Technical Presentation

1. Architecture Diagram

1.1 High-Level System Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                         Client Layer                            │
│              (Web / Mobile / API Clients / curl)                │
└────────────────────────────┬────────────────────────────────────┘
                             │ HTTP/REST
                             ▼
┌────────────────────────────────────────────────────────────────┐
│                      FastAPI Gateway                           │
│  ┌──────────┐ ┌──────────┐ ┌──────────┐ ┌──────────┐           │
│  │ Threads  │ │ Messages │ │Summaries │ │Collaborate│          │
│  │  Route   │ │  Route   │ │  Route   │ │  Route   │           │
│  └────┬─────┘ └────┬─────┘ └────┬─────┘ └────┬─────┘           │
└───────┼────────────┼────────────┼────────────┼─────────────────┘
        │            │            │            │
        ▼            ▼            ▼            ▼
┌─────────────────────────────────────────────────────────────────┐
│                      Service Layer                              │
│  ┌──────────────────┐  ┌──────────────────┐  ┌──────────────┐   │
│  │  Thread Manager  │  │ Message Handler  │  │  Summarizer  │   │
│  └────────┬─────────┘  └────────┬─────────┘  └──────┬───────┘   │
│           │                     │                    │          │
│           └─────────────────────┼────────────────────┘          │
│                                 │                               │
│                    ┌────────────▼────────────┐                  │
│                    │   LLM Orchestrator      │                  │
│                    │  (Model Selection)      │                  │
│                    └────────────┬────────────┘                  │
└─────────────────────────────────┼───────────────────────────────┘
                                  │
                                  ▼
┌─────────────────────────────────────────────────────────────────┐
│                    OpenRouter Adapter                           │
│              (HTTP Client + Retry Logic)                        │
└─────────────────────────────────┬───────────────────────────────┘
                                  │ HTTPS
                                  ▼
┌─────────────────────────────────────────────────────────────────┐
│                   OpenRouter API                                │
│  ┌─────────────┐  ┌──────────────┐  ┌────────────────┐          │
│  │  GPT-4      │  │  Claude 3.5  │  │  GPT-3.5       │          │
│  │  Turbo      │  │  Sonnet      │  │  Turbo         │          │
│  └─────────────┘  └──────────────┘  └────────────────┘          │
└─────────────────────────────────────────────────────────────────┘
                                  │
                                  ▼
┌─────────────────────────────────────────────────────────────────┐
│                     PostgreSQL Database                         │
│  ┌────────────┐ ┌────────────┐ ┌────────────┐ ┌────────────┐         │
│  │  threads   │ │  messages  │ │  summaries │ │token_usage │         │
│  └────────────┘ └────────────┘ └────────────┘ └────────────┘         │
└──────────────────────────────────────────────────────────────────────┘
```

1.2 Threading Workflow

Each thread maintains:

- **System Prompt**: Defines assistant behavior (constant per thread)
- **Message History**: All user/assistant exchanges
- **Current Model**: Default model for new messages
- **Auto-Summaries**: Compressed conversation history

1.3 Message Flow: Client → Service → LLMs

```
┌──────────────────────────────────────────────────────────────────────┐
│ Step 1: Client sends message                                         │
│   POST /api/threads/{thread_id}/messages                             │
│   Body: {"content": "Hello", "model": "optional"}                    │
└──────────────────────────────────────────────────────────────────────┘
                              │
                              │
                              ▼
┌──────────────────────────────────────────────────────────────────────┐
│ Step 2: Message Handler receives request                             │
│   - Acquires thread-level lock (concurrency control)                 │
│   - Validates thread exists                                          │
└──────────────────────────────────────────────────────────────────────┘
                              │
                              │
                              ▼
┌──────────────────────────────────────────────────────────────────────┐
│ Step 3: Save user message to database                                │
│   - INSERT INTO messages (role='user', content)                      │
│   - UPDATE threads SET message_count += 1                            │
└──────────────────────────────────────────────────────────────────────┘
                              │
                              │
                              ▼
┌──────────────────────────────────────────────────────────────────────┐
│ Step 4: Check summarization trigger                                  │
│   - If message_count % 10 == 0:                                      │
│     asyncio.create_task(generate_summary)  [non-blocking]            │
└──────────────────────────────────────────────────────────────────────┘
                              │
                              │
                              ▼
┌──────────────────────────────────────────────────────────────────────┐
│ Step 5: Assemble context                                             │
│   context = [                                                        │
│     {"role": "system", "content": thread.system_prompt},             │
│     {"role": "assistant", "content": "[Summary]: ..."},  (if any)│
│     ...recent messages (excluding summarized ones)...                │
│   ]                                                                  │
└──────────────────────────────────────────────────────────────────────┘
                              │
                              │
                              ▼
┌──────────────────────────────────────────────────────────────────────┐
│ Step 6: Token trimming                                               │
│   - Count tokens using tiktoken                                 │
│   - If > max_context_tokens (8000):                             │
│     Preserve: system prompt + summary                           │
│     Trim: oldest messages first                                 │
└─────────────────────────────────────────────────────────────────┘
                             │
                             │
                             ▼
┌─────────────────────────────────────────────────────────────────┐
│ Step 7: Model selection                                         │
│   effective_model = message.model || thread.current_model       │
│   - Validate model exists in registry                           │
└─────────────────────────────────────────────────────────────────┘
                             │
                             │
                             ▼
┌─────────────────────────────────────────────────────────────────┐
│ Step 8: Call OpenRouter API                                     │
│   POST https://openrouter.ai/api/v1/chat/completions            │
│   - Automatic retry on timeout (3 attempts, exponential backoff)│
│   - Error handling: 429 rate limit, 401 auth, 400 bad request  │
└─────────────────────────────────────────────────────────────────┘
                             │
                             │
                             ▼
┌─────────────────────────────────────────────────────────────────┐
│ Step 9: Save assistant response                                 │
│   - INSERT INTO messages (role='assistant', content, model)     │
│   - UPDATE threads SET message_count += 1                       │
│   - Track token usage for cost monitoring                       │
└─────────────────────────────────────────────────────────────────┘
                             │
                             │
                             ▼
┌─────────────────────────────────────────────────────────────────┐
│ Step 10: Return response to client                              │
│   Response: {message_id, content, model, tokens}                │
└─────────────────────────────────────────────────────────────────┘
```

2. LLM Orchestration

2.1 Model Routing Strategy

Available Models:

- `openai/gpt-4-turbo` (128K context, 0.01/0.03 per 1K tokens)
- `anthropic/claude-3.5-sonnet` (200K context, 0.003/0.015 per 1K tokens)
- `openai/gpt-3.5-turbo` (16K context, 0.0005/0.0015 per 1K tokens)

Model Selection Rule:

```
Effective Model = message.model (if provided and valid)
                  || thread.current_model (fallback)

Priority: Message-level override > Thread-level default
```

Example Scenario:

```
Thread Configuration:
  current_model = "openai/gpt-4-turbo"

Message 1: model = null
  → Uses: GPT-4 Turbo

Message 2: model = "anthropic/claude-3.5-sonnet"
  → Uses: Claude 3.5 Sonnet

Message 3: model = null
  → Uses: GPT-4 Turbo (back to default)

Message 4: model = "openai/gpt-3.5-turbo"
  → Uses: GPT-3.5 Turbo
```

2.2 Context and System Prompt Maintenance

Context Assembly Process:

1. **System Prompt** (always first)
    - Stored in `threads.system_prompt` column
    - Constant for the entire thread lifecycle
    - Defines assistant's behavior and personality
2. **Latest Summary** (if exists)
    - Injected as assistant message with prefix: `[Previous conversation summary]: ...`
    - Replaces the actual messages it summarizes
    - Reduces token usage while preserving context
3. **Recent Messages** (after last summary)
    - Fetched from database: `WHERE created_at > last_summary.created_at`
    - Limited to `max_context_messages` (default: 20)
    - Ordered chronologically

Context Structure:

```
[
  {
    "role": "system",
    "content": "You are a helpful product consultant with expertise in SaaS..."
  },
  {
    "role": "assistant",
    "content": "[Previous conversation summary]: Discussed Q1 performance,
identified 3 key risks..."
  },
  {
    "role": "user",
    "content": "What about Q2 planning?"
  },
  {
    "role": "assistant",
    "content": "For Q2, I recommend..."
  },
  ...
]
```

Token Management:

- Count tokens using `tiktoken` (accurate for OpenAI models)
- If total exceeds `max_context_tokens` (8000):
    - **Preserve:** System prompt + summary (essential context)
    - **Trim:** Oldest messages first (FIFO)
    - Ensures we stay within model limits

---

3. Auto-Summarization Logic

3.1 Summarization Approach

Trigger Condition:

```
if thread.message_count % SUMMARIZATION_MESSAGE_THRESHOLD == 0:
    # Trigger: Every 10 messages (configurable)
    asyncio.create_task(generate_summary(thread_id))
```

Key Design Decisions:

- **Async & Non-blocking**: Summary generation runs in background
- **Doesn't block user messages**: If summary fails, conversation continues
- **Configurable threshold**: Default 10, adjustable via `SUMMARIZATION_MESSAGE_THRESHOLD`

3.2 Summary Generation Process

```
┌─────────────────────────────────────────────────────────────────┐
│ 1. Fetch last N messages (N = threshold, default 10)            │
│    SELECT * FROM messages                                       │
│    WHERE thread_id = ? ORDER BY created_at DESC LIMIT 10        │
└─────────────────────────────────────────────────────────────────┘
                             │
                             │
                             ▼
┌─────────────────────────────────────────────────────────────────┐
│ 2. Format messages for summarization                            │
│    User: What are the Q2 risks?                                 │
│    Assistant: Based on market analysis, there are 3 key risks...│
│    User: How do we mitigate them?                               │
│    Assistant: Here's a mitigation strategy...                   │
└─────────────────────────────────────────────────────────────────┘
                             │
                             │
                             ▼
┌─────────────────────────────────────────────────────────────────┐
│ 3. Call LLM with summarization prompt                           │
│    Model: openai/gpt-4-turbo (configurable)                     │
│    Temperature: 0.3 (lower for consistency)                     │
│    Max tokens: 200                                              │
│                                                                 │
│    Prompt Template:                                             │
│    "Summarize the following conversation concisely.             │
│     Focus on:                                                   │
│     - Key topics discussed                                      │
│     - Important decisions or conclusions                        │
│     - User's main intent and questions                          │
│     - Any action items or follow-ups                                 │
│     Keep the summary under 150 words."                               │
└──────────────────────────────────────────────────────────────────────┘
                              │
                              │
                              ▼
┌──────────────────────────────────────────────────────────────────────┐
│ 4. Store summary in database                                         │
│    INSERT INTO summaries (                                           │
│      thread_id,                                                      │
│      summary_text,                                                   │
│      covered_message_count = 10,                                     │
│      covered_message_ids = [msg1_id, msg2_id, ...],                  │
│      trigger_reason = "message_count"                                │
│    )                                                                 │
└──────────────────────────────────────────────────────────────────────┘
```

3.3 Context Management Best Practices

Why Auto-Summarization Matters:

1. **Token Efficiency**
   - 10 messages ≈ 2000-3000 tokens
   - Summary ≈ 150-200 tokens
   - **Savings: ~90% reduction**
2. **Cost Reduction**
   - Fewer tokens = lower API costs
   - Especially important for long conversations
3. **Context Window Management**
   - Prevents hitting model token limits
   - Maintains conversation coherence

Integration with Context Assembly:

```
Without Summary:
  [System Prompt] + [50 messages] = ~10,000 tokens ❌ Exceeds limit

With Summary:
  [System Prompt] + [Summary of 40 msgs] + [10 recent msgs] = ~3,000 tokens ✅
```

Summary Lifecycle:

```
Messages 1-10  → Summary 1 created
Messages 11-20 → Summary 2 created
Messages 21-30 → Summary 3 created

Context for Message 31:
  - System Prompt
  - Summary 3 (covers msgs 21-30)
  - Messages 31 (current)
```

4. Demo Screenshots / Results

4.1 Sample Chat: Product Strategy Discussion

Initial Thread Creation:

```
$ curl -X POST http://localhost:8001/api/threads \
  -H "Content-Type: application/json" \
  -d '{
    "title": "SaaS Product Launch Strategy",
    "system_prompt": "You are a strategic product consultant with expertise in
B2B SaaS.",
    "current_model": "openai/gpt-4-turbo"
  }'

Response:
{
  "thread_id": "550e8400-e29b-41d4-a716-446655440000",
  "title": "SaaS Product Launch Strategy",
  "current_model": "openai/gpt-4-turbo",
  "message_count": 0,
  "created_at": "2025-02-03T10:00:00Z"
}
```

Conversation Flow:

Message 1 (User → GPT-4):

```
POST /api/threads/550e8400.../messages
{
  "content": "What are the key strategies for launching a SaaS product in 2025?"
}

Response:
{
  "role": "assistant",
  "content": "Launching a SaaS product in 2025 requires focus on:\n1. Product-Led
Growth (PLG)...\n2. AI Integration...\n3. Community Building...",
  "model": "openai/gpt-4-turbo",
  "tokens": 256
}
```

Message 2 (User → Claude, model switch):

```
POST /api/threads/550e8400.../messages
{
  "content": "Can you elaborate on the PLG strategy with specific tactics?",
  "model": "anthropic/claude-3.5-sonnet"
}

Response:
{
  "role": "assistant",
  "content": "Product-Led Growth for SaaS involves:\n\n**Free Trial
Strategy:**\n- Offer 14-day full-feature trial...\n\n**Onboarding
Excellence:**\n- Interactive product tours...",
  "model": "anthropic/claude-3.5-sonnet",
  "tokens": 412
}
```

Messages 3-10: (Continued discussion on pricing, marketing, customer success...)

4.2 Auto-Generated Summary (After 10 Messages)

```
$ curl http://localhost:8001/api/threads/550e8400.../summaries

Response:
{
  "summaries": [
    {
      "summary_id": "770e8400-e29b-41d4-a716-446655440001",
      "summary_text": "Discussion focused on SaaS product launch strategies for
2025. Key topics: Product-Led Growth (PLG) with free trials and interactive
onboarding, AI integration for personalization, pricing strategy (freemium vs.
tiered), community-building tactics, and customer success metrics. User
emphasized B2B market targeting small businesses. Agreed on MVP approach with 3
core features: dashboard analytics, team collaboration, and API integrations.",
      "covered_message_count": 10,
      "trigger_reason": "message_count",
      "created_at": "2025-02-03T10:15:00Z"
    }
  ]
}
```

4.3 Multi-Agent Collaboration Example

Request:

```
$ curl -X POST http://localhost:8001/api/collaborate \
  -H "Content-Type: application/json" \
  -d '{
    "query": "How should we price our B2B SaaS product for small businesses?",
    "include_process": true
  }'
```

Response (Abbreviated):

```
{
  "collaboration_id": "abc12345",
  "final_response": "For B2B SaaS targeting small businesses, I recommend a
three-tier pricing strategy:\n\n**Starter Plan ($29/month)**\n- Up to 5 users\n-
Core features\n- Email support\n\n**Professional Plan ($79/month)**\n- Up to 25
users\n- Advanced features + integrations\n- Priority support\n\n**Enterprise
Plan (Custom)**\n- Unlimited users\n- Full feature access\n- Dedicated account
manager\n\nKey considerations: Keep entry point low ($29) to reduce friction,
offer annual discounts (15-20%), and include a 14-day free trial...",

  "collaboration_process": {
    "plan": "1. Analyze small business budget constraints\n2. Research competitor
pricing\n3. Define value-based tiers\n4. Calculate unit economics\n5. Recommend
pricing structure",

    "draft": "[Writer's full content generation based on plan...]",

    "steps": [
      {
        "step": 1,
        "role": "planner",
        "model": "openai/gpt-4-turbo",
        "output": "1. Analyze small business budget constraints...",
        "tokens": 180
      },
      {
        "step": 2,
        "role": "writer",
        "model": "anthropic/claude-3.5-sonnet",
        "output": "[Detailed pricing strategy draft...]",
        "tokens": 650
      },
      {
        "step": 3,
        "role": "reviewer",
        "model": "openai/gpt-4-turbo",
        "output": "[Polished final response...]",
        "tokens": 520
      }
    ]
  },

  "metadata": {
    "total_tokens": 1350,
    "duration_ms": 7200.5,
    "estimated_cost_usd": {
      "planner": 0.0027,
      "writer": 0.0098,
      "reviewer": 0.0078,
      "total": 0.0203
    }
  }
}
```

4.4 Token Usage & Cost Tracking

```
$ curl http://localhost:8001/api/usage/summary?days=7

Response:
{
  "period_days": 7,
  "summary": {
    "total_input_tokens": 12500,
    "total_output_tokens": 18750,
    "total_tokens": 31250,
    "total_requests": 125,
    "total_cost_usd": 0.58
  },
  "by_model": [
    {
      "model": "openai/gpt-4-turbo",
      "total_tokens": 18000,
      "request_count": 75,
      "cost_usd": {"total": 0.42}
    },
    {
      "model": "anthropic/claude-3.5-sonnet",
      "total_tokens": 10500,
      "request_count": 40,
      "cost_usd": {"total": 0.13}
    },
    {
      "model": "openai/gpt-3.5-turbo",
      "total_tokens": 2750,
      "request_count": 10,
      "cost_usd": {"total": 0.03}
    }
  ]
}
```

5. Next Steps / Enhancements

5.1 Ordering & Message Flow Improvements

Current State:

- Sequential message processing per thread (thread-level locks)
- Async summarization (non-blocking)

Proposed Enhancements:

1. **Message Priority Queue**
   - Implement priority levels (urgent, normal, low)
   - Process high-priority messages first
   - Use case: Customer support escalations
2. **Batch Message Processing**
   - Group multiple user messages for single LLM call
   - Reduce API overhead

    - Better for rapid-fire questions
3. **Streaming Responses**
    - Implement Server-Sent Events (SSE)
    - Stream tokens as they're generated
    - Improved perceived latency

5.2 Latency Optimization

Current Bottlenecks:

- OpenRouter API latency: 2-5 seconds
- Database queries: 10-50ms
- Context assembly: 5-20ms

Optimization Strategies:

1. **Caching Layer**
    - Redis cache for thread metadata
    - Cache recent messages (last 20)
    - Reduce database round-trips
    - **Expected improvement: 30-50ms saved**
2. **Parallel Context Assembly**
    - Fetch system prompt, summary, and messages concurrently
    - Use `asyncio.gather()`
    - **Expected improvement: 20-30ms saved**
3. **Model Selection Optimization**
    - Use faster models for simple queries (GPT-3.5)
    - Route complex queries to GPT-4/Claude
    - Implement query complexity classifier
    - **Expected improvement: 40-60% cost reduction**
4. **Connection Pooling**
    - Increase database pool size for high traffic
    - Keep-alive connections to OpenRouter
    - **Expected improvement: 10-20ms saved**

5.3 Multi-Agent Role Enhancements

Current Implementation:

- 3 agents: Planner → Writer → Reviewer
- Fixed sequential flow
- Predefined models per role

Advanced Multi-Agent Patterns:

1. **Specialized Agent Roles**

```
User Query
    │
    ├─→ [Classifier Agent] → Determines query type
    │
    ├─→ [Technical Agent] → For technical questions
    ├─→ [Creative Agent] → For brainstorming
    ├─→ [Analytical Agent] → For data analysis
    │
    └─→ [Synthesizer Agent] → Combines outputs
```

2. **Parallel Agent Processing**

```
User Query: "Design a marketing campaign"
    │
    ├─→ [Strategy Agent] ─┐
    ├─→ [Creative Agent] ──┼─→ [Integration Agent] → Final Output
    └─→ [Budget Agent] ────┘
```

3. **Iterative Refinement**

```
Draft → [Critic Agent] → Feedback → [Writer Agent] → Improved Draft
             │                               │
             └──────────── Loop 2-3x ────────┘
```

4. **Context-Aware Agent Selection**
   - Analyze conversation history
   - Dynamically select best agent for current context
   - Use lightweight model for routing decisions

5.4 Advanced Summarization

Current Limitation:

- Single-level summaries
- Fixed 10-message threshold

Hierarchical Summarization:

```
Level 1: Every 10 messages → Summary 1 (150 words)
Level 2: Every 5 summaries → Meta-Summary (100 words)
Level 3: Every 5 meta-summaries → Ultra-Summary (50 words)

For 250 messages:
  Without hierarchy: 25 summaries × 150 words = 3,750 words
  With hierarchy: 1 ultra-summary = 50 words
  Compression ratio: 75:1
```

Adaptive Summarization:

- Trigger based on token count, not just message count
- Summarize when context > 5000 tokens
- Smart detection of topic changes

5.5 Production Readiness

Monitoring & Observability:

1. **Distributed Tracing**
    - OpenTelemetry integration
    - Track request flow across services
    - Identify bottlenecks
2. **Metrics Dashboard**
    - Prometheus + Grafana
    - Track: latency, error rate, token usage, costs
    - Alerts for anomalies
3. **Logging Enhancements**
    - Structured logging (already implemented with structlog)
    - Centralized log aggregation (ELK stack)
    - Request ID tracking

Scalability:

1. **Horizontal Scaling**
    - Stateless API servers (already achieved)
    - Load balancer (Nginx/HAProxy)
    - Database read replicas
2. **Rate Limiting**
    - Per-user rate limits
    - Token bucket algorithm
    - Prevent abuse
3. **Queue-Based Processing**
    - RabbitMQ/Redis for message queue
    - Decouple API from LLM calls
    - Better handling of traffic spikes

Security:

1. **API Key Management**
    - Rotate OpenRouter keys
    - Vault integration
    - Per-environment keys
2. **Input Validation**
    - Sanitize user inputs
    - Prevent prompt injection
    - Content filtering
3. **Audit Logging**
    - Track all API calls
    - User activity logs
    - Compliance (GDPR, SOC2)