

AIML Engineer Assessment Submission

Name: [Your Name]

Date: September 2, 2025

Table of Contents

1. Executive Summary
2. Multi-Agent System Design
 - System Overview Diagram
 - Lead Triage Agent
 - Engagement Agent
 - Campaign Optimization Agent
 - Communication Protocol
3. Adaptive Memory Architecture
 - Memory Layer Diagram
 - Short-Term Memory
 - Long-Term Memory
 - Episodic Memory
 - Semantic Memory
 - Memory Consolidation Algorithm
4. Architecture Decision Records (ADRs)
5. API Documentation (OpenAPI)
6. Deployment Runbooks
7. Agent Interaction Analysis
8. Security Enhancements
9. Scalability Analysis
10. Appendix

Executive Summary

Provide a concise overview of the marketing multi-agent system, its goals, and how it demonstrates collaborative intelligence, adaptive memory, and real-world deployability.

Multi-Agent System Design

System Overview Diagram

[Insert High-Resolution Block Diagram showing MCP Server, Lead Triage Agent, Engagement Agent, Campaign Optimization Agent, with JSON-RPC over HTTP/WebSocket connections]

Lead Triage Agent

Agent Name	Responsibilities	Input	Output
Lead Triage Agent	Classify incoming leads into 3 categories	Lead JSON payload	{ category: "..."

Describe classification logic, models, and integration points.

Engagement Agent

Agent Name	Responsibilities	Input	Output
Engagement Agent	Send personalized outreach (email, social)	{ category, lead }	Email/Social API calls

Explain outreach workflows and context handling.

Campaign Optimization Agent

Agent Name	Responsibilities	Input	Output
Campaign Optimization Agent	Monitor KPIs; adjust strategies	Metrics JSON array	Strategy update JSON messages

Communication Protocol

- **JSON-RPC 2.0** for request/response and notifications.
- **HTTP Transport** for REST-like calls.
- **WebSocket Transport** for real-time updates.

Example JSON-RPC Request:

```
{
  "jsonrpc": "2.0",
  "method": "classifyLead",
  "params": { "leadData": {...} },
  "id": 1
}
```

Adaptive Memory Architecture

Memory Layer Diagram

[Insert Diagram of Short-Term, Episodic, Long-Term, Semantic memory layers with consolidation arrows]

Short-Term Memory

- Stored in in-process cache (e.g., Redis).
- Holds current conversation context.

Long-Term Memory

Field	Type	Description
lead_id	string	Unique lead identifier
history	list	Previous interaction logs
preferences	object	Customer preferences

Episodic Memory

- Logs of successful lead resolutions.
- Stored in document store; supports querying past sessions.

Semantic Memory

- Domain knowledge graph in Neo4j.
- Nodes: Lead, Campaign, Keyword; Relationships: INTERESTED_IN, CONVERTED_IN.

Memory Consolidation Algorithm

```
# Nightly consolidation job
def consolidate_memory(short_term, long_term, episodic):
    for lead_id, session in short_term.items():
        if len(session['interactions']) > THRESHOLD:
            summary = summarize(session)
            long_term.store(lead_id, summary)
            episodic.log(lead_id, session)
            short_term.clear(lead_id)
```

Architecture Decision Records (ADRs)

Decision	Rationale
JSON-RPC 2.0 vs REST	Enables bidirectional calls and event notifications via WebSocket
WebSocket for Real-Time Updates	Low-latency push model, essential for campaign monitoring
Neo4j for Semantic Memory	Efficient graph traversals for relationship queries
Redis for Short-Term Memory	In-memory speed for active conversations
TLS 1.3 for Transport Security	Latest encryption standard ensuring secure agent communication

API Documentation (OpenAPI)

```
openapi: 3.0.0
info:
  title: Marketing Multi-Agent API
  version: 1.0.0
paths:
  /triage:
    post:
      summary: Classify a new lead
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Lead'
      responses:
        '200':
          description: Triage result
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/TriageResult'
components:
  schemas:
    Lead:
      type: object
      properties:
        name: { type: string }
        email: { type: string }
        message: { type: string }
    TriageResult:
      type: object
      properties:
        category: { type: string }
```

(Add /engage, /optimize, /memory paths similarly.)

Deployment Runbooks

1. **Pre-Deployment:** Set environment variables, configure secrets.

2. **Deployment Steps:**

- `git clone <repo>`
- `docker-compose up -d`
- `kubectl apply -f k8s/`

3. **Verification:**

- Call `/health` endpoints.
- Run smoke tests.

4. **Rollback:**

- `docker-compose down`
- Re-deploy previous image tag.

Endpoint	Expected Response	Purpose
/health	{status: "OK"}	MCP server health check
/agent/triage	200 OK	Triage agent health
/agent/optimize	200 OK	Optimization agent health

Agent Interaction Analysis

Conversation Flow Diagram

[Insert flowchart: Client → Triage → Engagement → Optimization → Client]

Sequence Diagram

Step	Caller	Callee	Message
1	Client	Triage Agent	JSON-RPC classifyLead
2	Triage Agent	Engagement	JSON-RPC engageLead
3	Engagement	Optimization	JSON-RPC optimizeCampaign
4	Optimization	Client	CampaignReport JSON

Security Enhancements

Layer	Recommendation
Transport	Enforce TLS 1.3 for HTTP and WebSocket
Authentication	JWT tokens with HMAC-SHA256

Layer	Recommendation
Validation	JSON Schema validation on all inputs
Rate Limiting	100 req/min per client IP

Scalability Analysis

Component	Bottleneck	Mitigation
Triage Agent	Single-threaded processing	Scale-out replicas behind load balancer
Memory Store	Graph DB query latency	Shard by customer segments
RPC Transport	High message throughput	Introduce message broker (Kafka)

Auto-Scaling Configuration

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: triage-agent-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: triage-agent
  minReplicas: 2
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 70
```

Appendix

Final scripts/train.py

```
# (Include your full train.py with quick test block here)
```

Sample Data (training_data.json)

text	label
"I forgot my password"	account
"My order hasn't arrived yet"	logistics
"I was charged twice"	billing
"The app keeps crashing"	technical

★End of Document★