AI Fundamentals & Prompt Engineering

Interview Preparation Guide

Prepared for: Ravi Kiran
Date: October 09, 2025

This guide covers fundamental AI concepts, GenAI/RAG/LLM technologies, and practical prompt engineering scenarios aligned with the job requirements for AI-integrated software development roles.

Section 1: Basic AI Concepts and Foundations

Q1: What is the difference between AI, Machine Learning, and Deep Learning?

Answer:

- **AI (Artificial Intelligence):** Broadest concept - any system that performs tasks requiring human intelligence. Includes rule-based systems, expert systems, and ML.
- **Machine Learning:** Subset of AI where systems learn patterns from data without explicit programming. Uses algorithms like decision trees, SVM, random forests.
- **Deep Learning:** Subset of ML using neural networks with multiple layers (deep). Excels at unstructured data like images, text, audio.

Practical Example from MessageWise: • AI: The entire system making intelligent support decisions • ML: Using patterns in past tickets to predict resolution paths • Deep Learning: BERT/Transformer models creating embeddings for semantic search

Q2: Explain supervised vs unsupervised vs reinforcement learning.

Answer:

- **Supervised Learning:** Training with labeled data (input-output pairs). Used for classification and regression. Example: Spam detection with labeled emails.
- **Unsupervised Learning:** Finding patterns in unlabeled data. Used for clustering and dimensionality reduction. Example: Customer segmentation.
- **Reinforcement Learning:** Learning through trial and error with rewards/penalties. Example: Game playing, robotics.

In MessageWise Context: • Supervised: Training classifiers for ticket urgency using labeled historical tickets • Unsupervised: Clustering similar error messages without labels • Reinforcement: Could optimize response ranking based on user feedback signals

Q3: What metrics would you use to evaluate an AI model?

Answer: Depends on the task type:

Classification Metrics:

- **Accuracy:** Overall correct predictions. Can be misleading with imbalanced data.
- **Precision:** Of predicted positives, how many were correct? Important when false positives are costly.
- **Recall:** Of actual positives, how many did we catch? Important when false negatives are costly.
- **F1 Score:** Harmonic mean of precision and recall. Good for balanced evaluation.
- **AUC-ROC:** Performance across different thresholds.

For RAG/LLM Systems (like MessageWise):

- **Relevance:** Are retrieved documents relevant to the query?
- **Groundedness:** Are answers supported by retrieved documents?
- **Latency:** Response time (p50, p95, p99)
- **User Satisfaction:** Thumbs up/down, resolution rate
- **Citation Accuracy:** Do citations actually support claims?

Section 2: GenAI, RAG, and LLM Concepts

Q4: What is Generative AI and how does it differ from traditional AI?

Answer:

- **Traditional AI:** Analyzes, classifies, or predicts based on input. Output is typically a label, number, or decision.
- **Generative AI:** Creates new content (text, images, code) that didn't exist before. Output is novel content similar to training data.

Examples: • Traditional: Classify email as spam/not spam, predict house prices • Generative: Write email responses, create marketing copy, generate code **Key Technologies:** • Transformers (GPT, BERT) • Diffusion Models (Stable Diffusion, DALL-E) • VAEs and GANs

Q5: Explain RAG (Retrieval-Augmented Generation). Why use it instead of just an LLM?

Answer: RAG combines information retrieval with language generation to ground LLM responses in factual, up-to-date information.

RAG Process:

1. **Query Processing:** User question is processed and sometimes expanded
2. **Retrieval:** Relevant documents fetched from knowledge base
3. **Context Assembly:** Retrieved docs + query combined into prompt
4. **Generation:** LLM generates answer based on retrieved context
5. **Citation:** Response includes references to source documents

Why RAG over pure LLM:

- **Accuracy:** Reduces hallucinations by grounding in real data
- **Updatable:** Knowledge base can be updated without retraining
- **Verifiable:** Provides citations for fact-checking
- **Domain-specific:** Can use proprietary/internal data
- **Cost-effective:** No need for expensive fine-tuning

Q6: What is an LLM? How do they work at a high level?

Answer: Large Language Models are neural networks trained on vast text data to predict the next word in a sequence.

Key Concepts:

- **Tokens:** Text broken into chunks (words or subwords)

- **Embeddings:** Tokens converted to numerical vectors
- **Attention Mechanism:** Model learns relationships between all tokens
- **Context Window:** Maximum tokens the model can process at once
- **Temperature:** Controls randomness (0=deterministic, 1=creative)

Common LLMs:

- **GPT Family:** OpenAI's models (GPT-3.5, GPT-4)
- **Claude:** Anthropic's models (Claude 2, Claude 3)
- **Open Source:** Llama 2, Mistral, Falcon
- **Specialized:** CodeLlama (code), BioGPT (biomedical)

Q7: When would you fine-tune an LLM vs using RAG vs prompt engineering?

Answer: Each approach has different use cases and trade-offs:

Prompt Engineering (First Choice):

- **When:** Task can be solved with good instructions and examples
- **Pros:** No training needed, immediate iteration, low cost
- **Cons:** Limited by context window, may need repeated examples
- **Example:** Creating email templates, classification with few categories

RAG (For Dynamic Knowledge):

- **When:** Need access to changing/proprietary information
- **Pros:** Up-to-date info, verifiable, no retraining
- **Cons:** Retrieval quality impacts results, added latency
- **Example:** MessageWise support system, documentation Q&A;

Fine-tuning (For Specialized Behavior):

- **When:** Need consistent style/format, domain-specific knowledge
- **Pros:** Better performance on specific tasks, faster inference
- **Cons:** Expensive, risk of catastrophic forgetting, needs lots of data
- **Example:** Medical diagnosis, legal document generation

Section 3: AI-Native Principles for Software Development

Q8: What does it mean to build 'AI-Native' applications?

Answer: AI-Native means designing applications with AI capabilities as core features from the start, not bolted on later.

Key Principles:

- **Probabilistic Thinking:** Design for uncertainty and confidence scores, not binary outcomes
- **Human-in-the-Loop:** AI assists but doesn't replace human judgment for critical decisions
- **Continuous Learning:** Systems improve from user feedback and new data
- **Graceful Degradation:** Fallbacks when AI fails or confidence is low
- **Explainability:** Users can understand and verify AI decisions

Implementation in MessageWise:

- Confidence scores on all responses
- Citations for verification
- Feedback loops for improvement
- Fallback to human experts when needed
- Audit logs for all decisions

Q9: How do you handle AI failures gracefully in production?

Answer: Multiple layers of defense and fallback strategies:

Technical Strategies:

- **Timeout Handling:** Set maximum response times with fallbacks
- **Confidence Thresholds:** Only show results above certain confidence
- **Circuit Breakers:** Temporarily disable failing services
- **Caching:** Serve cached similar responses during outages
- **Model Redundancy:** Multiple models (primary, secondary, fallback)

User Experience:

- Clear communication: 'I'm not confident about this answer'
- Provide alternatives: 'Here are related topics that might help'
- Enable escalation: Easy path to human expert
- Show working status: Loading indicators, partial results
- Request clarification: 'Could you rephrase your question?'

Section 4: Prompt Engineering Scenarios

Q10: You need to create a model that generates salad recipes based on available ingredients. How would you design the prompts?

Answer: I would use a structured, iterative approach with clear constraints and examples:

Initial System Prompt:

```
You are a professional chef specializing in healthy, creative salads.
Generate recipes that are: - Nutritionally balanced - Use only the provided
ingredients - Include preparation time and difficulty level - Suitable for
the specified dietary restrictions Output format: Recipe Name: [Creative
name] Prep Time: [X minutes] Servings: [Number] Difficulty:
[Easy/Medium/Hard] Ingredients: - [Ingredient 1 with amount] - [Ingredient 2
with amount] Instructions: 1. [Step 1] 2. [Step 2] Nutritional Info: [Brief
summary] Chef's Tip: [Optional suggestion]
```

User Prompt Template:

```
Create a salad recipe using these ingredients: Available: {user_ingredients}
Dietary restrictions: {restrictions} Preference: {taste_preference} Serving
size: {servings} Do not use any ingredients not listed above.
```

Improvement Strategies:

- **Few-shot examples:** Include 2-3 example recipes in the prompt
- **Negative examples:** Show what NOT to do (using unavailable ingredients)
- **Chain-of-thought:** Ask model to first list usable ingredients, then create recipe
- **Validation step:** Have model verify it only used available ingredients
- **Variations:** Request multiple options ranked by complexity

Q11: Design a prompt to extract key information from customer support emails.

Answer: Use structured extraction with clear field definitions:

Extraction Prompt:

```
Extract the following information from the customer email below. Return as
JSON with these exact fields: { "sentiment": "positive|negative|neutral",
"urgency": "high|medium|low", "category":
"technical|billing|feature_request|complaint|other", "key_issue": "one
sentence summary", "product_mentioned": ["list of products"],
"requested_action": "what customer wants done", "customer_emotion":
"frustrated|satisfied|confused|angry|neutral" } Rules: - Use null for missing
information - Keep summaries under 20 words - Extract exact product names
when mentioned - Base urgency on language and stated timelines Email:
{email_content} JSON Output:
```

Enhancement Techniques:

- **Examples in prompt:** Show 2-3 correctly extracted examples
- **Edge case handling:** Include examples with missing fields
- **Validation prompt:** Second pass to verify extraction accuracy
- **Confidence scoring:** Add confidence field for each extraction

Q12: Create a prompt for a code review assistant that provides constructive feedback.

Answer: Design prompts that are specific, constructive, and educational:

Code Review Prompt:

````
You are a senior developer conducting a code review. Analyze the following
code and provide feedback. Focus areas: 1. Bugs and potential errors 2.
Performance optimizations 3. Code readability and maintainability 4. Security
vulnerabilities 5. Best practices for {language} For each issue found: -
Severity: [Critical|High|Medium|Low] - Location: [Line numbers] - Issue:
[Brief description] - Suggestion: [How to fix] - Example: [Show corrected
code if applicable] Code to review: ```{language} {code} ``` Provide feedback
in this format: ### Summary [Overall assessment - be constructive] ###
Critical Issues [Must fix before deployment] ### Improvements [Suggested
enhancements] ### Good Practices Observed [Acknowledge what was done well]
````

Key Strategies:

- **Balanced feedback:** Include both improvements and positives
- **Actionable suggestions:** Don't just identify problems, show solutions
- **Severity levels:** Help prioritize what needs immediate attention
- **Learning opportunity:** Explain why something is an issue

Q13: How would you design prompts for a customer service chatbot that maintains context?

Answer: Use conversation history and state management:

Initial System Prompt:

```
You are a helpful customer service representative for Sur La Table. Your
capabilities: - Answer product questions - Help with order status - Process
returns/exchanges - Provide cooking class information - Escalate complex
issues Maintain context across the conversation: - Remember customer's name
and previous questions - Reference earlier topics naturally - Track
unresolved issues - Note customer preferences Always: - Be empathetic and
professional - Provide specific, actionable help - Admit when you need to
escalate - Confirm understanding of complex requests
```

Context Management Template:

```
Conversation History: {previous_messages} Customer Profile: - Name:
{customer_name} - Previous issues: {past_issues} - Current session topics:
{session_topics} Current Message: {user_message} Respond naturally while
maintaining context from the conversation. If referencing earlier topics, be
specific.
```

Context Preservation Strategies:

- **Summarization:** Periodically summarize long conversations
- **Key facts extraction:** Track important details (order numbers, issues)
- **Intent tracking:** Maintain list of unresolved customer needs
- **Emotion monitoring:** Adjust tone based on customer sentiment

Section 5: Practical AI Integration Questions

Q14: How do you ensure data quality for AI model training?

Answer: Data quality is critical for AI success. I use a systematic approach:

Data Quality Checklist:

- **Completeness:** Check for missing values, handle appropriately (impute, remove, or flag)
- **Consistency:** Standardize formats, units, naming conventions
- **Accuracy:** Validate against known sources, check for outliers
- **Relevance:** Ensure data matches the problem domain
- **Timeliness:** Use recent data that reflects current patterns
- **Balance:** Check class distributions, address imbalances

In MessageWise:

- Deduplicated documents to avoid retrieval bias
- Standardized formatting across different doc types
- Validated citations actually support claims
- Removed outdated runbooks
- Balanced training data across different incident types

Q15: How would you collaborate with data scientists as a software engineer on an AI project?

Answer: Effective collaboration requires clear communication and defined responsibilities:

Key Collaboration Areas:

- **Data Pipeline:** I build robust data ingestion and preprocessing systems based on their requirements
- **Model Deployment:** Convert research code to production-ready services (containerization, API design)
- **Monitoring:** Implement logging, metrics, and alerting for model performance
- **A/B Testing:** Build infrastructure for experimentation and gradual rollouts
- **Feedback Loops:** Create systems to collect user feedback for model improvement

Communication Best Practices:

- Regular sync meetings to align on requirements
- Clear documentation of APIs and data schemas
- Joint code reviews for model integration

- Shared metrics dashboards
- Collaborative debugging when issues arise

Q16: Describe a scenario where AI might not be the right solution.

Answer: AI isn't always the answer. Consider these scenarios:

When NOT to use AI:

- **Simple rules suffice:** If business logic is clear and deterministic (e.g., age verification)
- **Insufficient data:** Less than 1000 examples for supervised learning
- **Need 100% accuracy:** Critical safety systems, financial calculations
- **Regulatory constraints:** Some industries require explainable, auditable decisions
- **Cost exceeds benefit:** Simple lookup tables might work better than complex models

Real Example:

For MessageWise, we didn't use AI for: • User authentication (deterministic security rules) • Permission checks (clear role-based access) • Exact string matching for error codes • Calculating SLA metrics (precise formulas needed) These cases needed guaranteed correctness, not probabilistic answers.

Q17: How do you stay updated with rapidly evolving AI technology?

Answer: Continuous learning through multiple channels:

- **Practical experimentation:** Test new models/tools on side projects
- **Research papers:** Follow arXiv, read 1-2 papers monthly
- **Community engagement:** Reddit (r/MachineLearning), HackerNews, Twitter AI community
- **Courses:** FastAI, Coursera updates, YouTube tutorials
- **Conferences:** Watch keynotes from NeurIPS, ICML online
- **Tools/Frameworks:** Follow releases from OpenAI, Anthropic, HuggingFace
- **Hands-on:** Contribute to open-source AI projects

Section 6: Advanced Prompt Engineering Scenarios

Q18: Design a prompt system for generating personalized workout plans.

Answer: Create a multi-step prompt system with safety considerations:

Step 1: Information Gathering Prompt

```
Extract user fitness profile from their input: { "fitness_level":
"beginner|intermediate|advanced", "goals": ["weight_loss", "muscle_gain",
"endurance", "flexibility"], "available_equipment": ["list"],
"time_per_session": "minutes", "sessions_per_week": "number", "restrictions":
["injuries", "medical_conditions"], "preferences": ["liked_exercises",
"disliked_exercises"] }
```

Step 2: Workout Generation Prompt

```
Create a personalized workout plan: User Profile: {extracted_profile}
Generate a {sessions_per_week}-day workout plan with these requirements: -
Each session under {time_per_session} minutes - Progressive difficulty over 4
weeks - Use only {available_equipment} - Avoid exercises affecting
{restrictions} - Focus on {goals} Format each workout as: Day X: {Focus Area}
Warm-up (5 min): [exercises] Main Workout (X min): - Exercise name: Sets x
Reps (or time) - Rest periods Cool-down (5 min): [stretches] Include form
cues and modifications. Add safety disclaimer about consulting healthcare
providers.
```

Safety and Personalization Strategies:

- Always include medical disclaimer
- Provide easier/harder variations for each exercise
- Explain the 'why' behind exercise selection
- Include progress tracking metrics
- Add form videos/image references where applicable

Q19: Create a prompt for analyzing and summarizing legal documents (showing domain complexity).

Answer: Design careful prompts with appropriate disclaimers:

Legal Document Analysis Prompt:

```
You are a legal document analyzer (NOT providing legal advice). Analyze this
document and provide a structured summary. IMPORTANT DISCLAIMER: This is
informational analysis only, not legal advice. Users should consult qualified
attorneys. Document Type: {document_type} Jurisdiction: {jurisdiction}
Provide analysis in this structure: 1. DOCUMENT OVERVIEW - Type and purpose -
Parties involved - Effective dates - Governing law 2. KEY PROVISIONS - Main
obligations for each party - Payment terms (if applicable) - Duration and
termination clauses - Confidentiality/IP provisions 3. IMPORTANT CLAUSES -
Limitation of liability - Indemnification - Dispute resolution - Amendment
procedures 4. POTENTIAL CONCERNS - Unusual or one-sided terms - Missing
standard provisions - Ambiguous language 5. ACTION ITEMS - Dates/deadlines to
track - Required actions by parties Flag any terms that typically require
legal review. Use simple language, explain legal terms.
```

Risk Mitigation:

- Clear disclaimers about not being legal advice
- Encourage professional consultation
- Flag complex or unusual provisions
- Avoid interpretations of ambiguous language
- Focus on factual summary rather than recommendations

Key Interview Takeaways

Core Competencies to Demonstrate:

- **Practical Understanding:** Show you can apply AI concepts, not just define them
- **Problem-Solving Approach:** Start simple (prompts), then RAG, then fine-tuning
- **Production Mindset:** Consider latency, cost, failures, monitoring
- **Collaboration Skills:** Can work with data scientists and stakeholders
- **Ethical Considerations:** Understand bias, fairness, safety
- **Continuous Learning:** Stay updated with rapidly evolving field

Your Strengths to Highlight:

- Built production RAG system handling 350 queries/week
- Reduced resolution time by 66% with AI integration
- Experience with AWS Bedrock, OpenSearch, and LLM orchestration
- Practical prompt engineering for operational use cases
- Cross-functional collaboration with data science team
- Focus on measurable business impact ($120K/month protected revenue)

Red Flags to Avoid:

- Don't oversell AI - acknowledge limitations
- Don't ignore ethical concerns - show awareness
- Don't skip evaluation metrics - always measure
- Don't forget failure modes - plan for them
- Don't neglect costs - consider ROI

Questions to Ask Interviewers:

- "What AI/ML frameworks and tools does the team currently use?"
- "How do you handle model monitoring and retraining in production?"
- "What's the process for experimentation and A/B testing?"
- "How do you measure success for AI features?"
- "What are the main AI challenges the team is facing?"