Engineering Case Studies for Candidates

Time Expectation: ~2 hours (minimum) using coding agents (Claude Code, Cursor, Copilot, etc.) **Submission:** GitHub repository link with working code + brief README explaining your approach

---

Case Study 1: Prompt Similarity & Deduplication Service

Background

You're joining a team building a voice AI platform. The platform uses a **prompt-service** that manages hundreds of prompt templates organized in layers (org, os, team, engine, directive). As the prompt library grows, we're seeing issues:

1. **Duplicate prompts** - Similar content exists under different IDs
2. **No semantic search** - Users can only search by title, not meaning
3. **No optimization suggestions** - No way to identify prompts that could be consolidated

Your Task

Build a **Prompt Similarity Service** that:

1. **Generates embeddings** for all prompts in the system
2. **Finds similar prompts** given a query or prompt ID
3. **Identifies potential duplicates** above a similarity threshold
4. **Provides an API** for semantic search

Technical Requirements

```
 Input: A collection of prompts, each with:
{
  "prompt_id": "survey.question.base",
  "category": "survey",
  "layer": "engine",
  "name": "Base Question Template",
  "content": "You are asking the user: {{question_text}}. Wait for their response..."
}

Output: A service with these endpoints:

POST /api/embeddings/generate
  - Generates embeddings for all prompts (or specified list)
  - Stores embeddings efficiently (you choose: file, SQLite, in-memory)

GET /api/prompts/{prompt_id}/similar?limit=5&threshold=0.8
  - Returns prompts similar to the given prompt_id
  - Response: [{ prompt_id, similarity_score, content_preview }]

POST /api/search/semantic
  - Body: { "query": "how to handle user interruptions", "limit": 10 }
  - Returns prompts semantically matching the query

GET /api/analysis/duplicates?threshold=0.9
  - Returns clusters of potentially duplicate prompts
  - Response: [{ cluster_id, prompts: [{ prompt_id, similarity }] }]
```

Sample Data

Use this sample prompt collection for development and testing:

```
[
  {
    "prompt_id": "survey.question.base",
    "category": "survey",
    "layer": "engine",
    "content": "Present the following question to the user naturally: {{question_text}}. Listen carefully to their response. If they
  },
```

```
{
  "prompt_id": "survey.question.with_options",
  "category": "survey",
  "layer": "engine",
  "content": "Ask the user this question: {{question_text}}. The valid responses are: {{options}}. Accept their answer if it reaso
},
{
  "prompt_id": "form.field.prompt",
  "category": "form",
  "layer": "engine",
  "content": "You need to collect: {{field_name}}. Ask the user for this information in a conversational way. Validate their respo
},
{
  "prompt_id": "os.style.warm",
  "category": "os",
  "layer": "os",
  "content": "Maintain a warm, friendly tone throughout the conversation. Use encouraging language. Show genuine interest in the u
},
{
  "prompt_id": "os.style.professional",
  "category": "os",
  "layer": "os",
  "content": "Maintain a professional, courteous demeanor. Be respectful and efficient. Use clear, precise language. Avoid casual
},
{
  "prompt_id": "os.style.empathetic",
  "category": "os",
  "layer": "os",
  "content": "Show empathy and understanding in your responses. Acknowledge the user's feelings. Use supportive language. Be patie
},
{
  "prompt_id": "common.error_recovery",
  "category": "common",
  "layer": "engine",
  "content": "If something goes wrong or the user seems confused, acknowledge the issue calmly. Offer to repeat information or try
},
{
  "prompt_id": "common.error_handling",
  "category": "common",
  "layer": "engine",
  "content": "When errors occur or confusion arises, remain calm and helpful. Apologize briefly if appropriate. Offer alternatives
},
{
  "prompt_id": "receptionist.greeting",
  "category": "receptionist",
  "layer": "engine",
  "content": "Greet the caller warmly. Introduce yourself as {{agent_name}} from {{organization}}. Ask how you can help them today
},
{
  "prompt_id": "receptionist.handoff",
  "category": "receptionist",
  "layer": "engine",
  "content": "You are transitioning the caller to {{next_step}}. Briefly explain what will happen next. Thank them for their patie
},
{
  "prompt_id": "verification.dob",
  "category": "verification",
  "layer": "engine",
  "content": "To verify the caller's identity, ask for their date of birth. Accept various formats (month-day-year, spelled out mo
},
{
  "prompt_id": "verification.identity",
  "category": "verification",
  "layer": "engine",
  "content": "Verify the caller's identity by asking for their date of birth. Be flexible with how they provide it. Repeat back wh
}
```

```
  }
]
```

Evaluation Criteria

Criteria	Weight	What We're Looking For
Embedding Strategy	25%	Choice of embedding model, handling of template variables, batching
Similarity Algorithm	25%	Cosine similarity implementation, threshold tuning, clustering approach
API Design	20%	Clean REST design, error handling, response formats
Code Quality	15%	Readable, well-structured, appropriate abstractions
Documentation	15%	Clear README, setup instructions, design decisions explained

Bonus Points

- Handles prompt variables ({{variable}}) intelligently when computing similarity
- Provides a simple CLI or web UI to visualize clusters
- Includes benchmarks showing performance with 1000+ prompts
- Suggests which prompts could be merged with a unified version

---

Case Study 2: Call Outcome Prediction Model

Background

The voice AI platform handles thousands of calls daily. Each call generates metadata and events that we store for analytics. The business wants to predict call outcomes early to:

1. **Route struggling calls** to human agents before they fail
2. **Identify successful patterns** to replicate across agents
3. **Measure agent quality** based on predicted vs actual outcomes

Your Task

Build a **Call Outcome Predictor** that:

1. **Engineers features** from raw call event data
2. **Trains a classifier** to predict call success/failure
3. **Provides real-time predictions** during active calls
4. **Explains predictions** with feature importance

Technical Requirements

```
 Input: Call event streams with this structure:
{
  "call_id": "uuid",
  "events": [
    { "timestamp": 1704067200, "type": "call_start", "data": {...} },
    { "timestamp": 1704067205, "type": "speech_detected", "data": {"speaker": "agent"} },
    { "timestamp": 1704067210, "type": "speech_detected", "data": {"speaker": "user"} },
    ...
  ],
  "metadata": {
    "agent_id": "receptionist",
    "org_id": "hospital_a",
    "call_purpose": "sdoh_screening"
  },
  "outcome": "completed" | "abandoned" | "transferred" | "error"
}

Output: A service with these capabilities:

POST /api/model/train
  - Body: { "training_data_path": "path/to/calls.json" }
  - Trains model on historical call data
  - Returns: { model_id, metrics: { accuracy, precision, recall, f1 } }

POST /api/predict
  - Body: { "call_id": "uuid", "events_so_far": [...], "metadata": {...} }
  - Returns: {
      "predicted_outcome": "completed",
      "confidence": 0.85,
      "risk_score": 0.15,  # probability of failure
      "top_factors": [
        { "feature": "silence_ratio", "impact": -0.3, "value": 0.4 },
        { "feature": "turn_count", "impact": 0.2, "value": 12 }
      ]
    }

GET /api/model/{model_id}/importance
  - Returns global feature importance rankings
```

Sample Data

Use this synthetic dataset structure. Generate at least 500 samples for training:

```
{
  "calls": [
    {
      "call_id": "call_001",
      "metadata": {
        "agent_id": "receptionist",
        "org_id": "org_a",
        "call_purpose": "sdoh_screening",
        "caller_phone_type": "mobile",
        "time_of_day": "morning",
        "day_of_week": "monday"
      },
      "events": [
        {"ts": 0, "type": "call_start"},
        {"ts": 2, "type": "agent_speech", "duration_ms": 3500, "words": 45},
        {"ts": 6, "type": "user_speech", "duration_ms": 1200, "words": 15},
        {"ts": 8, "type": "silence", "duration_ms": 2000},
        {"ts": 12, "type": "agent_speech", "duration_ms": 2800, "words": 35},
        {"ts": 16, "type": "user_speech", "duration_ms": 800, "words": 8},
        {"ts": 45, "type": "tool_call", "tool": "submit_survey_response"},
        {"ts": 48, "type": "agent_speech", "duration_ms": 2000, "words": 25},
        {"ts": 120, "type": "call_end"}
      ],
      "outcome": "completed",
      "survey_completion_rate": 1.0
    },
    {
      "call_id": "call_002",
      "metadata": {
        "agent_id": "receptionist",
        "org_id": "org_b",
        "call_purpose": "appointment_scheduling",
        "caller_phone_type": "landline",
        "time_of_day": "evening",
        "day_of_week": "friday"
      },
      "events": [
        {"ts": 0, "type": "call_start"},
        {"ts": 2, "type": "agent_speech", "duration_ms": 4000, "words": 50},
        {"ts": 8, "type": "silence", "duration_ms": 5000},
        {"ts": 15, "type": "agent_speech", "duration_ms": 3000, "words": 38},
        {"ts": 20, "type": "silence", "duration_ms": 8000},
        {"ts": 30, "type": "user_speech", "duration_ms": 500, "words": 3},
        {"ts": 35, "type": "call_end"}
      ],
      "outcome": "abandoned",
      "survey_completion_rate": 0.0
    }
  ]
}
```


Suggested Features to Engineer

Feature Category	Examples
Timing	total_duration, time_to_first_user_speech, avg_response_latency
Speech Patterns	agent_talk_ratio, user_talk_ratio, silence_ratio, turn_count
Engagement	words_per_turn_user, words_per_turn_agent, interruption_count
Progress	tools_called_count, survey_questions_answered, verification_attempts

Feature Category	Examples
Context	time_of_day, day_of_week, agent_type, call_purpose

Evaluation Criteria

Criteria	Weight	What We're Looking For
Feature Engineering	30%	Creative, meaningful features; handling of temporal data
Model Selection	25%	Appropriate algorithm choice; proper train/test split; cross-validation
Prediction Quality	20%	Reasonable accuracy; calibrated confidence scores
Explainability	15%	SHAP, LIME, or similar; actionable insights
Code Quality	10%	Clean pipeline; reproducible; well-documented

Bonus Points

- Model works with partial call data (predicts during call, not just after)
- Includes data generation script with realistic patterns
- Compares multiple algorithms (logistic regression, random forest, gradient boosting)
- Provides a simple dashboard showing prediction accuracy over time

---

Case Study 3: LLM Response Quality Evaluator

Background

The voice AI platform uses LLMs to generate conversational responses. Quality varies based on prompts, model settings, and context. Currently, quality is measured manually by listening to calls. We need automated evaluation to:

1. **Score responses** on multiple dimensions (accuracy, empathy, conciseness)
2. **Compare prompt variants** for A/B testing
3. **Flag problematic responses** for human review
4. **Generate improvement suggestions** for low-scoring responses

Your Task

Build an **LLM Response Evaluator** using the "LLM-as-Judge" pattern:

1. **Design evaluation criteria** for voice AI responses
2. **Implement multi-dimensional scoring** using an LLM judge
3. **Build comparison tools** for A/B prompt testing
4. **Create a feedback loop** with improvement suggestions

Technical Requirements

```
Input: Response evaluation requests:
{
  "context": {
    "conversation_history": [...],
    "current_directive": "Ask the user about their food security",
    "user_input": "Well, we sometimes run out of food before the end of the month"
  },
  "response": "I understand that can be really challenging. Just to clarify for our records - would you say that happens often, some
  "metadata": {
    "agent_id": "survey_agent",
    "prompt_version": "v2.1",
    "model": "gpt-4o-mini"
  }
}

Output: A service with these endpoints:

POST /api/evaluate
  - Evaluates a single response
  - Returns: {
      "overall_score": 8.5,
      "dimensions": {
        "task_completion": { "score": 9, "reasoning": "Successfully asked clarifying question" },
        "empathy": { "score": 8, "reasoning": "Acknowledged difficulty appropriately" },
        "conciseness": { "score": 9, "reasoning": "Brief and focused" },
        "naturalness": { "score": 8, "reasoning": "Conversational but slightly formal" },
        "safety": { "score": 10, "reasoning": "No harmful content, appropriate boundaries" }
      },
      "flags": [],
      "suggestions": ["Consider warmer acknowledgment before clarifying question"]
    }

POST /api/evaluate/batch
  - Evaluates multiple responses (for dataset analysis)
  - Returns aggregate statistics and individual scores

POST /api/compare
  - Compares two responses to the same context
  - Body: { "context": {...}, "response_a": "...", "response_b": "..." }
  - Returns: {
      "winner": "a" | "b" | "tie",
      "comparison": {
        "task_completion": { "winner": "tie", "reasoning": "..." },
        "empathy": { "winner": "a", "reasoning": "..." },
        ...
      },
      "recommendation": "Response A is preferred for empathetic scenarios"
    }

POST /api/improve
  - Suggests improved version of a response
  - Returns: {
      "original_score": 6.5,
      "improved_response": "...",
      "improved_score": 8.5,
      "changes_made": ["Added empathetic acknowledgment", "Simplified language"]
    }
```

Evaluation Dimensions

Design your judge prompts to score these dimensions (1-10 scale):

Dimension	Description	Example Indicators

Dimension	Description	Example Indicators
Task Completion	Did the response accomplish the directive?	Asked required question, collected needed info
Empathy	Appropriate emotional attunement?	Acknowledged feelings, avoided dismissiveness
Conciseness	Efficient without being curt?	No unnecessary filler, but not robotic
Naturalness	Sounds like human conversation?	Natural phrasing, appropriate for voice
Safety	No harmful, biased, or inappropriate content?	Boundaries maintained, no hallucinations
Clarity	Easy to understand when spoken aloud?	Simple words, clear structure

Sample Evaluation Cases


```
[
  {
    "id": "eval_001",
    "context": {
      "directive": "Verify the caller's date of birth",
      "user_input": "It's March 15th, 1985"
    },
    "response_good": "Got it, March 15th, 1985. Thank you for confirming that.",
    "response_bad": "Perfect! I have recorded your date of birth as March 15th, 1985. Is there anything else I can help you verify t
    "expected_winner": "response_good",
    "reasoning": "Good response is concise and appropriate; bad response is verbose and goes beyond the task"
  },
  {
    "id": "eval_002",
    "context": {
      "directive": "Ask about food security using SDOH screening question",
      "user_input": "We haven't had enough to eat in weeks. It's been really hard."
    },
    "response_good": "I'm sorry to hear that - that sounds really difficult. I want to make sure we can connect you with the right r
    "response_bad": "Okay, noted. Next question: Within the past 12 months, did you worry that your food would run out before you go
    "expected_winner": "response_good",
    "reasoning": "Good response shows empathy before continuing; bad response ignores emotional content"
  },
  {
    "id": "eval_003",
    "context": {
      "directive": "Handle user confusion about the survey purpose",
      "user_input": "Wait, why are you asking me all these personal questions? Who is this for?"
    },
    "response_good": "That's a fair question. This survey helps us understand if there are any areas where you might benefit from co
    "response_bad": "I'm an AI assistant helping with your health screening. These questions are part of a standard SDOH assessment
    "expected_winner": "response_good",
    "reasoning": "Good response is reassuring and human; bad response is cold and bureaucratic"
  }
]
```


Evaluation Criteria

Criteria	Weight	What We're Looking For
Judge Prompt Design	30%	Clear criteria, reduces bias, consistent scoring
Multi-dimensional Analysis	25%	Meaningful dimensions, proper aggregation
Comparison Logic	20%	Fair A/B comparison, handles ties appropriately
Improvement Generation	15%	Actionable suggestions, actual quality

Criteria	Weight	What We're Looking For
Code Quality	10%	Clean architecture, good error handling

Bonus Points

- Implements calibration (scoring consistency across runs)
- Handles edge cases (empty responses, very long responses, non-English)
- Includes cost optimization (batching, caching, model selection)
- Provides analysis of scoring patterns across agent types or prompt versions

---

Submission Guidelines

What to Submit

1. **GitHub Repository** (public or private with access granted)
    - Working code that runs locally
    - Requirements file (requirements.txt, pyproject.toml, or similar)
    - Sample data or data generation scripts
    - Tests demonstrating functionality
2. **README.md** containing:
    - Setup instructions (should work with `pip install` + `python main.py`)
    - Design decisions and tradeoffs
    - What you would improve with more time
    - Any assumptions you made
3. **Brief Demo** (optional but appreciated)
    - Screen recording (2-5 min) showing the solution working
    - Or deployed demo link

Technical Constraints

- **Language:** Python preferred (TypeScript acceptable)
- **LLM API:** Use OpenAI, Anthropic, or Google
- **Embeddings:** Any provider (OpenAI, Voyage, Cohere, or open-source)
- **Framework:** FastAPI preferred for APIs; any ML framework for models

Evaluation Process

1. We review your code for quality and approach
2. We run your solution against our test cases
3. 30-minute call to discuss your design decisions
4. We may ask you to extend the solution live

Questions?

If anything is unclear, email[hdokoohaki@gobloominghealth.com] with your question. We'll respond within 24 hours and may update these instructions if the question benefits all candidates.

Good luck! We're excited to see your approach.