I love this product!" → Positive

Text: "Terrible service." → Negative

Text: "The food was great!" → Positive

Now classify: "This movie was boring."

The model learns the pattern and applies it to new inputs without needing fine-tuning.

9. What is chain of thought (CoT)? Discuss how it works in prompt engineering.

Chain of Thought (CoT) makes AI explain step-by-step reasoning before answering.

How it works:

Instead of just solving an equation, I make it show the full process:

Solve for x: 3x + 5 = 11

1. Move 5 to the other side → 3x = 6

2. Divide by 3 → x = 2

Why use it?
✅ Improves accuracy.
✅ Helps debug mistakes.
✅ Makes AI's reasoning transparent.

10. What is ReAct? How does it work? What are the differences between ReAct and CoT?

ReAct (Reason + Act) combines reasoning and actions.

It makes AI think step-by-step AND take action, like searching for facts online.

How it works:

Instead of just answering, ReAct makes the AI search for real-time data:

Question: "Who are the Italian male climbers for the Paris 2024 Olympics?"

1. AI thinks: "I should check current news."

2. AI searches online.

3. AI updates its answer based on real-time results.

Differences between ReAct and CoT:

When to use ReAct?

If I need factual, up-to-date answers, like sports results or news.

Final Thoughts

Prompt engineering is not just about asking questions—it's about structuring inputs to get the best outputs.

Few-shot learning helps models adapt quickly.

CoT improves logic, and ReAct enhances real-time accuracy.

Using delimiters, breaking tasks down, and refining prompts can significantly boost AI performance.

Here are the answers to your questions based on Huyen 2024: Chapter 5, formatted to match your writing style:

Huyen 2024: Chapter 5 - Key Questions and Answers

1. In this book, what does "context" mean? What does "prompt" mean?

Context refers to the information provided within a model's input, which helps shape its response. It can include system instructions, user messages, relevant documents, and prior conversation history.

Prompt is the specific input given to the model, which includes instructions, questions, or examples that guide the AI's response.

2. What is in-context learning? What is zero-shot learning?

In-context learning is when a model learns from the examples provided in the prompt itself, without additional training.

Zero-shot learning is when the model performs a task without seeing examples beforehand—essentially, it applies general knowledge to answer the question.

3. This is a refresher – describe a system prompt and a user prompt.

System prompt sets the AI's behavior, tone, and scope. Example: "You are an expert career coach. Give concise and actionable career advice."

User prompt is the actual input given by the user. Example: "How do I negotiate my salary for a data analyst role?"

4. How do the needles in a haystack (NIAH) work?

The Needle in a Haystack (NIAH) test evaluates how well a model can retrieve specific information from a long prompt. Research shows that AI models are better at recalling details from the beginning and end of a prompt rather than the middle.

5. What are some best practices in prompt engineering mentioned in this chapter? Compare them with Alto's book.

Huyen's Best Practices:

Write clear and explicit instructions.

Ask the model to adopt a persona (e.g., a first-grade teacher vs. a college professor).

Provide examples to guide the model's responses.

Specify the output format to avoid unnecessary or inconsistent responses.

Break complex tasks into subtasks to improve accuracy and efficiency.

Encourage "thinking" by using Chain-of-Thought (CoT) prompting.

Comparison with Alto's Book:

Alto also emphasizes clear instructions and step-by-step reasoning, but it focuses more on interactive refinement (iterating prompts) rather than decomposition.

Huyen introduces defensive prompt engineering, which is not a major focus in Alto's work.

6. How can AI models help prompt engineering?

AI can generate and optimize prompts automatically, reducing human effort.

Tools like Promptbreeder evolve prompts through mutations to find the most effective versions.

AI models can critique and refine prompts, ensuring clarity and efficiency.

7. What are the potential risks of prompt attacks?

Remote code execution (e.g., an AI mistakenly running unauthorized code).

Data leaks (revealing private or confidential information).

Social harm (AI being tricked into spreading dangerous or offensive content).

Misinformation (bad actors manipulating AI responses).

Service subversion (hijacking AI to approve fraudulent activities).

Brand risk (AI making inappropriate statements under a company's name).

8. Describe prompt extraction in prompt attacks. How you may defend against it.

Prompt Extraction: Attackers attempt to trick AI into revealing its hidden system prompt, which could be used to reverse-engineer or manipulate the system.

Defense:

Assume any prompt could become public and avoid including sensitive details.

Use structured prompt management (e.g., breaking prompts into multiple parts).

Implement monitoring and red teaming to detect and prevent leaks.

9. Describe jailbreaking and prompt injection and how you may defend against it.

Jailbreaking: Tricking an AI into ignoring its built-in safety measures, making it answer restricted or harmful queries.

Prompt Injection: Inserting malicious instructions within a user query to manipulate AI behavior.

Defense:

Train models to prioritize system instructions over user inputs.

Detect common attack patterns, such as roleplaying or obfuscation.

Isolate sensitive actions, requiring human approval for risky operations.

10. Describe information extraction in prompt attacks and how you may defend against it.

Information Extraction: Attackers attempt to extract sensitive data (e.g., private emails, proprietary training data, or copyrighted content).

Defense:

Implement filters to block prompts requesting PII (Personally Identifiable Information).

Limit the AI's access to external databases unless necessary.

Use anomaly detection to flag suspicious behavior (e.g., repeated attempts to extract similar data).

reading guide week 5.docx

Alto 2024: selected texts from Chapters 5 and 6

1. What is a completion model? What is a chat model? What are the key differences?

Completion Model: A completion model takes an input prompt and generates a single response as a continuation of the text. These models are stateless, meaning they do not inherently remember previous interactions.

Example: GPT-3.5 text-davinci-003

Use Case: Generating text summaries, story writing, code completion

Chat Model: A chat model is designed for multi-turn conversations. It takes a series of messages as input (e.g., user prompts, system messages) and generates context-aware responses. These models are optimized for conversational tasks and often include memory mechanisms to handle context.

Example: GPT-3.5-turbo, GPT-4

Use Case: Chatbots, customer support, virtual assistants

Key Differences

2. What is a vector store? How do you measure similarities between two vectors?

Vector Store: A vector store (or vector database) stores vector embeddings of text, images, or other data. These embeddings represent the semantic meaning of the content in a high-dimensional space, allowing for efficient similarity search.

Examples: FAISS (Meta), Pinecone, Weaviate, ChromaDB

Measuring Similarities Between Two Vectors

Cosine Similarity: Measures the angle between two vectors. Values range from -1 (opposite) to 1 (identical). Commonly used in NLP tasks. cos(θ)=A·B∥A∥∥B∥\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}

Euclidean Distance: Measures the straight-line distance between two points in vector space. d(A,B)=∑i=1n(Ai−Bi)2d(\mathbf{A}, \mathbf{B}) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}

Dot Product: Measures similarity based on magnitude and direction. Larger values indicate higher similarity.

Use Case Example

A travel chatbot using FAISS can store travel guides as embeddings. When a user asks, “What should I visit in Rome?”, the system retrieves the most relevant travel recommendations based on cosine similarity.

3. How would you design a memory system for a GenAI application?

Scenario: GlobeBotter Travel Assistant

Problem: Users interact with the GlobeBotter AI, asking about travel recommendations. The chatbot needs to remember user preferences across multiple sessions.

Memory System Design

ConversationBufferMemory: Stores the entire chat history, allowing for continuity.

ConversationSummaryMemory: Summarizes long conversations into concise summaries, keeping responses relevant.

Vector Store Memory (FAISS/Pinecone): Stores user queries and responses as embeddings, allowing semantic recall.

Example

User: "I'm planning a 7-day trip to Japan. I love history and food."

AI (After storing memory): "Since you love history, I recommend visiting Kyoto’s temples and trying local ramen."

📌 Key Benefit: The AI remembers the user’s interests, providing personalized recommendations.

4. What are the different types of Chains in LangChain?

Chains in LangChain help orchestrate multiple steps in an LLM-powered application.

5. What are agents in LangChain? Describe different types of agents under the ReAct approach.

Agents are dynamic decision-makers in LangChain that use LLMs to plan and execute actions.

📌 Key Difference from Chains:

Chains = Predefined steps

Agents = Decide actions dynamically

ReAct (Reasoning + Acting) Approach

ReAct agents think before acting, combining reasoning and tool use.

6. What is a Streamlit callback handler? How can it be used in a LangChain-powered chatbot?

📌 Streamlit Callback Handler is a real-time UI component that visualizes LangChain agent actions in Streamlit apps.

How It Works

Tracks agent execution (e.g., tool usage, thought process)

Displays intermediate steps (like tool calls & responses)

Enables streaming responses (token-by-token updates)

Example: Integrating Streamlit with LangChain

```
from langchain.callbacks.base import BaseCallbackHandler

import streamlit as st


# Define a Streamlit Callback Handler

class StreamHandler(BaseCallbackHandler):

    def __init__(self, container, initial_text=""):

        self.container = container

        self.text = initial_text


    def on_llm_new_token(self, token: str, **kwargs) -> None:

        self.text += token

        self.container.markdown(self.text)


# Streamlit Chatbot Interface
```

```
st.header("GlobeBotter - AI Travel Assistant")

user_query = st.text_input("Ask me anything about your next trip!")


if user_query:

    with st.chat_message("assistant"):

        stream_handler = StreamHandler(st.empty())

        response = agent(user_query, callbacks=[stream_handler])

        st.write(response)
```

Key Benefits

Better user experience (real-time streaming)

Visual debugging (see AI reasoning & tool use)

Seamless integration (works with existing LangChain agents)

📌 Summary Table

🚀 Next Steps: Explore LangChain + Streamlit to build interactive AI applications!

reading guide week 6.docx

1. Describe retrieve-then-generate pattern.

The retrieve-then-generate pattern is a two-step process that enhances a model's ability to generate responses by first retrieving relevant external information before generating an output. This pattern was first introduced in Reading Wikipedia to Answer Open-Domain Questions (Chen et al., 2017) and later formalized as Retrieval-Augmented Generation (RAG) by Lewis et al. (2020).

Step 1: Retrieval – The model retrieves relevant documents or data chunks from an external knowledge source (e.g., a database, previous chat history, or the internet). This ensures that only the most relevant information is included in the model's input.

Step 2: Generation – The retrieved information is then used as additional context for the model, allowing it to generate more accurate, detailed, and factually correct responses while minimizing hallucinations.

The key benefit of this approach is that it allows the model to access up-to-date and domain-specific information without requiring direct retraining or storing all knowledge in its parameters.

2. What are the main components of RAG?

A Retrieval-Augmented Generation (RAG) system consists of two main components:

Retriever – The retriever fetches relevant information from an external knowledge base (e.g., databases, previous conversations, document stores, vector databases).

Indexing: Prepares and organizes the data for efficient retrieval.

Querying: Fetches the most relevant data based on a user's query.

Common retrieval methods:

Term-based retrieval (Sparse): Uses traditional search techniques like BM25 and Elasticsearch.

Embedding-based retrieval (Dense): Uses vector databases and similarity search methods.

Generator – The generator uses the retrieved context to produce a response.

Typically based on a large language model (LLM) or another text generation model.

Takes both the user's query and the retrieved information as input.

Produces a coherent and relevant response that integrates external knowledge.

Some enhancements include:

Hybrid retrieval (combining sparse and dense methods for better results).

Reranking (sorting retrieved documents to ensure the most relevant ones are prioritized).

Memory integration (storing previous interactions for personalized responses).

3. The book covered a few different chunking strategies. Explain them.

Chunking is the process of splitting documents into smaller, manageable pieces to improve retrieval efficiency in RAG systems. The chunking strategy significantly affects how well a retriever can find relevant information. The book discusses several methods:

Fixed-Length Chunking

Splits documents into equal-sized chunks based on characters, words, sentences, or paragraphs.

Example: Every chunk contains 512 words or 2048 characters.

Pros: Simple to implement.

Cons: May split important contextual information between chunks.

Recursive Chunking

Splits documents hierarchically, starting from sections → paragraphs → sentences until each chunk is small enough.

Pros: Preserves more context by keeping related information together.

Cons: More complex and computationally expensive.

Overlapping Chunking

Ensures that important context isn’t lost by allowing chunks to overlap by a certain percentage.

Example: If a chunk is 2048 characters, the next chunk overlaps by 20 characters.

Pros: Reduces the risk of missing critical context.

Cons: Increases retrieval storage and redundancy.

Semantic Chunking

Uses machine learning or NLP techniques to identify logical breaks in content (e.g., breaking at topic changes rather than at arbitrary points).

Pros: More intelligent chunking that aligns with meaning rather than length.

Cons: Requires additional computational resources.

Token-Based Chunking

Splits text into tokens, ensuring chunks fit within the context limit of the generative model.

Example: If using Llama 3, chunking would align with Llama 3’s tokenizer.

Pros: Avoids exceeding token limits.

Cons: Needs to be adjusted when switching to a different model with a different tokenizer.

Metadata-Augmented Chunking

Each chunk is enriched with metadata such as tags, keywords, summaries, or title information to improve retrieval accuracy.

Example: A chunk from a product catalog could include price, category, and customer reviews in its metadata.

Pros: Helps retrievers find relevant information more efficiently.

Cons: Requires extra processing to generate metadata.

Conclusion

Fixed-length chunking is simple but may lose important context.

Overlapping chunking reduces information loss but increases redundancy.

Semantic and metadata-enhanced chunking are more advanced but require additional processing power.

Selecting the right chunking strategy depends on the retrieval method, dataset structure, and the trade-off between speed and accuracy.

1. What is the objective of creating a knowledge base in the RAG architecture?

The objective of creating a knowledge base in the Retrieval-Augmented Generation (RAG) architecture is to establish a central repository of easily searchable and digestible information. This step ensures that the system can efficiently retrieve relevant knowledge when responding to user queries.

Specifically, the knowledge base is designed to:

Structure and organize data in a way that facilitates quick and accurate retrieval.

Improve searchability by breaking down large documents into smaller, more manageable chunks.

Enhance the chatbot's response quality by ensuring that the retrieved information is concise, relevant, and up to date.

Convert text into a machine-readable format (vector embeddings), allowing for semantic search rather than relying solely on keyword matching.

By properly structuring the knowledge base, RAG-based chatbots can deliver more precise and contextually relevant responses.

2. What are the key considerations in this process?

Several key considerations must be taken into account when creating a knowledge base in a RAG system:

Document Chunking Strategy

How to split documents into smaller chunks: The method used for breaking down information impacts retrieval accuracy.

Choosing an optimal chunk size: Too large may contain unnecessary data, while too small may lose context.

Balancing precision and recall: Ensuring the retrieved chunks are relevant without overloading the model with unnecessary context.

Representation Methods

Choosing the right vectorization model to convert text into embeddings for semantic search.

Optimizing storage and retrieval: Balancing accuracy and cost when selecting representation methods.

Metadata and Filtering

Adding relevant metadata (e.g., source location, access rights, timestamps) to improve filtering and precision.

Ensuring proper access control: Defining user permissions to maintain data security.

Scalability and Performance

Handling large-scale document libraries efficiently (e.g., processing millions of records).

Optimizing search speed and retrieval cost while maintaining high accuracy.

Integration with Retrieval Mechanisms

Ensuring that the knowledge base aligns with the document retrieval system for effective search queries.

Allowing flexible adaptation to different data sources and evolving information.

By carefully considering these factors, the knowledge base can significantly enhance the efficiency, accuracy, and reliability of the RAG-powered chatbot.

reading guide week 7.docx

1. Explain Elasticsearch

Elasticsearch is a distributed, open-source search and analytics engine designed for handling large amounts of structured and unstructured data. It is based on Apache Lucene and is commonly used for full-text search, logging, and data analysis.

Key Features of Elasticsearch:

Scalability: Supports horizontal scaling with sharding and replication.

Real-time search: Provides near real-time search capabilities.

JSON-based REST API: Communicates via RESTful APIs using JSON.

Full-text search: Uses inverted indexes and BM25 scoring for efficient text search.

Distributed architecture: Stores data across multiple nodes for fault tolerance.

Support for structured and unstructured data: Handles logs, text, numbers, and geospatial data.

How Elasticsearch Works:

Indexing:

Data is indexed and stored in an inverted index for fast lookups.

Documents are stored in JSON format.

Searching:

Uses BM25 scoring for ranking search results.

Supports fuzzy search, autocomplete, faceted search, and filtering.

Querying:

Elasticsearch supports two types of queries:

Term-based search (keyword-based, exact match).

Full-text search (analyzes the text, breaks it into terms).

Sharding and Replication:

Large indices are divided into shards for parallel processing.

Replication ensures high availability and fault tolerance.

🔹 Use Cases: Log analysis (ELK stack), e-commerce search, recommendation systems, application monitoring.

2. Explain the BM25 Retrieval Algorithm

BM25 (Best Matching 25) is a ranking function used in information retrieval to score documents based on their relevance to a search query.

How BM25 Works:

It scores documents based on term frequency (TF) and inverse document frequency (IDF).

Formula:
BM25(D,Q)=∑t∈QIDF(t)×TF(t,D)×(k1+1)TF(t,D)+k1×(1−b+b×∣D∣avgD)\text{BM25}(D, Q) = \sum_{t \in Q} IDF(t) \times \frac{TF(t, D) \times (k_1 + 1)}{TF(t, D) + k_1 \times (1 - b + b \times \frac{|D|}{\text{avgD}})} Where:

TF(t, D): Term frequency of t in document D.

IDF(t): Inverse document frequency.

|D|: Length of document D.

avgD: Average document length.

k1, b: Hyperparameters to adjust term weighting.

k1: Controls term frequency saturation.

b: Controls document length normalization.

🔹 Why is BM25 Effective?

Gives higher scores to shorter, more relevant documents.

Handles stop words efficiently by adjusting term weighting.

Commonly used in Elasticsearch, Lucene, and search engines.

🔹 Use Cases: Text search, ranking in search engines (Google, Bing), recommendation systems.

3. Describe How an Embedding-Based Retriever Works

An embedding-based retriever ranks documents based on semantic similarity instead of exact keyword matches.

How It Works:

Convert Text into Vectors:

Each document and query is converted into a high-dimensional vector using embedding models (e.g., BERT, OpenAI embeddings).

Example: "Artificial Intelligence" → [0.2, 0.8, 0.1, ...]

Index the Embeddings:

Store embeddings in a vector database (e.g., FAISS, Milvus, Pinecone, PGVector).

Retrieve the Most Similar Vectors:

When a query is issued, it is also converted into an embedding.

The system finds K nearest neighbors (K-NN) based on cosine similarity or Euclidean distance.

Rank and Return the Top-K Documents:

The retrieved documents are ranked by similarity scores.

🔹 Why Use Embedding-Based Retrieval?

Captures semantic meaning (e.g., "laptop" ~ "notebook").

Works well for synonyms, paraphrasing, and context understanding.

Can be fine-tuned for domain-specific retrieval (e.g., medical, legal, finance).

🔹 Use Cases:

Semantic search (ChatGPT-style RAG).

Question answering systems.

Recommendation engines.

4. Explain PGVector Indexing: IVFFlat and HNSW

PGVector is a vector search extension for PostgreSQL, commonly used for embedding-based retrieval.

IVFFlat (Inverted File Index)

How it Works:

Clusters embeddings into partitions using K-means clustering.

At query time, the retriever searches only the nearest clusters instead of scanning all vectors.

Pros:

Faster search (compared to brute-force K-NN).

Works well for large-scale retrieval.

Cons:

May have lower accuracy due to cluster-based approximation.

Needs careful tuning of the number of clusters.

HNSW (Hierarchical Navigable Small World)

How it Works:

Builds a multi-layer graph where similar vectors are linked together.

Search is performed by navigating the graph to find the closest matches.

Pros:

Fast and highly accurate.

Works well for high-dimensional vectors.

Cons:

Requires more memory than IVFFlat.

Indexing is slower than IVFFlat.

🔹 Choosing Between IVFFlat and HNSW

✅ Use IVFFlat if you need fast retrieval with a large dataset.
✅ Use HNSW if you need high accuracy and can afford more memory.

5. How to Choose Between Term-Based and Embedding-Based Retrieval for RAG?

The choice depends on use case, cost, and retrieval quality.

✅ When to Use Term-Based Retrieval (BM25)?

If the data is structured (e.g., product catalogs, legal documents).

If queries use exact keywords.

If low computational cost is important.

✅ When to Use Embedding-Based Retrieval?

If the goal is semantic search (understanding intent, synonyms).

If text is unstructured (e.g., customer support, RAG systems).

If queries are in natural language (e.g., “What are the symptoms of COVID-19?”).

✅ Hybrid Approach: Combine Both

Use BM25 for initial candidate retrieval (fast filtering).

Use embedding-based retrieval for re-ranking (higher accuracy).

Conclusion

Elasticsearch is a scalable, distributed search engine using BM25.

BM25 ranks documents based on TF-IDF with length normalization.

Embedding-based retrieval uses vector similarity for semantic search.

PGVector (IVFFlat vs. HNSW) provides optimized vector indexing.

Choosing term-based vs. embedding-based retrieval depends on use case and cost.

Hybrid retrieval (BM25 + embeddings) offers the best of both worlds for RAG systems.

🚀 For a RAG system, a hybrid approach combining BM25 and embeddings often gives the best performance!

6. Metrics for Retrieval Performance Evaluation

Evaluating retrieval performance is essential for understanding how well a search system ranks relevant documents. The most commonly used retrieval evaluation metrics are:

Metric Formulas and Intuition

1. Precision (How accurate are the retrieved results?)

Precision=Relevant Documents RetrievedTotal Documents Retrieved\text{Precision} = \frac{\text{Relevant Documents Retrieved}}{\text{Total Documents Retrieved}}

✅ High precision means fewer false positives (irrelevant results).
❌ Issue: If the system retrieves only one highly relevant document, precision is high but recall is low.

2. Recall (Did we retrieve all the relevant documents?)

Recall=Relevant Documents RetrievedTotal Relevant Documents in Dataset\text{Recall} = \frac{\text{Relevant Documents Retrieved}}{\text{Total Relevant Documents in Dataset}}

✅ High recall ensures most relevant documents are retrieved.
❌ Issue: If the system retrieves too many documents, recall is high but precision suffers.

🔹 Tradeoff: Precision-Recall balance (F1-score is often used as a compromise).

3. NDCG (Normalized Discounted Cumulative Gain)

NDCG=DCGIDCGNDCG = \frac{DCG}{IDCG}

Where DCG (Discounted Cumulative Gain) is:

DCG=∑i=1prelevanceilog2(i+1)DCG = \sum_{i=1}^{p} \frac{\text{relevance}_i}{\log_2(i+1)}

✅ Gives higher weight to top-ranked relevant results.
✅ Useful when ranking order matters (e.g., Google Search).

4. MAP (Mean Average Precision)

MAP=1N∑q=1NAPq\text{MAP} = \frac{1}{N} \sum_{q=1}^{N} AP_q

Where AP (Average Precision) is:

AP=1R∑k=1nP(k)×rel(k)AP = \frac{1}{R} \sum_{k=1}^{n} P(k) \times \text{rel}(k)

✅ Useful for evaluating ranking across multiple queries.
✅ Ideal for multi-query search scenarios (e.g., FAQ search systems).

5. MRR (Mean Reciprocal Rank)

MRR=1N∑i=1N1rank of first relevant documentMRR = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\text{rank of first relevant document}}

✅ Measures how quickly the system retrieves the first relevant document.
✅ Useful for search applications where finding the first match is important (e.g., voice assistants, chatbot responses).

7. Evaluating the Quality of Embeddings for Semantic Retrieval

Evaluating embedding-based retrieval requires checking how well similar documents are ranked.

1. Intrinsic Evaluation

Cosine Similarity: Ensures similar words/documents have embeddings that are closer in vector space.

t-SNE or PCA Visualization: Helps visualize whether semantically similar words cluster together.

MTEB Benchmark (Massive Text Embedding Benchmark): Standard benchmark to test embeddings across retrieval, classification, clustering.

2. Extrinsic Evaluation

Retrieval Metrics: Use Precision, Recall, NDCG, MAP, and MRR to evaluate search results.

Downstream Task Performance: Use embeddings in search, clustering, classification tasks and measure task accuracy.

✅ Best embeddings should maximize retrieval performance and semantic accuracy.

8. Evaluating the Quality of a RAG System

A RAG (Retrieval-Augmented Generation) system combines retrieval and generation. Evaluation must measure both components.

1. Retrieval Evaluation

Precision, Recall, NDCG, MRR, MAP: Check how well the retriever selects relevant documents.

2. Generation Evaluation

Faithfulness: Does the output factually match retrieved documents? (Avoids hallucination).

Relevance: Does the answer actually answer the query?

Fluency: Is the response coherent and natural?

BLEU/ROUGE Scores: Measures overlap between AI-generated text and expected answers.

✅ A great RAG system should retrieve the best context AND generate factually correct responses.

9. How Hybrid Search in RAG Retrieval Works

Hybrid Search combines term-based retrieval (BM25) and semantic search (embedding-based retrieval) for higher accuracy.

How Hybrid Search Works:

1️⃣BM25 Search: Retrieves documents with exact term matches.
2️⃣Embedding-Based Search: Retrieves documents with semantic similarity.
3️⃣Result Fusion: Merges both results using re-ranking or Reciprocal Rank Fusion (RRF).

✅ Advantages of Hybrid Search

Improves recall: Embedding search helps find synonyms and reworded phrases.

Improves precision: BM25 ensures important keywords are matched.

🔹 Example Use Case

Query: “How to fix my laptop screen?”

BM25 Match: Documents with “fix laptop screen”.

Embedding Match: Documents with “repair broken display”.

Final Results: BM25 + Embedding combined and re-ranked.

10. Describe the RRF (Reciprocal Rank Fusion) Algorithm

Reciprocal Rank Fusion (RRF) is a ranking algorithm that combines multiple ranking lists into a final ranked list.

How RRF Works:

Each retrieved document is assigned a reciprocal rank score:

Score(D)=∑i=1n1k+ri(D)\text{Score}(D) = \sum_{i=1}^{n} \frac{1}{k + r_i(D)}

Where:

n = number of ranking lists.

r_i(D) = rank of document D in ranking list i.

k = a constant to avoid division by zero (typically k = 60).

The final score of a document is the sum of reciprocal ranks across ranking lists.

Example:

Final Ranking:
1️⃣D2 (1.50)
2️⃣D1 (1.33)
3️⃣D3 (0.83)

✅ Why Use RRF?

Simple and parameter-free.

Balances term-based and embedding-based retrieval.

Works well when merging different retrieval methods.

Conclusion

Retrieval Evaluation: Use Precision, Recall, NDCG, MAP, MRR.

Embedding Evaluation: Use cosine similarity, MTEB, downstream task accuracy.

RAG System Evaluation: Test retrieval quality + generation faithfulness.

Hybrid Search: Combines BM25 and embeddings for better recall and precision.

RRF Algorithm: Fusion ranking method to combine multiple retrieval rankings.

🚀 Best Practice: Hybrid Search + RRF = Best RAG Retrieval Performance!

11. How Reranking Works

Reranking is a process that reorders retrieved documents to improve relevance and ranking accuracy before passing them to the generative model in RAG.

How Reranking Works:

Initial Retrieval: The system retrieves candidate documents using BM25, embeddings, or hybrid search.

Scoring and Sorting: The retrieved documents are scored based on additional criteria.

Final Ranking: The top-ranked documents are sent to the language model for response generation.

✅ Why Use Reranking?

Ensures most relevant documents are placed higher in ranking.

Improves accuracy while keeping retrieval efficient.

Reduces hallucination by filtering less relevant context.

Types of Reranking

1Context-Based Reranking

Uses the query context to reorder retrieved results.

Example:

Query: "What is Tesla's latest revenue?"

Initial results: Tesla's 2022, 2021, 2023 revenue reports.

Context-aware reranking places 2023 revenue at the top.

✅ When to Use:

In Q&A systems where context precision matters.

To boost relevance in conversational search.

2Time-Based Reranking

Prioritizes newer documents when time-sensitive information is needed.

Example:

Query: "Latest NVIDIA stock performance?"

Initial results: Stock reports from 2021, 2022, 2023, 2024.

Time-based reranking places 2024 results at the top.

✅ When to Use:

News search, financial analysis, real-time monitoring.

🔹 Combination Approach:
Use context-based + time-based reranking together for better ranking accuracy.

12. How Query Rewriting Works

Query rewriting improves retrieval accuracy by rephrasing or expanding the query to make it clearer.

How Query Rewriting Works:

Rephrasing: Rewrites vague or ambiguous queries.

Example:

Originalr: "How about Apple?"

Rewritten: "What is Apple's latest stock price?"

Query Expansion: Adds synonyms, context, or additional terms.

Example:

Original: "How to fix my PC?"

Expanded: "How to fix my Windows 10 computer?"

Disambiguation: Clarifies user intent.

Example:

Query: "Python tutorial"

Disambiguated: "Python programming tutorial (not the snake)."

✅ Why Use Query Rewriting?

Enhances retrieval recall.

Ensures contextually relevant results.

Improves conversational AI interactions.

13. How to Use Metadata Filtering for Contextual Retrieval

Metadata filtering improves retrieval precision by using structured metadata to filter irrelevant documents.

How Metadata Filtering Works:

Index metadata: Attach metadata like date, author, category, location to documents.

Query Filtering: Use metadata conditions to narrow down retrieval results.

Example Use Cases

✅ Why Use Metadata Filtering?

Reduces noise in retrieval results.

Ensures only contextually relevant documents are used.

Speeds up search performance.

🔹 Example: Hybrid Search + Metadata Filtering

Step 1: BM25 & Embeddings retrieve relevant documents.

Step 2: Metadata filtering removes outdated/irrelevant results.

Step 3: Final refined documents are ranked and used in response generation.

14. What is a Contextualized Chunk in RAG?

A contextualized chunk is a document fragment that includes additional context to improve retrieval accuracy.

How Contextualized Chunking Works:

Splitting: Documents are split into smaller, manageable chunks.

Augmenting Context:

Each chunk includes surrounding context (title, section heading).

Example:

Original Chunk: "Revenue in 2023 was $10M."

Contextualized Chunk: "Tesla's revenue in 2023 was $10M."

Indexing: The enriched chunk is stored in a vector database for retrieval.

✅ Why Use Contextualized Chunking?

Prevents loss of key information.

Improves search accuracy by ensuring retrieved chunks contain useful context.

How to Implement Contextualized Chunking

1️⃣Use Hierarchical Chunking

First split into sections (e.g., chapters, subheadings).

Then split into paragraphs.

2️⃣Add Metadata to Chunks

Store titles, headers, timestamps with each chunk.

3️⃣Use Context-Aware Retrieval

Use Hybrid Search (BM25 + embeddings) for better ranking.

🔹 Example Implementation in Python

```
def contextualize_chunk(document, chunk):
    return f"Title: {document['title']} | Section: {document['section']} | Content: {chunk}"
```

15. Key Considerations for Implementing a RAG-Based Retrieval Solution

A successful RAG system depends on multiple optimization factors.

1️⃣Retrieval Strategy

Term-Based Retrieval: BM25 for fast keyword matching.

Embedding-Based Retrieval: Vector search for semantic matching.

Hybrid Search: Combines both for best results.

✅ Best Practice: Use Hybrid Search + Reranking.

2️⃣Chunking Strategy

Fixed-size chunks (e.g., 512 tokens).

Hierarchical chunking (sections → paragraphs).

Overlapping chunks (to maintain context).

✅ Best Practice: Contextualized Chunking improves relevance.

3️⃣Reranking & Filtering

Context-Based Reranking: Prioritize most relevant documents.

Time-Based Reranking: Prioritize newer data for time-sensitive queries.

Metadata Filtering: Remove irrelevant/outdated documents.

✅ Best Practice: Fine-tune Reranking Models for domain-specific search.

4️⃣Scalability & Performance

Use FAISS, Milvus, or PGVector for efficient vector search.

Sharding & replication for distributed search scalability.

Cache frequently used embeddings.

✅ Best Practice: Optimize query latency & storage efficiency.

5️⃣Evaluation Metrics

Retrieval Performance: Use Precision, Recall, NDCG, MRR, MAP.

Generation Performance: Evaluate faithfulness, fluency, and relevance.

✅ Best Practice: Continuously evaluate & optimize using real user queries.

Conclusion

Reranking: Improves retrieval ranking (context-based, time-based).

Query Rewriting: Refines ambiguous queries.

Metadata Filtering: Ensures contextually relevant retrieval.

Contextualized Chunking: Enhances search accuracy.

Key Considerations for RAG: Use Hybrid Search, Chunking, Filtering, and Evaluation.

🚀 For the best RAG system, use a combination of Hybrid Search, Contextualized Chunking, Reranking, and Metadata Filtering!

1. How does Multi-Query Rewrite (MQR) work?

Multi-Query Rewrite (MQR) is a technique that enhances information retrieval by expanding the user's original query into multiple variations. This approach is widely used in Retrieval-Augmented Generation (RAG) to ensure that the system retrieves the most comprehensive and high-quality context possible.

How Multi-Query Rewrite Works

Parse the user query: The system receives the initial query and analyzes its intent and keywords.

Generate multiple query variations:

Synonym expansion: Replacing query keywords with synonyms or related terms.

Original query: "Latest product from Apple"

Expanded queries: "Newest Apple device", "Recently launched Apple products"

Sub-query generation: Breaking a complex query into multiple focused queries.

Original query: "Financial performance of Apple and Microsoft"

Sub-queries:

"Apple's financial performance"

"Microsoft's financial performance"

Different phrasings: Reformulating the query in various ways.

"What has Apple released recently?"

"What are the latest Apple product launches?"

Parallel execution of multiple queries: The generated queries are sent to the search engine, vector database, or retriever, retrieving a broader set of relevant documents.

Merge and deduplicate results: The system consolidates results from all queries, removes duplicates, and ranks documents based on relevance.

Benefits of Multi-Query Rewrite

Improves recall: Reduces the chance of missing relevant documents.

Enhances query understanding: Helps retrieve the most meaningful information even when the user's query is vague.

Boosts RAG performance: A richer document set improves the quality of responses generated by the model, reducing hallucinations.

2. How does RAG-Fusion work in query rewriting?

RAG-Fusion is an advanced retrieval strategy that builds on Multi-Query Rewrite (MQR) by integrating ranking fusion techniques to improve the selection of relevant documents. It ensures that retrieved information is both comprehensive and highly relevant while minimizing redundancy.

How RAG-Fusion Works

Multi-query generation (similar to MQR):

Generates different versions of the query:

"Latest Apple product"

"Newest Apple device"

"What did Apple release this year?"

Parallel retrieval across multiple retrieval methods:

Uses both keyword-based retrieval (BM25) and vector-based retrieval (HNSW, IVFFlat).

Queries multiple sources (structured databases, embeddings, search engines).

Reciprocal Rank Fusion (RRF) for result ranking:

This is more like an algorithm to reduce the weight for top rank criteria to avoid focus too much on the top criteria, by increase the weight of other lower rank criteria to let ai utilize other lower ranked criteria.

Contextual chunking for optimized document retrieval:

Instead of retrieving entire documents, RAG-Fusion extracts the most relevant paragraphs (contextual chunking), ensuring the model processes only the necessary information.

Refined context input to LLM:

The most relevant documents (or document chunks) are passed to the Large Language Model (LLM) for answer generation.

Advantages of RAG-Fusion

More precise retrieval: Filters out irrelevant documents, making retrieval more accurate than traditional RAG.

Higher recall: Expands query scope while refining document selection.

Reduces hallucinations: By providing richer and more relevant context, it minimizes AI-generated misinformation.

Use Cases

Legal & medical applications: Where accuracy is critical, and missing context could lead to incorrect conclusions.

Finance & market research: To ensure comprehensive data retrieval from various sources.

Enterprise knowledge retrieval: Optimizes search accuracy in corporate databases.

Summary

Multi-Query Rewrite (MQR) improves retrieval by generating multiple variations of a query, ensuring a broader and more accurate document set.

RAG-Fusion enhances retrieval by combining multi-query rewriting with fusion-based ranking (RRF) and contextual chunking, leading to better document selection and AI-generated responses.

reading guide week 8.docx

Huyen 2024: Chapter 6: Agents, Memory

✅ Chapter 6: Agents & Memory — Answers Based on Your Translation

1. How is an AI agent defined in the book? What are the core components of an AI agent?

An AI agent is defined as anything that can perceive its environment and act upon that environment.
The core components are:

The environment it operates in

The set of actions it can perform, which depends on the tools it has access to

📌 "An agent is anything that can perceive its environment and act upon that environment… characterized by the environment it operates in and the set of actions it can perform."

2. Why do agents require more powerful models?

Because:

Compound mistakes: The more steps an agent takes, the higher the cumulative error rate. (e.g., 95% accuracy per step drops to 60% after 10 steps, 0.6% after 100 steps)

Higher stakes: Tool access gives agents the power to do impactful tasks, which raises the risk of serious failure.

📌 "Compared to non-agent use cases, agents typically require more powerful models for two reasons: compound mistakes and higher stakes."

3 & 5. What are knowledge augmentation tools for AI agents? Provide an example.

These tools help agents gain access to relevant external knowledge, especially organizational or up-to-date information.
Examples:

Text retriever, image retriever, SQL executor,

Web browsing, Slack/email reader, Inventory API

📌 "An important category of tools includes those that help augment your agent's knowledge… web browsing prevents a model from going stale."

4. What are capability extension tools for AI agents? Provide an example.

These tools overcome the inherent limitations of AI models, like poor math or code execution.
Examples:

Calculator

Code interpreter

Calendar, unit converter, LaTeX renderer

📌 "Instead of trying to train the model to be good at arithmetic, it's more resource-efficient to just give the model access to a tool… code interpreters… translator."

6. What are tools that allow AI agents to act upon their environment? Provide an example.

These are write tools that let agents make changes.
Examples:

SQL executor (can modify or delete a table)

Email API (can send responses)

Banking API (can initiate a bank transfer)

📌 "Write actions enable a system to do more… a SQL executor can retrieve a data table but can also change or delete the table."

7. What are the key stages of planning when designing an AI agent?

Plan generation – Break the task into manageable steps

Reflection and error correction – Check if the plan makes sense

Execution – Carry out the steps using tools

Post-execution reflection – Evaluate results and replan if needed

📌 "Solving a task typically involves: Plan generation → Reflection → Execution → Reflection again."

8. Why should planning be decoupled from execution?

To avoid executing invalid or useless plans.
Separate validation allows you to:

Check for tool availability

Enforce step limits

Prevent wasteful or harmful actions

📌 “To avoid fruitless execution, planning should be decoupled from execution… only after the plan is validated is it executed.”

9. What are some different approaches to improve an agent’s planning capabilities?

Write better prompts with examples

Improve tool descriptions

Simplify tool interfaces

Use stronger models

Finetune on planning tasks

Plan using natural language

Use intent classifiers and evaluators

📌 “A few approaches: write better prompts, describe tools more clearly, refactor functions, use stronger models, or finetune.”

10. Describe different orders in which a plan can be executed.

Sequential – Each step depends on the previous

Parallel – Multiple steps at once

If statement – Branching based on previous result

For loop – Repeat until a condition is met

📌 “The order in which actions can be executed is called a control flow… sequential, parallel, if statement, and for loop.”

11. How does a reflection agent work? What are strengths and weaknesses of this approach?

A reflection agent evaluates plans and results, then improves them.
Strengths:

Higher success rate

Can learn from failures
Weaknesses:

Higher token cost

Increased latency

📌 "Reflection… generates insights that help uncover errors… Reflexion separates evaluation and self-reflection… downside is latency and cost."

12. What factors should be considered when selecting tools for an AI agent?

Task/environment fit

Model's ability to use the tool

Tool complexity

Frequency of use

Prompting difficulty

Security risks

Cost of integration

📌 "Tool selection requires careful consideration… performance drops if a tool is removed… test tools independently."

13. What are different agent failure modes? How would you evaluate an agent for planning failures?

Failure types:

Planning failure: invalid tool/parameters, unmet goals

Tool failure: wrong outputs or inaccessible tools

Efficiency failure: too many steps or time taken
Evaluation:

% of valid plans

Avg. number of attempts to get a valid plan

Invalid tool/param usage rate

📌 "Planning is hard and can fail in many ways… create a dataset… compute metrics like tool validity and failure frequency."

14. How would you evaluate an agent's efficiency?

Track:

Average steps per task

Time per action

Cost per task

Most/least efficient actions
Compare against Human or other AI baselines

📌 "Evaluate an agent's efficiency: how many steps… how long each action takes… compare with baseline agents."

15. How are memories used by an agent?

To store and recall:

Instructions

Context and user preferences

Plans, tool outputs, reflections
Memory helps across multi-step and multi-session tasks.

📌 "An agentic system needs memory to store instructions, examples, context, tool inventories, plans, tool outputs, reflections…"

16. What are the benefits of augmenting an AI model with a memory system?

Handle context overflow

Personalize behavior across sessions

Improve consistency

Store structured data (e.g., tables, queues)

📌 "Many benefits… manage overflow, persist info between sessions, boost consistency, preserve structured integrity…"

17. How does memory management work for AI models?

Involves:

Adding or deleting info from short-/long-term memory

Short-term memory has limited space (context window)

Use strategies like:

FIFO

Summarization

Reflection-based update

Redundancy removal

📌 "Memory management typically consists of two operations: add and delete… FIFO is simple but risky… summarization and reflection help."

Singh et al. 2025:

1. What are the core components of an AI Agent?

The core components of an AI agent in Agentic RAG are:

LLM (with defined Role and Task): The central reasoning engine and dialogue interface.

Memory:

Short-Term Memory: Tracks the current conversation state.

Long-Term Memory: Stores accumulated knowledge and experience.

Planning (Reflection & Self-Critique): Breaks down complex tasks via reflection and strategic planning.

Tools: Accesses external resources like vector search, web search, APIs, etc., to extend capabilities.

2. What are the agentic workflow patterns presented in the paper? Describe each agentic workflow pattern in simple language.

Prompt Chaining:

Breaks a big task into smaller steps.

Each step builds on the previous one.

Good for tasks that benefit from step-by-step logic.

Routing:

Sorts queries and sends them to the right processing path.

Useful when different types of queries need different treatments.

Parallelization:

Splits tasks into parts that run at the same time.

Boosts speed and handles large-scale processing.

Orchestrator-Worker:

A main agent assigns subtasks to worker agents.

Dynamic and adapts in real time.

Evaluator-Optimizer:

Generates an initial result, then evaluates and refines it.

Ideal when quality can improve through feedback cycles.

3. Seven Agentic RAG architectural frameworks: characteristics, strengths, and limitations

a. Single-agent Agentic RAG

Characteristics: One agent controls everything—retrieves, selects tools, and integrates data.

Strengths: Simple, efficient, easy to manage.

Limitations: Limited scalability; not ideal for complex tasks.

b. Multi-agent Agentic RAG

Characteristics: Multiple agents, each specialized (e.g., SQL, web search, etc.).

Strengths: Modular, scalable, more accurate due to specialization.

Limitations: Coordination is complex and computationally expensive.

c. Hierarchical Agentic RAG

Characteristics: Agents are structured in layers—top-level agent assigns tasks to sub-agents.

Strengths: Strategic prioritization, good for complex decision-making.

Limitations: Higher orchestration complexity and risk of bottlenecks.

d. Agentic Corrective RAG

Characteristics: Agents evaluate and correct errors in retrieved data.

Strengths: Improves factuality, relevance, and accuracy through iteration.

Limitations: Adds layers of complexity and can be resource-intensive.

e. Adaptive Agentic RAG

Characteristics: Uses a classifier to choose how complex the query is and adjusts accordingly.

Strengths: Resource-efficient, dynamically adapts to different query types.

Limitations: Requires well-trained classifiers; may misclassify complex queries.

f. Graph-based Agentic RAG

Characteristics: Integrates graph data and unstructured text; uses critic modules and feedback loops.

Strengths: Great for reasoning over relationships (e.g., healthcare, law).

Limitations: Depends on high-quality graph data; complex integration.

g. Agentic Document Workflow (ADW)

Characteristics: Automates document-centric tasks end-to-end (parse, retrieve, reason, output).

Strengths: Maintains state, supports multi-step workflows, applies business rules.

Limitations: High domain-specific setup cost; resource-intensive.

4. What are the differences among traditional RAG, agentic RAG, and Agentic Document Workflow?

reading guide week 9.docx

Evaluation-Driven Development: Answers Based on Provided Materials

1. What is evaluation-driven development?

Evaluation-driven development is the approach where application development is guided by systematic evaluation. It emphasizes having “an evaluation pipeline that you can rely upon,” because “not having a reliable evaluation pipeline is one of the biggest blocks to AI adoption.”

2. The author provides categories of evaluation criteria. What are they? Explain each in your own words.

The main categories include:
- Domain-specific capabilities: Tasks the model is supposed to be good at, like code generation or language translation.
- Generation capabilities: Whether the output is fluent, coherent, and factually correct.
- Factual consistency: Whether outputs align with known or provided facts.
- Safety: Ensuring outputs are not harmful, biased, or toxic.
- Instruction-following: Whether the model adheres to given instructions, formats, or content guidelines.
- Cost and latency: Whether the model is fast and affordable enough to be useful.
- Roleplaying: Whether the model stays in character for interactive use cases.

3. Based on the application domain for your GenAI project, what domain-specific capability criteria might you use to evaluate it?

If building a writing assistant, criteria might include “creative writing benchmarks” and evaluating for “relevance,” “fluency,” and “coherence.” If it’s a coding agent, criteria would include “code generation,” “functional correctness,” and “efficiency.”

4. Similarly, what general capability criteria might you use for your application?

Examples include:
- Factual consistency: Using AI judges or entailment models to verify outputs.
- Safety: Avoiding toxic or biased outputs.
- Instruction-following: Whether the model can produce outputs following an expected format.
- Latency/Cost: Measuring “time to first token” and “cost per output token.”
- Fluency/Coherence: Measured with “perplexity” or AI judges.

5. What are the differences between local and global factual consistency? How would you evaluate each type?

Local: “The output is evaluated against a context.”
Global: “The output is evaluated against open knowledge.”

Local consistency is easier to evaluate using context provided by a user or retrieval step. Global consistency may involve search and fact-checking.

6. How does self-verification work in evaluation? Would you incorporate it into your application? If so, how?

Self-verification uses “SelfCheckGPT,” which “generates N new responses and measures how consistent [they are] with respect to [the original].” It’s useful for hallucination detection but is “prohibitively expensive.” Incorporating it could help improve reliability for critical outputs.

7. How does SAFE (Search-Augmented Factuality Evaluator) work? Would you incorporate it into your application? If so, how?

SAFE “leverages search engine results” by:
1. Decomposing a response into statements
2. Making each self-contained
3. Proposing fact-checking queries
4. Using AI to evaluate consistency

Yes, this would be useful for validating facts in customer support or research tools.

8. How does textual entailment work? Would you incorporate it into your application? If so, how?

It determines whether a hypothesis is entailed, contradicted, or neutral to a premise. “Entailment implies factual consistency.” It’s implemented with classifiers like “DeBERTa-v3-base-mnli-fever-anli.” Yes, it can be used to verify summary consistency or QA answers.

1. What kind of user feedback would be useful for evaluating or improving your application?

Both explicit and implicit user feedback are useful:

Explicit feedback includes thumbs up/down, star ratings, upvotes/downvotes, or direct answers to questions like “Did we solve your problem?” These help with evaluation and fine-tuning models based on clearly expressed user opinions.

Implicit feedback comes from user actions such as:

Regenerating responses

Editing model outputs (especially code or text)

Early termination of conversations

Rephrasing queries

Asking for confirmation (“Are you sure?” or “Check again”)

Sharing, bookmarking, or deleting conversations

These types of feedback can be used for:

Evaluation: monitoring model quality and application performance

Development: guiding future model iterations

Personalization: customizing the app for individual user preferences

Other valuable signals include:

Complaints about hallucination, verbosity, or toxicity

Sentiment analysis from conversations

Output token length, turn count, and topic diversity

2. When and how would you collect user feedback for your application?

You should collect feedback throughout the user journey—but non-intrusively, so it doesn't disrupt the user experience.

When:

At onboarding (e.g., calibrating user preferences or skill level)

When something goes wrong (e.g., hallucination, slow response, or rejection of valid input)

When model confidence is low (e.g., show two response options for users to choose from)

When something goes well (e.g., allow optional positive feedback)

How:

Allow in-flow options like thumbs up/down, edit, or regenerate

Offer simple UI interactions (e.g., like Midjourney’s U/V buttons or GitHub Copilot’s Tab-to-accept)

Use conversational language as implicit signals (e.g., corrections, preferences)

Collect session metadata and conversational logs (with privacy considerations)

Optionally request user consent to include surrounding context for deeper analysis

Comparative feedback, sentiment cues, and conversation organization (e.g., renaming or deleting) also serve as important feedback mechanisms.

3. What are some limitations of user feedback?

User feedback has several limitations:

Biases:

Leniency bias: Users rate positively to avoid effort or confrontation.

Position bias: Users tend to select the first presented option.

Preference bias: Users might prefer longer answers or recent responses, even if they are worse.

Response bias: Negative feedback tends to be overrepresented.

Noise and randomness:

Users might click randomly when comparing multiple options.

Sparse or inconsistent responses from low-effort users.

Interpretation ambiguity:

Implicit actions like sharing or deleting may have multiple meanings (e.g., frustration vs. value).

Requires deep understanding of user intent.

Degenerate feedback loops:

Models might overfit to what users "like" (e.g., cute cat photos), reinforcing biases.

Can lead to echo chambers or "sycophancy" (models telling users what they want to hear rather than what's accurate).

Privacy risks:

Feedback may include sensitive or personal information.

Logs and context for feedback need proper handling and consent.

In short, while user feedback is essential for AI application development, it must be interpreted carefully and used responsibly to avoid reinforcing poor behaviors or misjudging user needs.

reading guide week 10.docx

1. What is Responsible AI, and why do we need it?

“Responsible AI refers to the ethical and accountable development, deployment, and use of AI systems. It involves ensuring fairness, transparency, privacy, and avoiding biases in AI algorithms.” (Responsible AI Chapter)

We need it because:

“LLMs open the doors to a new set of risks and biases… to mitigate them with defensive attacks.”
“Responsible AI plays a crucial role in steering decisions toward positive and fair results.”

Risks include hallucinations, harmful content, and discrimination. Responsible AI aims to protect fundamental rights, ensure trust, and promote human-centric outcomes.

2. What ethical implications of Responsible AI do you foresee in your Generative AI application?

“Bias: AI systems can inherit biases present in their training data... leading to discriminatory outcomes.”
“Explainability: Black-box models (such as LLMs) lack interpretability...”
“Data protection… Consent, anonymization, and data minimization principles should guide AI development.”
“Human oversight… especially in high-stakes contexts.”
“Security: Ensuring AI systems are secure and resistant to attacks is crucial.”
“Environmental impact: Training large models consumes significant energy.”

These implications directly affect generative AI applications, especially when used for public-facing tasks or content creation.

3. How would you ensure responsible AI development at the model level?

“If the training data is biased, the model will inherit a biased vision of the world.”
“Redact and curate training data… carefully select the training data.”
“Use reinforcement learning with human feedback (RLHF)... pivotal in making it less biased.”
“OpenAI employs this strategy to avoid language models generating harmful or toxic content.”
“Toolkits like the Responsible AI Toolbox... help developers incorporate Responsible AI practices into their workflows.”

Also:

“Provide metrics to assess group fairness and tools to mitigate identified biases.”

4. How would you ensure responsible AI development at the Metaprompt level?

“The meta prompt… is a key component to make our LLM-powered application successful.”
“Providing clear instructions and guidelines to the AI model about what it can and cannot do.”
“Implementing grounding strategies… can ensure the model does not hallucinate or provide harmful information.”
“Include some defensive techniques to prevent… prompt injection.”

Specific mitigation:

“Adversarial Prompt Detector… enforces the desired behavior through the instruction given to the model.”

5. What is prompt injection? How would you protect your system against it?

“Prompt injection stands as a form of attack on LLMs... deceived by adversarial user input.”
Prompt leakage: “An attacker could leak the prompt and change it...”
Goal hijacking: “Prompts that have been tested as capable of jailbreaking the metaprompt... e.g., DAN (Do Anything Now).”

Prevention:

“Include some defensive techniques… Adversarial Prompt Detector.”
“Enforcing the desired behavior through the instruction given to the model.”

6. How would you ensure responsible AI development at the user interface level?

“The user interface represents the last mile... to mitigate the potential associated risks.”
“Disclose the LLM’s role in the interaction.”
“Cite references and sources… such as context from a vector DB.”
“Show the reasoning process… SQL query run against the provided database.”
“Show the tools used… make sure the model uses them properly.”
“Prepare pre-defined questions… give a better UX to the user.”
“Provide system documentation… covering the system’s capabilities, constraints, and risks.”
“Publish user guidelines and best practices… integrate these directly into the UX.”

Also:

"It is important to establish a systematic approach to assess the effectiveness of implemented mitigations."

7. What additional considerations can you think of to ensure responsible AI development that are not covered in the readings?

While not explicitly stated, the materials hint that Responsible AI is:

"An evolving field of research, and it definitely has an interdisciplinary flavor."

So, considerations like cross-cultural fairness, accessibility, and user education may be natural //extensions of the core principles in the readings, though they are not directly mentioned.

8. Discuss the AI Act by the European Commission – what are the stakeholders and what are the objectives?

Stakeholders:

"Providers… Users… Importers… Distributors… Third-country entities."

Objectives:

"Categorizing AI systems by risk… promote human-centric and trustworthy AI."
"Safeguard health, safety, fundamental rights, democracy, the rule of law, and the environment."
"Establishes an EU AI Office… mandates member states to appoint national supervisory authorities for AI."
"Providers of generative AI… must train, design, and develop their systems with state-of-the-art safeguards…"
"Must document and publicly disclose use of copyrighted data… label deepfakes."

Progress:

"On June 14, 2023, the European Parliament had endorsed its stance on the AI Act…"
"Approved in December 2023… grace period of 2 to 3 years for preparation."
"Positioning the EU as a potential trailblazer in oversight of generative AI."