Intelligent Multi-Agent Customer Service System

System Design Document

Executive Summary

To eliminate customer frustration caused by repeated transfers and loss of context, we propose a **hierarchical multi-agent architecture** with a **single orchestration layer** and **specialized expert agents**.
This design preserves context, enables deep specialization, and provides a seamless, “one-conversation” customer experience.

---

1. System Architecture Choice

✅ Chosen Pattern: Hierarchical Architecture

Why not centralized?

- A single “mega-agent” becomes:
    - Hard to maintain
    - Difficult to scale
    - Error-prone across domains (products, billing, tech)

Why not fully decentralized?

- Peer-to-peer agents:
    - Lose global customer context
    - Recreate today’s “handoff chaos”
    - Make accountability unclear

✅ Why hierarchical is the right fit

A **hierarchical pattern** introduces a **Conversation Orchestrator** that:

- Owns customer context
- Routes tasks to specialists
- Synthesizes responses
- Prevents context loss

Customer sees one assistant.
System internally coordinates many experts.

Alignment with customer pain

Customer Problem	Architectural Solution
Repeating information	Centralized context store
Department transfers	Invisible internal routing
Conflicting answers	Orchestrated synthesis
Slow resolution	Parallel expert consultation

---

2. Agent Roles & Specializations

Agent Overview

Agent	Role	Core Expertise
Conversation Orchestrator	Control & coordination	Intent detection, routing, synthesis
Product Intelligence Agent	Product domain	Catalog, recommendations, compatibility
Order & Logistics Agent	Fulfillment	Order status, shipping, returns
Technical Support Agent	Post-purchase support	Setup, troubleshooting
Billing & Payments Agent	Financials	Invoices, refunds, payment issues

2.1 Conversation Orchestrator (Supervisor Agent)

Responsibilities

- Maintains full customer context
- Interprets intent(s)
- Delegates to expert agents
- Merges partial answers into one response

Why it matters

- Prevents “agent ping-pong”
- Guarantees continuity
- Enforces consistent tone and policy

2.2 Product Intelligence Agent

Responsibilities

- Product recommendations
- Compatibility checks
- Feature comparisons
- Stock awareness (read-only)

Customer impact

- Accurate, personalized recommendations
- Reduced pre-purchase friction

2.3 Order & Logistics Agent

Responsibilities

- Order tracking
- Delivery issues
- Returns & exchanges
- Carrier coordination

Customer impact

- Faster resolution of “Where is my order?”
- No re-authentication loops

2.4 Technical Support Agent

Responsibilities

- Installation help
- Troubleshooting
- Warranty and defect triage

Customer impact

- Context-aware support (knows product + order)
- Reduced escalations

2.5 Billing & Payments Agent

Responsibilities

- Payment failures
- Refund status
- Invoice corrections
- Fraud-safe explanations

Customer impact

- Clear, compliant financial communication
- Reduced chargeback risk

Overlap Handling Strategy

Example overlap: Refund request due to defective product

- Technical Agent → confirms defect
- Billing Agent → processes refund
- Orchestrator → sequences and explains outcome

Rule: Agents never negotiate directly with the customer — the orchestrator does.

---

3. Communication Flows

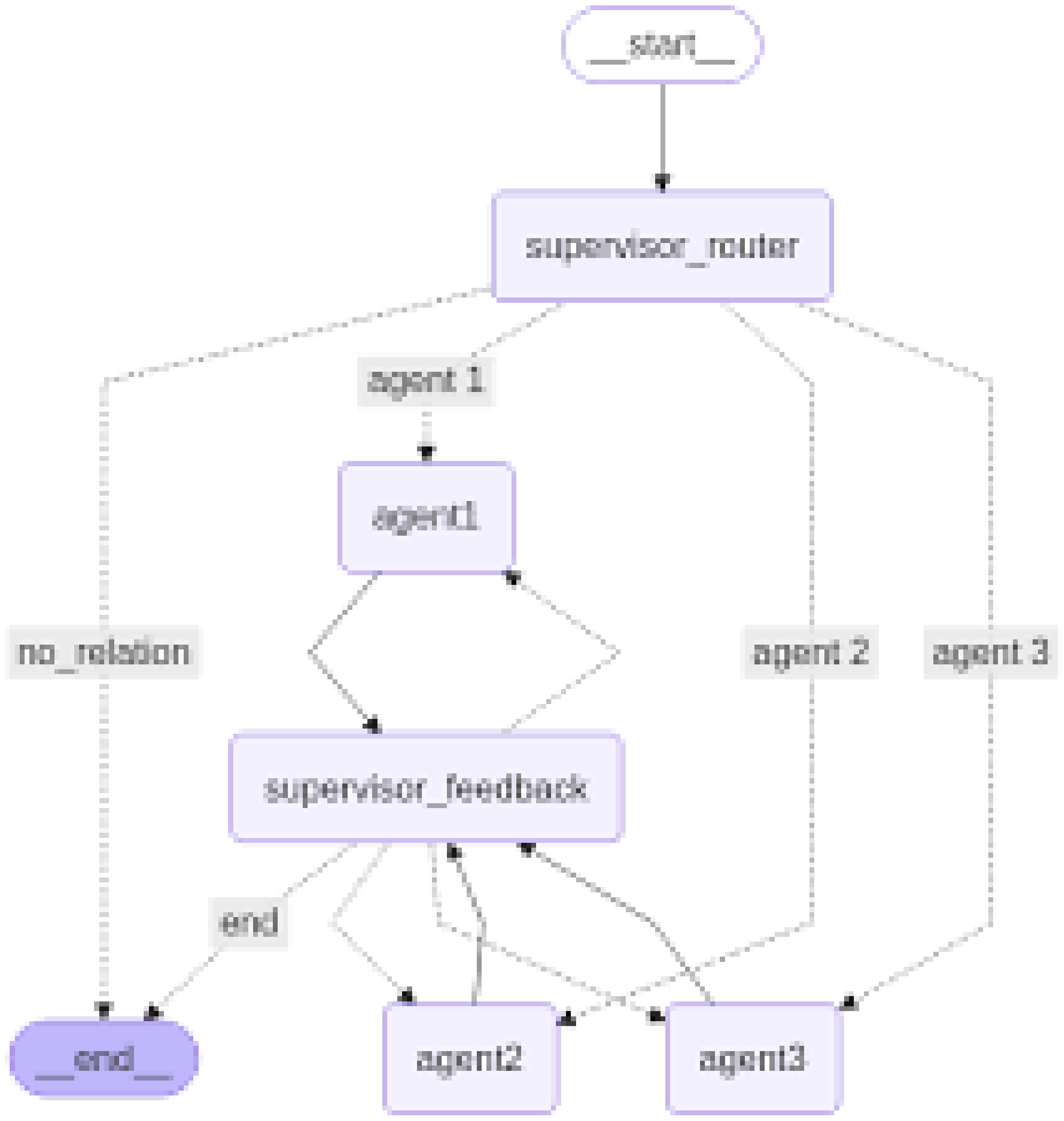
Communication Model

- **Hub-and-spoke**
- No agent-to-agent direct conversation
- All messages flow through the orchestrator

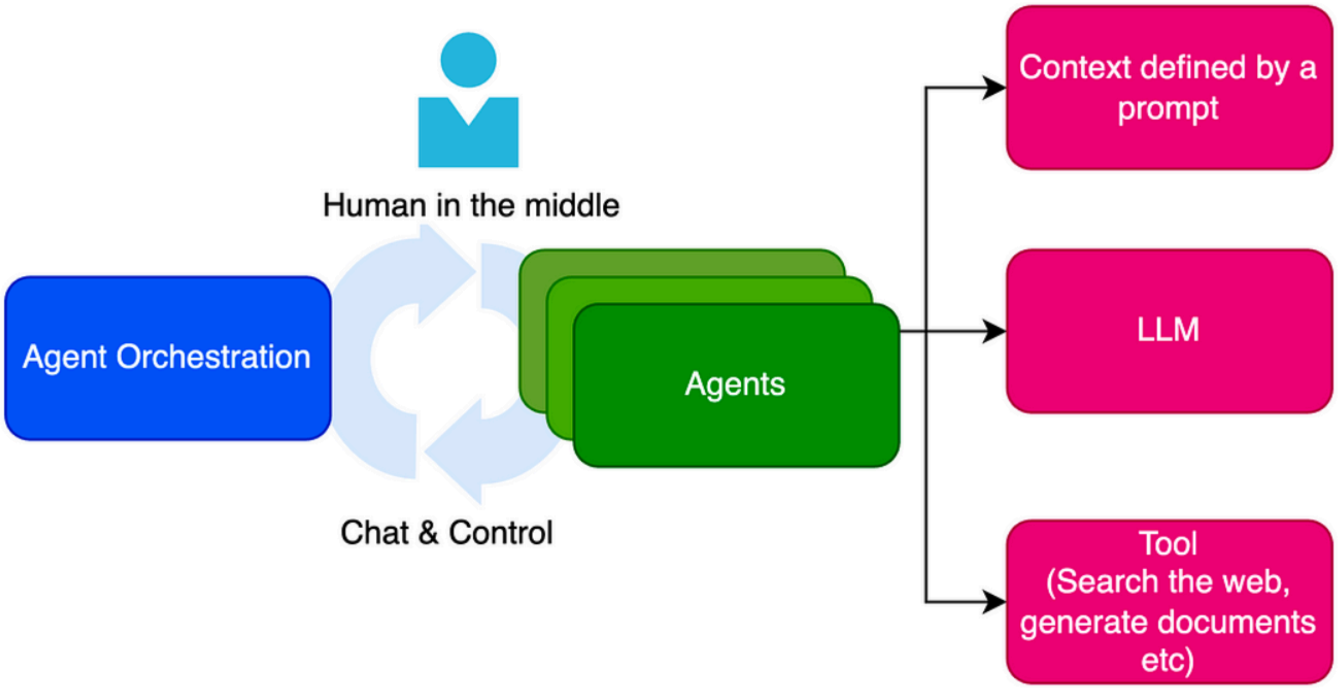
Message Types (Structured)

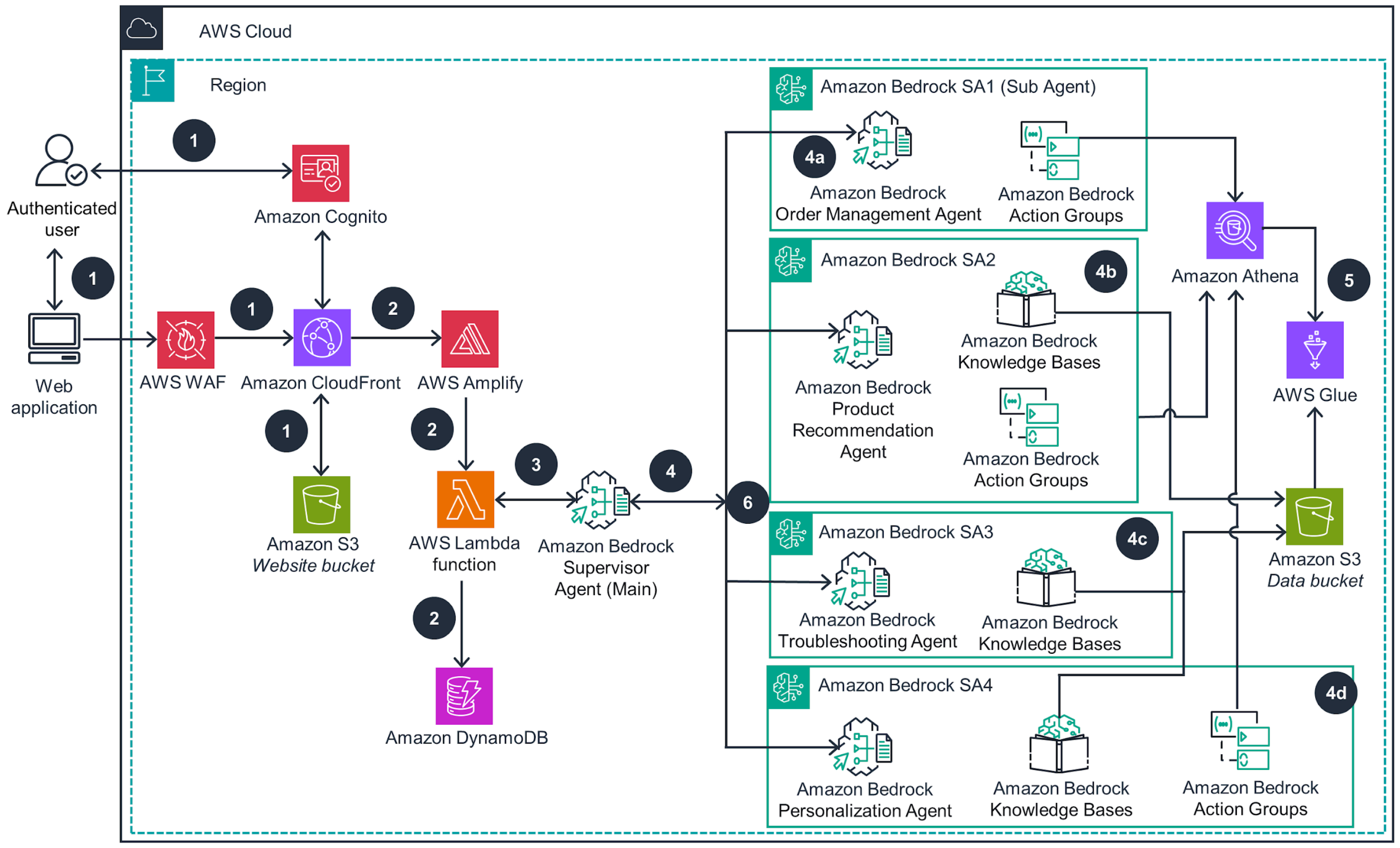
```
{
  "customer_id": "12345",
  "intent": "refund_request",
  "context_refs": ["order_789", "product_456"],
  "agent_response": {
    "status": "eligible",
    "confidence": 0.92,
    "notes": "Defect confirmed under warranty"
  }
}
```

Communication Flow Diagram



__start__
supervisor_router
agent 1
agent1
no_relation
agent 2
agent 3
supervisor_feedback
end
__end__
agent2
agent3


Human in the middle
Agent Orchestration
Chat & Control
Agents
Context defined by a prompt
LLM
Tool
(Search the web, generate documents etc)


AWS Cloud
Region
1
Authenticated user
Amazon Cognito
1
Web application
1
AWS WAF
Amazon CloudFront
2
AWS Amplify
1
Amazon S3
Website bucket
2
AWS Lambda function
3
Amazon Bedrock Supervisor Agent (Main)
4
6
2
Amazon DynamoDB
Amazon Bedrock SA1 (Sub Agent)
4a
Amazon Bedrock Order Management Agent
Amazon Bedrock Action Groups
Amazon Bedrock SA2
4b
Amazon Bedrock Knowledge Bases
Amazon Bedrock Product Recommendation Agent
Amazon Bedrock Action Groups
Amazon Athena
5
AWS Glue
Amazon S3
Data bucket
Amazon Bedrock SA3
4c
Amazon Bedrock Troubleshooting Agent
Amazon Bedrock Knowledge Bases
Amazon Bedrock SA4
4d
Amazon Bedrock Personalization Agent
Amazon Bedrock Knowledge Bases
Amazon Bedrock Action Groups

Flow Example

1. Customer: *“My headphones stopped working and I want a refund.”*
2. Orchestrator:
   - Detects **technical issue + billing**
3. Parallel calls:
   - Technical Agent → defect validation
   - Billing Agent → refund policy check
4. Orchestrator:
   - Combines results
   - Responds with one coherent answer

---

4. Memory & Context Management

4.1 Shared Memory Strategy

Centralized Context Store

- Conversation state
- Customer profile
- Order history
- Partial agent outputs

Read/Write Rules

Agent	Read	Write
Orchestrator	✅	✅
Specialist Agents	✅	❌ (propose only)

4.2 Context Preservation

- Every agent receives:
    - Conversation summary
    - Relevant entities only
- No raw transcript flooding
- Structured, minimal context

4.3 Partial Solution Propagation

Example:

- Technical Agent writes:

```
{
  "finding": "hardware defect",
  "eligible_for_refund": true
}
```

- Orchestrator uses this to:
    - Trigger Billing Agent
    - Explain resolution to customer

4.4 Safeguards Against Information Loss

- Immutable conversation IDs
- Step-level summaries after each agent response
- Automatic replay on failure
- Orchestrator as **single source of truth**

---

5. Customer Experience Improvements

Before

- Multiple transfers
- Repeated explanations
- Inconsistent answers
- Long resolution times

After

- One continuous conversation
- Invisible expert collaboration
- Faster, clearer resolutions
- Higher trust and satisfaction

Final Takeaway

> **This hierarchical multi-agent design turns internal complexity into external simplicity.**

Customers experience:

- One assistant
- One conversation
- One resolution

While the system benefits from:

- Deep specialization
- Strong governance
- Scalable architecture