Problem Statement 13: E-commerce Customer Service AI

AI-Powered Intelligent Customer Service and Support Automation Platform

Problem Overview

Develop an AI-powered customer service platform that provides intelligent, personalized, and efficient customer support for e-commerce businesses. The system should handle multi-channel customer interactions, provide instant responses to common queries, escalate complex issues to human agents, and continuously learn from interactions to improve service quality.

Key Requirements

Core Functionality

- **Multi-Channel Support**: Handle customer inquiries across email, chat, phone, social media, and messaging platforms
- **Intelligent Query Processing**: Natural language understanding for customer intent recognition and context analysis
- **Automated Response Generation**: AI-powered responses with personalized recommendations and solutions
- **Escalation Management**: Smart routing to human agents based on complexity, sentiment, and priority
- **Knowledge Base Integration**: Dynamic access to product information, policies, and troubleshooting guides
- **Order Management Integration**: Real-time access to order status, shipping, returns, and refund processing
- **Sentiment Analysis**: Real-time emotion detection and appropriate response adaptation
- **Multi-Language Support**: Automatic language detection and response in customerâ€™s preferred language

Advanced Features

- **Predictive Customer Service**: Proactive issue identification and resolution suggestions
- **Customer Journey Analytics**: Comprehensive tracking of customer interactions and satisfaction metrics
- **Agent Assistance Tools**: Real-time suggestions and knowledge recommendations for human agents
- **Performance Analytics**: Detailed reporting on response times, resolution rates, and customer satisfaction
- **Integration Ecosystem**: Seamless connectivity with CRM, ERP, inventory, and marketing automation systems

Data Requirements

Customer Data

- Customer profiles and purchase history
- Previous support interactions and resolutions
- Communication preferences and channels
- Satisfaction ratings and feedback

Product and Service Data

- Comprehensive product catalog with specifications
- Inventory levels and availability status
- Pricing, promotions, and discount information
- Shipping policies and delivery options
- Return and refund policies

Operational Data

- Order management system integration
- Real-time inventory and logistics data
- Agent performance and availability metrics
- Knowledge base articles and FAQs
- Historical support ticket data and resolutions

Technical Themes

AI/ML Technologies

- **Natural Language Processing**: Advanced NLP for intent recognition, entity extraction, and context understanding
- **Conversational AI**: Sophisticated chatbot capabilities with context retention and multi-turn conversations
- **Sentiment Analysis**: Real-time emotion detection and response adaptation
- **Machine Learning**: Continuous learning from interactions to improve response accuracy and personalization
- **Predictive Analytics**: Proactive issue identification and customer behavior prediction

Integration and Automation

- **Omnichannel Platform**: Unified customer experience across all communication channels
- **API-First Architecture**: Comprehensive REST and GraphQL APIs for seamless integrations
- **Workflow Automation**: Intelligent routing, escalation, and resolution workflows
- **Real-Time Processing**: Instant response generation and live agent collaboration

Enterprise Features

- **Scalability**: Cloud-native architecture supporting millions of concurrent interactions
- **Security**: End-to-end encryption, data privacy compliance, and secure authentication
- **Analytics**: Advanced reporting, performance monitoring, and business intelligence
- **Customization**: Configurable workflows, response templates, and branding options

Expected Business Outcomes

Operational Efficiency

- **Response Time Reduction**: 80% faster initial response times through AI automation
- **Resolution Rate Improvement**: 70% of queries resolved without human intervention
- **Agent Productivity**: 50% increase in agent efficiency through AI assistance tools
- **Cost Optimization**: 40% reduction in customer service operational costs

Customer Experience Enhancement

- **24/7 Availability**: Round-the-clock customer support across all channels
- **Personalized Service**: Tailored responses based on customer history and preferences
- **Proactive Support**: Anticipate and resolve issues before customer complaints
- **Satisfaction Improvement**: 25% increase in customer satisfaction scores

Business Intelligence

- **Customer Insights**: Deep analytics on customer behavior, preferences, and pain points

- **Performance Metrics**: Comprehensive reporting on service quality and operational efficiency
- **Predictive Analytics**: Forecasting customer service demand and resource requirements
- **Continuous Improvement**: Data-driven optimization of service processes and AI models

Implementation Strategy

Phase 1: Foundation (Months 1-3)

- Core AI chatbot development with basic NLP capabilities
- Multi-channel integration (email, chat, phone)
- Knowledge base integration and management system
- Basic analytics and reporting dashboard

Phase 2: Intelligence (Months 4-6)

- Advanced conversational AI with context retention
- Sentiment analysis and emotion-aware responses
- Intelligent escalation and routing algorithms
- Agent assistance tools and real-time suggestions

Phase 3: Optimization (Months 7-9)

- Predictive customer service capabilities
- Advanced analytics and business intelligence
- Multi-language support and localization
- Performance optimization and scalability enhancements

Phase 4: Enterprise (Months 10-12)

- Advanced integration ecosystem (CRM, ERP, marketing automation)
- Custom workflow automation and business rule engines
- Advanced security features and compliance certifications
- AI model fine-tuning and continuous learning optimization

Success Metrics

- **Response Time**: Average first response time < 30 seconds
- **Resolution Rate**: 70% automated resolution rate for common queries
- **Customer Satisfaction**: CSAT score > 4.5/5.0
- **Agent Efficiency**: 50% improvement in tickets handled per agent per hour
- **Cost Reduction**: 40% decrease in customer service operational costs
- **Availability**: 99.9% uptime across all channels
- **Accuracy**: 95% intent recognition accuracy for customer queries

This comprehensive e-commerce customer service AI platform will transform customer support operations through intelligent automation, personalized service delivery, and data-driven optimization while maintaining the human touch for complex interactions. # Product Requirements Document (PRD) ## E-commerce Customer Service AI - AI-Powered Intelligent Customer Service and Support Automation Platform

Building upon README foundation for comprehensive product specification

ETVX Framework

ENTRY CRITERIA

- âœ… README completed with problem overview, key requirements, and technical themes
- âœ… Problem statement analysis completed with business context understanding
- âœ… Stakeholder requirements gathered from e-commerce and customer service domains
- âœ… Market research completed on customer service automation trends

TASK

Define comprehensive product requirements including business objectives, market analysis, user personas, success metrics, feature specifications, technical requirements, constraints, assumptions, and risk assessment.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] Business objectives align with e-commerce customer service automation goals - [ ] Market analysis covers competitive landscape and differentiation opportunities - [ ] User personas represent all customer service stakeholder groups - [ ] Success metrics are measurable and aligned with business outcomes - [ ] Feature requirements are comprehensive and technically feasible

Validation Criteria: - [ ] Requirements validated with e-commerce business stakeholders - [ ] Technical feasibility validated with AI/ML and integration specialists - [ ] User experience validated with customer service managers and agents - [ ] Compliance requirements validated with legal and security teams

EXIT CRITERIA

- âœ… Complete PRD with business case, market analysis, and product specifications
- âœ… Detailed user personas and journey mapping
- âœ… Comprehensive feature requirements with acceptance criteria
- âœ… Technical requirements and architectural considerations
- âœ… Risk assessment and mitigation strategies
- âœ… Foundation established for FRD development

Reference to Previous Documents

This PRD builds upon the **README** foundation, expanding the problem overview into detailed product specifications with business justification, market analysis, and comprehensive requirements that will guide subsequent FRD, NFRD, and architectural design phases.

1. Business Objectives

1.1 Primary Business Goals

- **Customer Experience Transformation**: Deliver exceptional, personalized customer service experiences across all touchpoints with 24/7 availability and instant response capabilities
- **Operational Efficiency Optimization**: Reduce customer service operational costs by 40% while improving service quality through intelligent automation and AI-powered assistance
- **Scalability Achievement**: Support unlimited concurrent customer interactions without proportional increase in human agent requirements
- **Revenue Protection**: Minimize customer churn through proactive issue resolution and superior service quality
- **Data-Driven Insights**: Generate actionable business intelligence from customer interactions to drive product and service improvements

1.2 Strategic Alignment

- **Digital Transformation**: Accelerate e-commerce digital transformation through AI-powered customer service automation
- **Competitive Advantage**: Establish market leadership in customer service excellence through advanced AI capabilities
- **Customer Retention**: Increase customer lifetime value through superior service experiences and proactive support
- **Operational Excellence**: Achieve industry-leading efficiency metrics in customer service operations
- **Innovation Leadership**: Pioneer next-generation customer service technologies and methodologies

1.3 Success Metrics and KPIs

- **Response Time**: Average first response time < 30 seconds (vs.Â industry average 12 hours)
- **Resolution Rate**: 70% automated resolution rate for common queries (vs.Â current 20%)
- **Customer Satisfaction**: CSAT score > 4.5/5.0 (vs.Â current 3.8/5.0)
- **Agent Productivity**: 50% improvement in tickets handled per agent per hour
- **Cost Efficiency**: 40% reduction in customer service operational costs
- **Availability**: 99.9% uptime across all communication channels
- **Accuracy**: 95% intent recognition accuracy for customer queries
- **Customer Retention**: 15% improvement in customer retention rates

2. Market Analysis

2.1 Market Opportunity

- **Total Addressable Market (TAM)**: $24.3B global customer service software market
- **Serviceable Addressable Market (SAM)**: $8.7B AI-powered customer service automation market
- **Serviceable Obtainable Market (SOM)**: $435M e-commerce customer service AI market segment
- **Growth Rate**: 23.2% CAGR in AI customer service market through 2028
- **Market Drivers**: Digital transformation, customer experience expectations, operational cost pressures

2.2 Competitive Landscape

Direct Competitors: - **Zendesk Answer Bot**: Basic AI chatbot with limited e-commerce integration - **Salesforce Einstein Case Classification**: AI-powered case routing and classification - **Microsoft Dynamics 365 Customer Service**: Comprehensive platform with AI capabilities - **Freshworks Freddy AI**: Conversational AI with predictive insights

Competitive Advantages: - **E-commerce Specialization**: Purpose-built for e-commerce customer service workflows - **Advanced AI Capabilities**: State-of-the-art NLP and conversational AI technologies - **Comprehensive Integration**: Deep integration with e-commerce platforms and tools - **Predictive Analytics**: Proactive issue identification and resolution capabilities - **Multi-Channel Excellence**: Unified experience across all customer touchpoints

2.3 Market Positioning

- **Target Position**: Premium AI-powered customer service platform for enterprise e-commerce businesses
- **Value Proposition**: â€œTransform customer service from cost center to competitive advantage through intelligent automationâ€
- **Differentiation**: E-commerce-specific AI models, predictive customer service, and comprehensive integration ecosystem

3. User Personas and Stakeholders

3.1 Primary Personas

3.1.1 Customer Service Manager (Sarah)

Demographics: - Age: 35-45, 8+ years customer service management experience - Role: Oversees customer service operations for e-commerce company - Goals: Improve service quality, reduce costs, increase team productivity

Pain Points: - High operational costs and staffing challenges - Inconsistent service quality across channels - Difficulty scaling during peak periods - Limited visibility into performance metrics

Requirements: - Comprehensive analytics and reporting dashboard - Automated workflow management and escalation rules - Agent performance monitoring and coaching tools - Cost optimization and ROI tracking capabilities

3.1.2 Customer Service Agent (Mike)

Demographics: - Age: 25-35, 2-5 years customer service experience - Role: Handles customer inquiries across multiple channels - Goals: Resolve customer issues efficiently, meet performance targets

Pain Points: - Information scattered across multiple systems - Repetitive queries consuming time - Difficulty accessing relevant customer context - Pressure to meet response time targets

Requirements: - Unified customer information dashboard - AI-powered response suggestions and recommendations - Automated handling of routine inquiries - Real-time access to product and order information

3.1.3 E-commerce Business Owner (Lisa)

Demographics: - Age: 30-50, e-commerce business owner or executive - Role: Responsible for overall business performance and customer experience - Goals: Increase revenue, improve customer satisfaction, reduce operational costs

Pain Points: - Customer service costs impacting profitability - Customer complaints affecting brand reputation - Difficulty scaling customer service with business growth - Limited insights into customer service impact on business

Requirements: - Business impact analytics and ROI measurement - Customer satisfaction and retention metrics - Cost optimization and efficiency improvements - Integration with business intelligence systems

3.1.4 End Customer (David)

Demographics: - Age: 25-55, online shopping customer - Role: Purchases products through e-commerce platforms - Goals: Quick issue resolution, convenient service access, personalized support

Pain Points: - Long wait times for customer service response - Having to repeat information across channels - Difficulty finding relevant help information - Inconsistent service quality

Requirements: - Instant response to common queries - Seamless experience across all channels - Personalized recommendations and solutions - Self-service options for simple issues

3.2 Secondary Stakeholders

- **IT Administrators**: System integration, security, and maintenance
- **Data Analysts**: Performance metrics, reporting, and business intelligence
- **Compliance Officers**: Data privacy, security, and regulatory compliance
- **Product Managers**: Customer feedback integration and product improvement insights

4. Core Features and Requirements

4.1 Intelligent Conversation Management

Feature Description: Advanced AI-powered conversational interface with natural language understanding and context retention.

Requirements: - Multi-turn conversation handling with context preservation - Intent recognition with 95% accuracy for e-commerce queries - Entity extraction for products, orders, and customer information - Sentiment analysis with emotion-aware response adaptation - Multi-language support with automatic language detection

Acceptance Criteria: - System maintains conversation context for up to 50 exchanges - Intent classification accuracy exceeds 95% for trained domains - Response generation time under 2 seconds for 99% of queries - Sentiment analysis accuracy exceeds 90% for customer emotions

4.2 Omnichannel Integration Platform

Feature Description: Unified customer service experience across email, chat, phone, social media, and messaging platforms.

Requirements: - Real-time synchronization across all communication channels - Unified customer conversation history and context - Channel-specific response optimization and formatting - Seamless agent handoff between channels - Mobile-responsive interface for all channels

Acceptance Criteria: - Customer context available within 1 second across channel switches - 99.9% message delivery success rate across all channels - Zero data loss during channel transitions - Mobile interface usability score > 4.5/5.0

4.3 Intelligent Escalation and Routing

Feature Description: AI-powered decision engine for routing inquiries to appropriate resources based on complexity, urgency, and expertise requirements.

Requirements: - Automated complexity assessment and routing decisions - Skills-based routing to specialized agents - Priority queuing based on customer tier and issue urgency - Real-time agent availability and workload balancing - Escalation triggers based on sentiment and interaction patterns

Acceptance Criteria: - Routing accuracy exceeds 90% for appropriate resource assignment - Average queue time reduced by 60% compared to manual routing - Agent utilization optimization within 5% variance across team - Escalation triggers activate within 30 seconds of threshold breach

4.4 Knowledge Base Integration and Management

Feature Description: Dynamic access to comprehensive product information, policies, and troubleshooting guides with AI-powered content recommendations.

Requirements: - Real-time integration with product catalogs and inventory systems - Automated content updates and synchronization - AI-powered content recommendation based on query context - Version control and approval workflows for knowledge articles - Performance analytics for content effectiveness

Acceptance Criteria: - Knowledge base synchronization latency under 5 minutes - Content recommendation relevance score > 85% - Article effectiveness measured through resolution success rates - Content approval workflow completion within 24 hours

4.5 Predictive Customer Service

Feature Description: Proactive issue identification and resolution suggestions based on customer behavior patterns and predictive analytics.

Requirements: - Customer journey analysis and behavior pattern recognition - Proactive issue identification before customer complaints - Automated resolution suggestions and preventive measures - Risk scoring for customer satisfaction and churn prediction - Integration with marketing automation for proactive outreach

Acceptance Criteria: - Issue prediction accuracy exceeds 80% for common problems - Proactive resolution reduces incoming support tickets by 25% - Customer satisfaction improvement of 20% for proactive interventions - Churn prediction accuracy exceeds 85% for at-risk customers

5. Technical Requirements

5.1 Performance Requirements

- **Response Time**: API response time < 200ms for 95% of requests
- **Throughput**: Support 10,000+ concurrent conversations
- **Availability**: 99.9% uptime with maximum 4 hours downtime per month
- **Scalability**: Auto-scaling to handle 10x traffic spikes within 5 minutes
- **Data Processing**: Real-time processing of customer interactions and analytics

5.2 Integration Requirements

- **E-commerce Platforms**: Shopify, WooCommerce, Magento, BigCommerce, custom platforms
- **CRM Systems**: Salesforce, HubSpot, Microsoft Dynamics, Zoho CRM
- **Communication Channels**: Email (SMTP/IMAP), SMS, WhatsApp, Facebook Messenger, Slack
- **Analytics Platforms**: Google Analytics, Adobe Analytics, custom BI tools
- **Payment Systems**: Stripe, PayPal, Square, custom payment processors

5.3 Security and Compliance Requirements

- **Data Encryption**: AES-256 encryption at rest and TLS 1.3 in transit
- **Authentication**: Multi-factor authentication and SSO integration
- **Compliance**: GDPR, CCPA, PCI DSS, SOC 2 Type II, ISO 27001
- **Data Privacy**: Customer data anonymization and right to deletion
- **Audit Logging**: Comprehensive audit trails for all system activities

5.4 AI/ML Requirements

- **Natural Language Processing**: Advanced NLP with transformer-based models
- **Machine Learning**: Continuous learning from customer interactions
- **Model Performance**: Regular model evaluation and retraining pipelines
- **Bias Detection**: Automated bias detection and mitigation in AI responses
- **Explainability**: AI decision transparency for agent assistance

6. Constraints and Assumptions

6.1 Technical Constraints

- **Legacy System Integration**: Must integrate with existing e-commerce and CRM systems
- **Data Migration**: Seamless migration from existing customer service platforms
- **Bandwidth Limitations**: Optimize for varying internet connection speeds
- **Mobile Compatibility**: Full functionality on mobile devices and tablets
- **Browser Support**: Support for all major browsers including legacy versions

6.2 Business Constraints

- **Budget Limitations**: Development within allocated budget parameters
- **Timeline Constraints**: Phased delivery approach to meet market timing requirements
- **Regulatory Compliance**: Adherence to industry-specific regulations and standards
- **Resource Availability**: Development team size and expertise limitations
- **Market Competition**: Rapid feature development to maintain competitive advantage

6.3 Assumptions

- **Customer Adoption**: Customers will adopt AI-powered service interactions
- **Data Availability**: Sufficient historical data for AI model training
- **Integration Cooperation**: Third-party vendors will provide necessary API access
- **Technology Evolution**: AI/ML technologies will continue advancing during development
- **Market Demand**: Sustained market demand for AI customer service solutions

7. Risk Assessment and Mitigation

7.1 Technical Risks

High Risk: AI Model Performance - **Risk**: Insufficient accuracy in intent recognition and response generation - **Impact**: Poor customer experience and increased escalations - **Mitigation**: Extensive training data collection, continuous model improvement, human-in-the-loop validation

Medium Risk: Integration Complexity - **Risk**: Challenges integrating with diverse e-commerce platforms - **Impact**: Delayed deployment and limited functionality - **Mitigation**: Standardized API development, comprehensive testing, phased integration approach

7.2 Business Risks

High Risk: Customer Acceptance - **Risk**: Customers may prefer human agents over AI interactions - **Impact**: Lower adoption rates and ROI realization - **Mitigation**: Gradual AI introduction, transparent AI capabilities, seamless human handoff

Medium Risk: Competitive Response - **Risk**: Competitors may launch similar solutions during development - **Impact**: Reduced market differentiation and pricing pressure - **Mitigation**: Accelerated development timeline, unique feature development, patent protection

7.3 Operational Risks

Medium Risk: Data Privacy Compliance - **Risk**: Evolving privacy regulations may impact data usage - **Impact**: Feature limitations and compliance costs - **Mitigation**: Privacy-by-design architecture, legal consultation, flexible data handling

Low Risk: Scalability Challenges - **Risk**: System performance degradation under high load - **Impact**: Service disruptions and customer dissatisfaction - **Mitigation**: Cloud-native architecture, load testing, auto-scaling implementation

8. Success Criteria and Validation

8.1 Business Success Criteria

- **Revenue Impact**: 15% increase in customer lifetime value within 12 months
- **Cost Reduction**: 40% decrease in customer service operational costs
- **Customer Satisfaction**: CSAT score improvement from 3.8 to 4.5+
- **Market Position**: Top 3 market position in e-commerce customer service AI
- **ROI Achievement**: 300% ROI within 18 months of deployment

8.2 Technical Success Criteria

- **Performance Benchmarks**: All performance requirements consistently met
- **Integration Success**: Seamless integration with 95% of target platforms
- **AI Accuracy**: Intent recognition and response quality exceeding targets
- **System Reliability**: 99.9% uptime achievement with minimal downtime
- **Security Compliance**: Full compliance with all security and privacy requirements

8.3 User Acceptance Criteria

- **Agent Satisfaction**: Agent productivity improvement and satisfaction scores > 4.0/5.0
- **Customer Experience**: Customer effort score reduction and satisfaction improvement
- **Manager Adoption**: 90% of customer service managers actively using analytics features
- **Business User Value**: Measurable business impact and ROI demonstration

This comprehensive PRD establishes the foundation for developing an industry-leading e-commerce customer service AI platform that transforms customer support operations through intelligent automation while maintaining exceptional service quality and customer satisfaction. # Functional Requirements Document (FRD) ## E-commerce Customer Service AI - AI-Powered Intelligent Customer Service and Support Automation Platform

Building upon README and PRD foundations for detailed functional specifications

ETVX Framework

ENTRY CRITERIA

- âœ… README completed with problem overview and technical approach
- âœ… PRD completed with business objectives, market analysis, user personas, and success metrics
- âœ… Stakeholder requirements validated with customer service managers and e-commerce teams

TASK

Define detailed functional requirements covering system behaviors, user interactions, AI/ML capabilities, integration interfaces, and acceptance criteria.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] Functional requirements align with PRD business objectives - [ ] AI/ML capabilities meet performance targets - [ ] Integration requirements support all specified platforms

Validation Criteria: - [ ] Requirements validated with customer service operations teams - [ ] AI/ML specifications validated with data science teams - [ ] Integration requirements validated with platform specialists

EXIT CRITERIA

- âœ… Complete functional requirements for all system modules
- âœ… Detailed user interaction flows and system behaviors
- âœ… AI/ML processing requirements with accuracy specifications
- âœ… Foundation established for NFRD and architectural design

---

1. Conversation Management Module

1.1 Multi-Channel Conversation Handling

Requirement ID: FR-CM-001 **Description**: System shall provide unified conversation management across all communication channels.

Functional Specifications: - Support email, chat, phone, social media, and messaging platforms - Real-time message processing with <500ms latency - Conversation threading across channel switches - Channel-specific response formatting - Message history preservation for 30+ days

Acceptance Criteria: - Message processing latency <500ms for 99% of messages - Zero message loss during channel transitions - Support 50+ concurrent conversations per agent

1.2 Natural Language Understanding

Requirement ID: FR-CM-002 **Description**: Advanced NLP for intent recognition and entity extraction.

Functional Specifications: - Intent classification with 95% accuracy - Entity extraction for products, orders, customer info - Context preservation across conversation turns - Real-time sentiment analysis - Multi-language support (25+ languages)

Acceptance Criteria: - Intent recognition accuracy â‰¥95% - Entity extraction precision â‰¥90%, recall â‰¥85% - Sentiment analysis accuracy â‰¥90%

1.3 Intelligent Response Generation

Requirement ID: FR-CM-003 **Description**: AI-powered contextual response generation.

Functional Specifications: - Human-like responses using fine-tuned language models - Personalization based on customer history - Brand voice consistency - Multi-modal responses (text, images, actions) - Confidence scoring for escalation decisions

Acceptance Criteria: - Response generation time <2 seconds - Response relevance score â‰¥85% - Customer satisfaction â‰¥4.0/5.0 for AI responses

2. Knowledge Management Module

2.1 Dynamic Knowledge Base Integration

Requirement ID: FR-KM-001 **Description**: Real-time access to product information and policies.

Functional Specifications: - Product catalog synchronization - Policy and procedure management - Troubleshooting guide integration - Content versioning and approval workflows - Semantic search capabilities

Acceptance Criteria: - Synchronization latency <5 minutes - Search relevance score â‰¥90% - Content approval within 24 hours

2.2 Automated Content Recommendations

Requirement ID: FR-KM-002 **Description**: AI-powered content recommendations based on context.

Functional Specifications: - Contextual matching algorithms - Similarity scoring and ranking - Multi-source information aggregation - Real-time recommendation updates - Performance tracking and optimization

Acceptance Criteria: - Recommendation relevance â‰¥85% - Response time <1 second - Content utilization rate â‰¥70%

3. Intelligent Routing Module

3.1 Automated Query Classification

Requirement ID: FR-RE-001 **Description**: Automatic routing based on complexity and expertise.

Functional Specifications: - Complexity assessment algorithms - Skills-based agent matching - Priority queuing by customer tier - Load balancing optimization - Escalation trigger monitoring

Acceptance Criteria: - Routing accuracy â‰¥90% - Queue time reduction â‰¥60% - Agent utilization within 5% variance

3.2 Dynamic Escalation Management

Requirement ID: FR-RE-002 **Description**: Intelligent escalation with seamless handoff.

Functional Specifications: - Multi-criteria escalation triggers - Complete context transfer - Warm handoff capabilities - Escalation analytics - De-escalation support tools

Acceptance Criteria: - Escalation accuracy â‰¥85% - Context transfer completeness â‰¥95% - Handoff time <30 seconds

4. Order Management Integration

4.1 Real-Time Order Access

Requirement ID: FR-OM-001 **Description**: Instant order information retrieval.

Functional Specifications: - Multi-field order lookup - Real-time status tracking - Shipping carrier integration - Payment status access - Inventory synchronization

Acceptance Criteria: - Retrieval time <2 seconds - Data accuracy â‰¥99.5% - Synchronization latency <5 minutes

4.2 Automated Order Operations

Requirement ID: FR-OM-002 **Description**: Automated order modifications and processing.

Functional Specifications: - Order modification automation - Cancellation processing - Refund automation - Return management - Business rules engine

Acceptance Criteria: - Operation success rate â‰¥95% - Processing time <30 seconds - Business rule compliance â‰¥99.9%

5. Analytics and Reporting

5.1 Real-Time Performance Monitoring

Requirement ID: FR-AR-001 **Description**: Comprehensive performance metrics monitoring.

Functional Specifications: - Live performance dashboards - Automated alert system - Trend analysis capabilities - Comparative analytics - Drill-down functionality

Acceptance Criteria: - Dashboard refresh â‰¤30 seconds - Alert delivery <1 minute - Report generation <5 seconds

5.2 Customer Journey Analytics

Requirement ID: FR-AR-002 **Description**: Customer interaction analysis and optimization.

Functional Specifications: - Journey mapping visualization - Behavior pattern analysis - Satisfaction correlation analysis - Predictive insights modeling - Customer segmentation

Acceptance Criteria: - Journey analysis <10 seconds - Prediction accuracy â‰¥85% - Insight actionability â‰¥80%

6. Integration and API Module

6.1 E-commerce Platform Integration

Requirement ID: FR-IA-001 **Description**: Seamless integration with major platforms.

Functional Specifications: - Pre-built platform connectors - Custom API support - Real-time data synchronization - Webhook integration - Secure authentication

Acceptance Criteria: - Integration setup <4 hours - Synchronization latency <5 minutes - API response time <200ms

6.2 Third-Party Service Integration

Requirement ID: FR-IA-002 **Description**: Integration with CRM and business applications.

Functional Specifications: - CRM bidirectional sync - Communication platform APIs - Analytics tool integration - Marketing automation connectivity - Help desk compatibility

Acceptance Criteria: - Support 20+ platforms - Data mapping accuracy â‰¥99% - Integration reliability â‰¥99.9%

7. Security and Compliance

7.1 Data Protection

Requirement ID: FR-SC-001 **Description**: Comprehensive data protection and privacy.

Functional Specifications: - AES-256 encryption at rest, TLS 1.3 in transit - Role-based access control - Data anonymization capabilities - Consent management - Configurable retention policies

Acceptance Criteria: - 100% sensitive data encryption - Access control accuracy â‰¥99.9% - Anonymization effectiveness â‰¥95%

7.2 Audit and Compliance

Requirement ID: FR-SC-002 **Description**: Audit logging and compliance monitoring.

Functional Specifications: - Complete audit trail logging - Automated compliance monitoring - Security event detection - Compliance reporting - Incident management workflows

Acceptance Criteria: - Audit log completeness â‰¥99.9% - Compliance monitoring accuracy â‰¥95% - Security incident detection <5 minutes

This FRD establishes detailed functional specifications for the e-commerce customer service AI platform, ensuring comprehensive coverage of all system capabilities and user requirements. # Non-Functional Requirements Document (NFRD) ## E-commerce Customer Service AI - AI-Powered Intelligent Customer Service and Support Automation Platform

Building upon README, PRD, and FRD foundations for comprehensive non-functional specifications

ETVX Framework

ENTRY CRITERIA

- âœ… README completed with problem overview and technical approach
- âœ… PRD completed with business objectives, market analysis, and success metrics
- âœ… FRD completed with 21 detailed functional requirements across 7 modules
- âœ… Technical architecture considerations identified from functional requirements

TASK

Define comprehensive non-functional requirements covering performance, scalability, reliability, security, usability, compliance, maintainability, and operational requirements.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] Performance requirements align with FRD response time specifications - [ ] Scalability requirements support projected user loads and growth - [ ] Security requirements meet enterprise and compliance standards - [ ] Usability requirements ensure optimal user experience for all personas

Validation Criteria: - [ ] Performance requirements validated with infrastructure and DevOps teams - [ ] Security requirements validated with information security and compliance teams - [ ] Scalability requirements validated with system architects and capacity planners - [ ] Usability requirements validated with UX designers and end users

EXIT CRITERIA

- âœ… Complete non-functional requirements for all quality attributes
- âœ… Performance benchmarks and scalability targets defined
- âœ… Security and compliance requirements specified
- âœ… Operational and maintainability requirements established
- âœ… Foundation prepared for architectural design phase

Reference to Previous Documents

This NFRD builds upon **README** technical themes, **PRD** business objectives and constraints, and **FRD** functional specifications to define the quality attributes and operational characteristics that ensure the system meets enterprise-grade performance, security, and reliability standards.

1. Performance Requirements

1.1 Response Time Requirements

Requirement ID: NFR-PERF-001 **Priority**: Critical **Description**: System shall provide fast response times across all user interactions and API operations.

Specifications: - **UI Response Time**: Web interface responds within 1 second for 95% of user actions - **API Response Time**: REST API responses within 200ms for 95% of requests, 500ms for 99% - **AI Response Generation**: Conversational AI generates responses within 2 seconds for 99% of queries - **Knowledge Base Search**: Search results returned within 1 second for 95% of queries - **Order Lookup**: Order information retrieval within 2 seconds for 99% of requests - **Real-time Updates**: Live dashboard updates with maximum 30-second latency

Measurement Criteria: - Response times measured from client request initiation to complete response delivery - Performance monitoring with 1-minute granularity - 99.5% of all operations must complete within specified timeframes - Performance degradation alerts triggered at 80% of threshold limits

1.2 Throughput Requirements

Requirement ID: NFR-PERF-002 **Priority**: Critical **Description**: System shall handle high-volume concurrent operations without performance degradation.

Specifications: - **Concurrent Users**: Support 10,000+ concurrent active users - **Message Processing**: Handle 50,000+ messages per minute across all channels - **API Throughput**: Process 100,000+ API requests per minute - **Database Operations**: Execute 500,000+ database transactions per minute - **AI Model Inference**: Generate 10,000+ AI responses per minute - **Real-time Analytics**: Process 1M+ events per minute for analytics

Load Testing Criteria: - Sustained performance under 150% of expected peak load - Performance degradation <10% under maximum specified load - Recovery time <2 minutes after load spike resolution - Zero data loss during high-load periods

1.3 Resource Utilization Requirements

Requirement ID: NFR-PERF-003 **Priority**: High **Description**: System shall optimize resource utilization for cost-effective operations.

Specifications: - **CPU Utilization**: Average CPU usage <70% under normal load, <90% under peak load - **Memory Usage**: Memory utilization <80% with

automatic garbage collection optimization - **Storage I/O**: Disk I/O operations optimized with <5ms average latency - **Network Bandwidth**: Efficient bandwidth usage with compression and caching - **Database Connections**: Connection pooling with <100ms connection acquisition time - **Cache Hit Ratio**: Minimum 85% cache hit ratio for frequently accessed data

2. Scalability Requirements

2.1 Horizontal Scalability

Requirement ID: NFR-SCALE-001 **Priority**: Critical **Description**: System shall scale horizontally to accommodate growth in users, data, and transactions.

Specifications: - **Auto-scaling**: Automatic horizontal scaling based on CPU, memory, and request metrics - **Load Distribution**: Even load distribution across multiple instances with <5% variance - **Stateless Design**: Stateless application components enabling seamless scaling - **Database Scaling**: Read replicas and sharding support for database scalability - **Microservices Architecture**: Independent scaling of individual service components - **Container Orchestration**: Kubernetes-based orchestration with automatic pod scaling

Scaling Targets: - Scale from 1,000 to 100,000 concurrent users within 10 minutes - Support 10x traffic spikes with automatic scaling response within 5 minutes - Linear performance scaling with additional compute resources - Zero-downtime scaling operations

2.2 Data Scalability

Requirement ID: NFR-SCALE-002 **Priority**: High **Description**: System shall handle massive data volumes with consistent performance.

Specifications: - **Data Volume**: Support 100TB+ of customer interaction data - **Database Growth**: Handle 1TB+ monthly data growth with performance consistency - **Archive Strategy**: Automated data archiving with configurable retention policies - **Search Scalability**: Elasticsearch clusters supporting billions of documents - **Analytics Data**: Real-time processing of 10M+ events per hour - **Backup Scalability**: Incremental backups completing within 4-hour windows

3. Reliability and Availability Requirements

3.1 System Availability

Requirement ID: NFR-REL-001 **Priority**: Critical **Description**: System shall provide high availability with minimal downtime.

Specifications: - **Uptime Target**: 99.9% availability (maximum 8.77 hours downtime per year) - **Service Level Agreement**: 99.5% availability during business hours (8 AM - 8 PM) - **Planned Maintenance**: Maximum 4 hours monthly maintenance window - **Recovery Time Objective (RTO)**: System recovery within 15 minutes of failure - **Recovery Point Objective (RPO)**: Maximum 5 minutes of data loss acceptable - **Geographic Redundancy**: Multi-region deployment with automatic failover

High Availability Features: - Active-active deployment across multiple availability zones - Database replication with automatic failover - Load balancer health checks with 30-second intervals - Circuit breaker patterns for external service dependencies - Graceful degradation during partial system failures

3.2 Fault Tolerance

Requirement ID: NFR-REL-002 **Priority**: High **Description**: System shall continue operating despite component failures.

Specifications: - **Component Redundancy**: No single points of failure in critical system components - **Graceful Degradation**: Reduced functionality rather than complete failure - **Error Handling**: Comprehensive error handling with user-friendly messages - **Retry Mechanisms**: Exponential backoff retry logic for transient failures - **Timeout Management**: Configurable timeouts preventing resource exhaustion - **Health Monitoring**: Continuous health checks with automatic remediation

3.3 Data Integrity and Consistency

Requirement ID: NFR-REL-003 **Priority**: Critical **Description**: System shall maintain data integrity and consistency across all operations.

Specifications: - **ACID Compliance**: Database transactions maintain ACID properties - **Data Validation**: Input validation preventing data corruption - **Backup Verification**: Regular backup integrity verification - **Consistency Checks**: Automated data consistency validation - **Audit Trail**: Complete audit trail for all data modifications - **Conflict Resolution**: Automated conflict resolution for concurrent updates

4. Security Requirements

4.1 Authentication and Authorization

Requirement ID: NFR-SEC-001 **Priority**: Critical **Description**: System shall implement robust authentication and authorization mechanisms.

Specifications: - **Multi-Factor Authentication**: MFA required for all administrative accounts - **Single Sign-On (SSO)**: SAML 2.0 and OAuth 2.0 SSO integration - **Role-Based Access Control**: Granular RBAC with principle of least privilege - **Session Management**: Secure session handling with automatic timeout - **Password Policy**: Strong password requirements with regular rotation - **Account Lockout**: Automatic account lockout after failed login attempts

Authentication Standards: - Support for LDAP, Active Directory, and cloud identity providers - JWT token-based authentication with secure key management - API key authentication for system integrations - Certificate-based authentication for high-security environments

4.2 Data Protection

Requirement ID: NFR-SEC-002 **Priority**: Critical **Description**: System shall protect sensitive data through encryption and access controls.

Specifications: - **Encryption at Rest**: AES-256 encryption for all stored data - **Encryption in Transit**: TLS 1.3 for all network communications - **Key Management**: Hardware Security Module (HSM) for encryption key management - **Data Masking**: PII masking in non-production environments - **Secure Deletion**: Cryptographic erasure for data deletion requirements - **Field-Level Encryption**: Sensitive fields encrypted at application level

4.3 Network Security

Requirement ID: NFR-SEC-003 **Priority**: High **Description**: System shall implement comprehensive network security measures.

Specifications: - **Firewall Protection**: Web Application Firewall (WAF) with DDoS protection - **Network Segmentation**: Micro-segmentation with zero-trust architecture - **VPN Access**: Secure VPN access for administrative functions - **Intrusion Detection**: Real-time intrusion detection and prevention - **API Security**: API rate limiting, throttling, and abuse prevention - **Security Monitoring**: 24/7 security monitoring with incident response

5. Compliance Requirements

5.1 Data Privacy Compliance

Requirement ID: NFR-COMP-001 **Priority**: Critical **Description**: System shall comply with data privacy regulations and standards.

Specifications: - **GDPR Compliance**: Full compliance with EU General Data Protection Regulation - **CCPA Compliance**: California Consumer Privacy Act compliance - **Data Subject Rights**: Implementation of data subject access, portability, and deletion rights - **Consent Management**: Granular consent management and tracking - **Privacy by Design**: Privacy considerations integrated into system architecture - **Data Processing Records**: Comprehensive records of data processing activities

5.2 Industry Standards Compliance

Requirement ID: NFR-COMP-002 **Priority**: High **Description**: System shall comply with relevant industry standards and certifications.

Specifications: - **SOC 2 Type II**: Service Organization Control 2 Type II certification - **ISO 27001**: Information Security Management System certification - **PCI**

DSS: Payment Card Industry Data Security Standard compliance - **HIPAA**: Health Insurance Portability and Accountability Act compliance (if applicable) - **FedRAMP**: Federal Risk and Authorization Management Program (if applicable)

6. Usability Requirements

6.1 User Experience

Requirement ID: NFR-USE-001 **Priority**: High **Description**: System shall provide intuitive and efficient user experience.

Specifications: - **Learning Curve**: New users productive within 2 hours of training - **Task Completion**: 90% task completion rate for primary workflows - **Error Prevention**: Proactive error prevention with validation and warnings - **Help System**: Contextual help and documentation integrated into interface - **Accessibility**: WCAG 2.1 AA compliance for accessibility - **Mobile Responsiveness**: Full functionality on mobile devices and tablets

6.2 Interface Design

Requirement ID: NFR-USE-002 **Priority**: Medium **Description**: System shall provide consistent and professional interface design.

Specifications: - **Design Consistency**: Consistent UI patterns and visual elements - **Brand Customization**: Customizable branding and white-label options - **Dark Mode**: Support for light and dark interface themes - **Internationalization**: Multi-language interface support - **Keyboard Navigation**: Full keyboard navigation support - **Screen Reader Support**: Compatible with assistive technologies

7. Maintainability Requirements

7.1 System Maintainability

Requirement ID: NFR-MAINT-001 **Priority**: Medium **Description**: System shall be designed for easy maintenance and updates.

Specifications: - **Modular Architecture**: Loosely coupled components enabling independent updates - **Configuration Management**: Externalized configuration with environment-specific settings - **Logging and Monitoring**: Comprehensive logging with structured log formats - **Documentation**: Complete technical documentation with regular updates - **Code Quality**: Automated code quality checks and testing requirements - **Deployment Automation**: Automated deployment pipelines with rollback capabilities

7.2 Operational Requirements

Requirement ID: NFR-MAINT-002 **Priority**: Medium **Description**: System shall support efficient operational management.

Specifications: - **Monitoring Dashboards**: Comprehensive operational monitoring dashboards - **Alerting System**: Intelligent alerting with escalation procedures - **Backup and Recovery**: Automated backup with tested recovery procedures - **Performance Tuning**: Built-in performance monitoring and optimization tools - **Capacity Planning**: Automated capacity monitoring and planning tools - **Update Management**: Zero-downtime update deployment capabilities

8. Integration Requirements

8.1 API Performance

Requirement ID: NFR-INT-001 **Priority**: High **Description**: System APIs shall provide consistent performance and reliability.

Specifications: - **API Response Time**: 95% of API calls complete within 200ms - **API Availability**: 99.9% API availability with comprehensive error handling - **Rate Limiting**: Configurable rate limiting with graceful degradation - **API Versioning**: Backward-compatible API versioning strategy - **Documentation**: Complete API documentation with interactive examples - **SDK Support**: Official SDKs for major programming languages

8.2 Third-Party Integration

Requirement ID: NFR-INT-002 **Priority**: Medium **Description**: System shall integrate reliably with external services and platforms.

Specifications: - **Integration Resilience**: Graceful handling of third-party service failures - **Timeout Management**: Configurable timeouts for external service calls - **Retry Logic**: Intelligent retry mechanisms with exponential backoff - **Circuit Breaker**: Circuit breaker pattern for external service protection - **Monitoring**: Comprehensive monitoring of external service integrations - **Fallback Mechanisms**: Fallback options when external services are unavailable

This comprehensive NFRD establishes the quality attributes and operational characteristics required for an enterprise-grade e-commerce customer service AI platform, ensuring optimal performance, security, reliability, and user experience. # Architecture Diagram ## E-commerce Customer Service AI - AI-Powered Intelligent Customer Service and Support Automation Platform

Building upon README, PRD, FRD, and NFRD foundations for comprehensive architectural design

ETVX Framework

ENTRY CRITERIA

- âœ… README completed with problem overview and technical approach
- âœ… PRD completed with business objectives, market analysis, and success metrics
- âœ… FRD completed with 21 detailed functional requirements across 7 modules
- âœ… NFRD completed with 24 non-functional requirements covering performance, security, and scalability
- âœ… Technical constraints and integration requirements identified

TASK

Design comprehensive system architecture including microservices design, AI/ML pipeline, data architecture, integration patterns, security framework, and cloud-native deployment strategy.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] Architecture supports all functional requirements from FRD - [ ] Design meets non-functional requirements for performance and scalability - [ ] Security architecture addresses all compliance and protection requirements - [ ] Integration architecture supports all specified platforms and APIs

Validation Criteria: - [ ] Architecture validated with system architects and technical leads - [ ] AI/ML pipeline validated with data science and ML engineering teams - [ ] Security design validated with information security teams - [ ] Integration patterns validated with platform and API specialists

EXIT CRITERIA

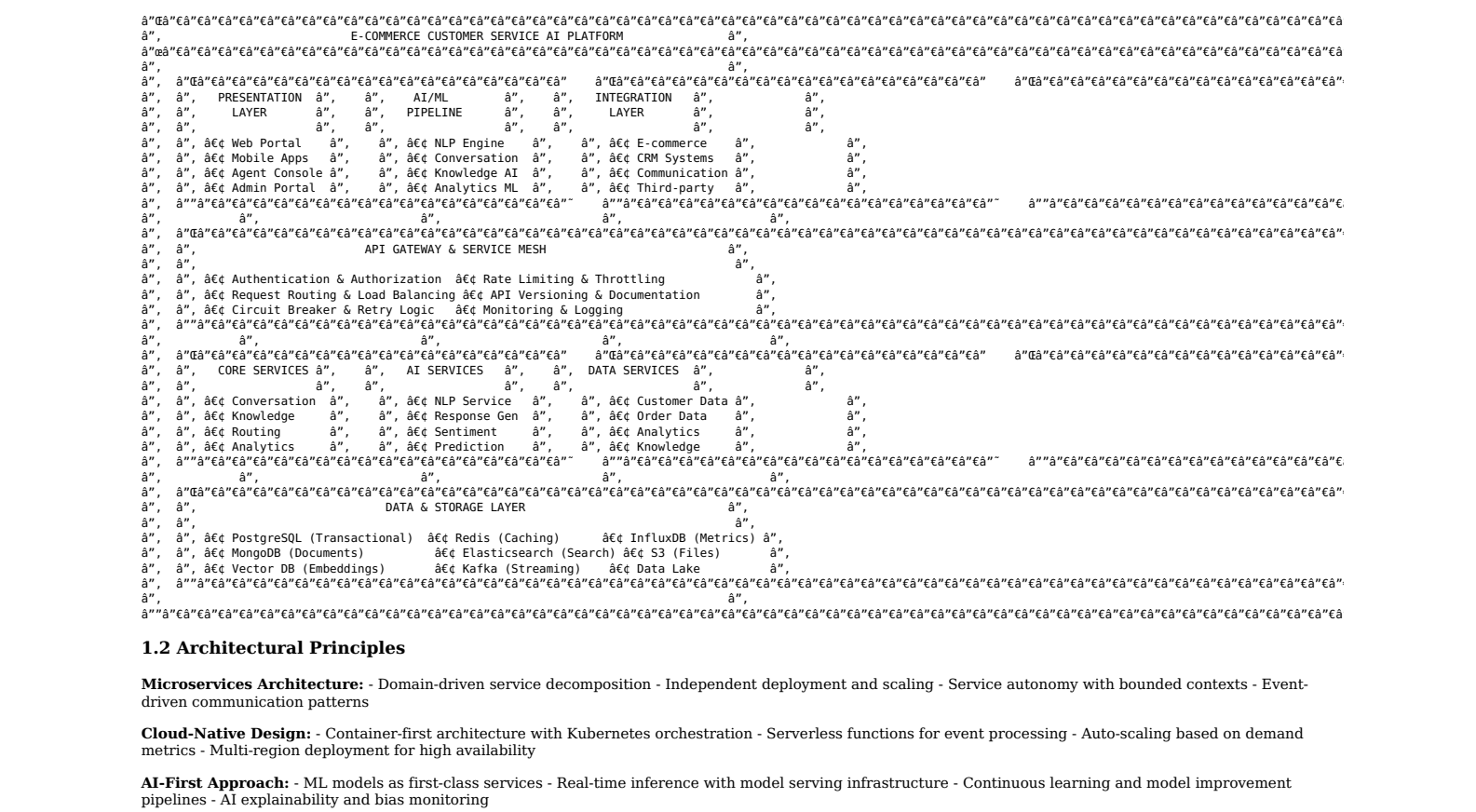
- âœ… Complete system architecture with all components and interactions
- âœ… AI/ML pipeline architecture with model serving and training workflows
- âœ… Data architecture with storage, processing, and analytics layers
- âœ… Security and compliance architecture framework
- âœ… Cloud-native deployment architecture with scalability and reliability
- âœ… Foundation established for HLD development

Reference to Previous Documents

This Architecture Diagram builds upon **README** technical themes, **PRD** business requirements, **FRD** functional specifications, and **NFRD** quality attributes to create a comprehensive system architecture that supports intelligent customer service automation with enterprise-grade performance, security, and scalability.

1. System Architecture Overview

1.1 High-Level Architecture



1.2 Architectural Principles

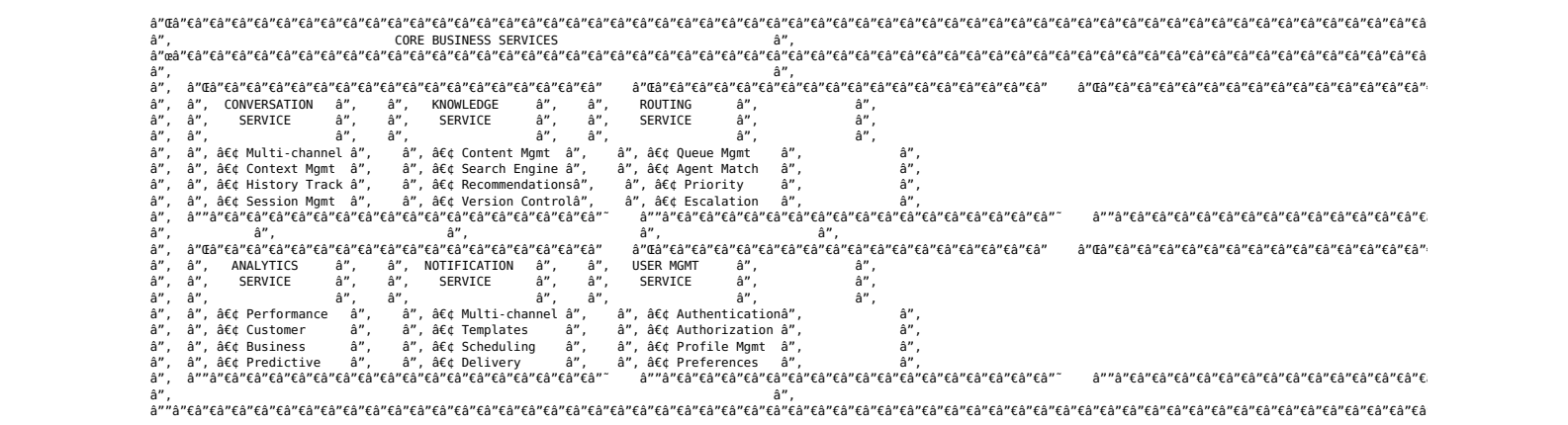
Microservices Architecture: - Domain-driven service decomposition - Independent deployment and scaling - Service autonomy with bounded contexts - Event-driven communication patterns

Cloud-Native Design: - Container-first architecture with Kubernetes orchestration - Serverless functions for event processing - Auto-scaling based on demand metrics - Multi-region deployment for high availability

AI-First Approach: - ML models as first-class services - Real-time inference with model serving infrastructure - Continuous learning and model improvement pipelines - AI explainability and bias monitoring

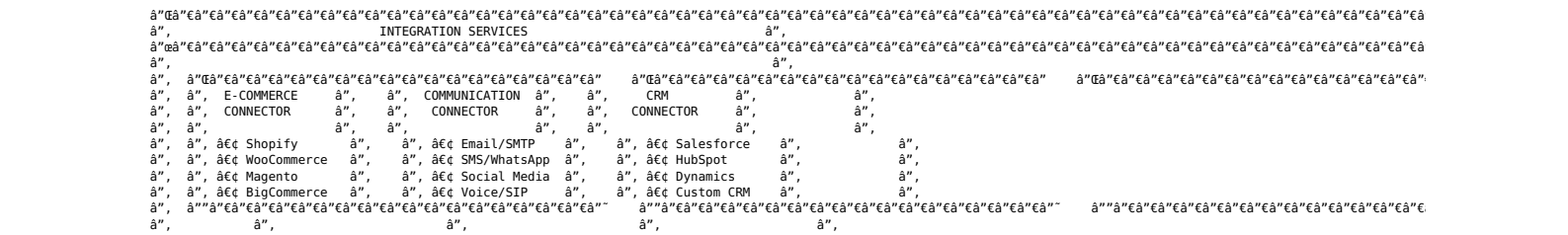
2. Microservices Architecture

2.1 Core Business Services



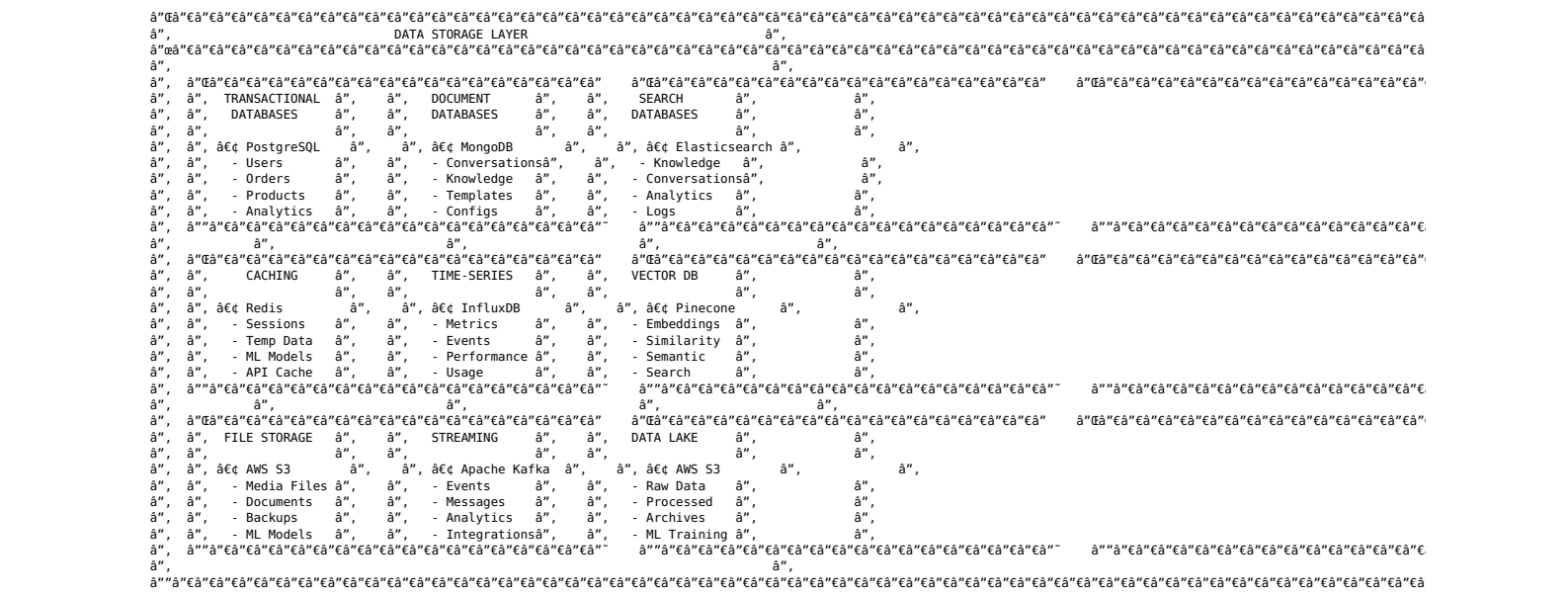
2.2 AI/ML Services Architecture

2.3 Integration Services



3. Data Architecture

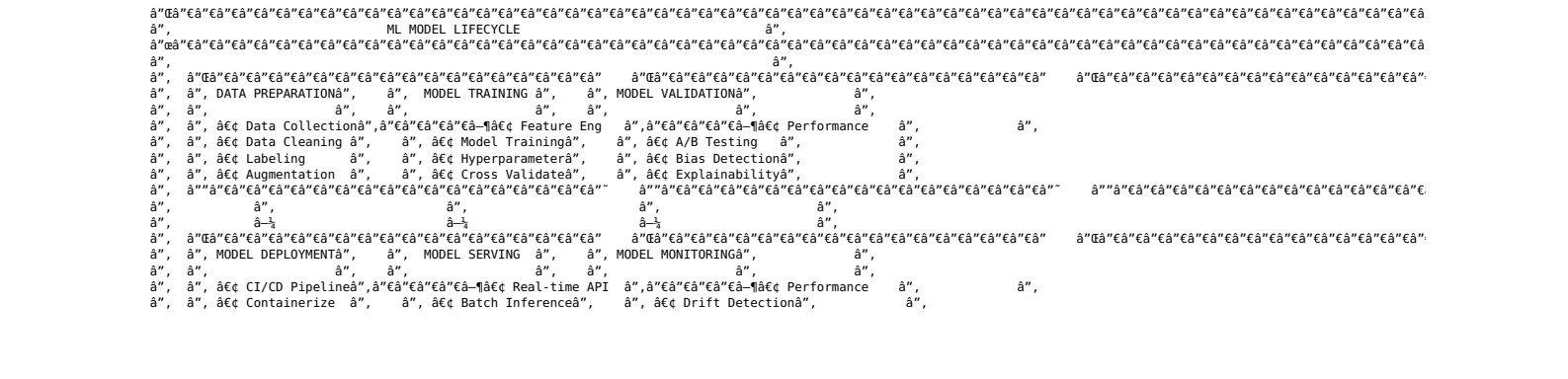
3.1 Data Storage Strategy

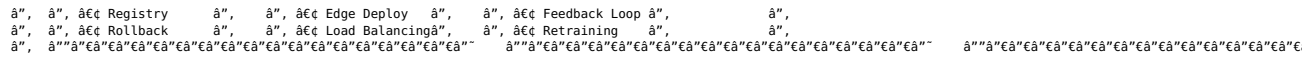


3.2 Data Flow Architecture

4. AI/ML Pipeline Architecture

4.1 ML Model Lifecycle





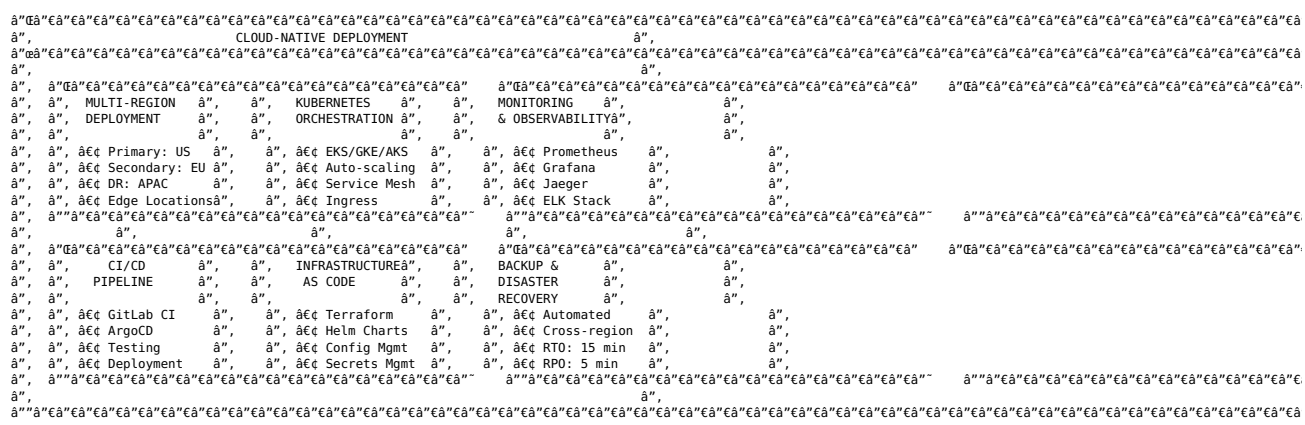
4.2 AI Model Architecture

5. Security Architecture

5.1 Security Framework

6. Deployment Architecture

6.1 Cloud-Native Deployment



This comprehensive architecture provides a robust, scalable, and secure foundation for the e-commerce customer service AI platform, supporting all functional and non-functional requirements while enabling future growth and innovation. # High Level Design (HLD) ## E-commerce Customer Service AI - AI-Powered Intelligent Customer Service and Support Automation Platform

Building upon README, PRD, FRD, NFRD, and AD foundations for detailed component specifications

ETVX Framework

ENTRY CRITERIA

- âœ… README completed with problem overview and technical approach
- âœ… PRD completed with business objectives, market analysis, and success metrics
- âœ… FRD completed with 21 detailed functional requirements across 7 modules
- âœ… NFRD completed with 24 non-functional requirements covering performance, security, and scalability
- âœ… AD completed with microservices architecture and cloud-native deployment strategy
- âœ… System architecture validated with technical stakeholders

TASK

Design detailed component specifications including API interfaces, data models, processing workflows, AI/ML architectures, integration patterns, and deployment configurations.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] Component designs align with architectural patterns from AD - [ ] API specifications meet functional requirements from FRD - [ ] Data models support all required operations and integrations - [ ] AI/ML workflows meet performance and accuracy targets from NFRD

Validation Criteria: - [ ] Component specifications validated with development teams - [ ] API designs validated with integration and frontend teams - [ ] AI/ML architectures validated with data science teams - [ ] Performance specifications validated with infrastructure teams

EXIT CRITERIA

- âœ… Complete component specifications for all system modules
- âœ… Detailed API interfaces and data model definitions
- âœ… AI/ML processing workflows and model architectures
- âœ… Integration patterns and deployment configurations
- âœ… Foundation established for LLD implementation details

Reference to Previous Documents

This HLD builds upon **README** technical approach, **PRD** business requirements, **FRD** functional specifications, **NFRD** quality attributes, and **AD** system architecture to provide detailed component designs that enable implementation teams to build the e-commerce customer service AI platform.

1. Core Services Component Design

1.1 Conversation Service

Component Overview: Central service managing multi-channel customer conversations with context preservation and real-time processing capabilities.

API Interface:

```
ConversationService:
  endpoints:
    - POST /conversations
      description: Create new conversation
      request: ConversationCreateRequest
      response: ConversationResponse

    - GET /conversations/{id}
      description: Retrieve conversation details
      response: ConversationDetailResponse

    - POST /conversations/{id}/messages
      description: Add message to conversation
      request: MessageRequest
      response: MessageResponse

    - PUT /conversations/{id}/status
      description: Update conversation status
      request: StatusUpdateRequest

    - GET /conversations/{id}/context
      description: Get conversation context
      response: ConversationContextResponse
```

Data Models:

```
Conversation:
  id: string (UUID)
  customer_id: string
  channel: enum [email, chat, phone, social, sms]
  status: enum [active, resolved, escalated, closed]
  priority: enum [low, medium, high, critical]
  created_at: datetime
  updated_at: datetime
  metadata: object

Message:
  id: string (UUID)
  conversation_id: string
  sender_type: enum [customer, agent, ai]
  sender_id: string
  content: string
  message_type: enum [text, image, file, system]
  timestamp: datetime
  metadata: object

ConversationContext:
  conversation_id: string
  customer_profile: CustomerProfile
  interaction_history: List[Message]
  current_intent: string
  sentiment_score: float
  context_variables: object
```

Processing Workflows: 1. **Message Processing Pipeline:** - Receive message from channel adapter - Validate and sanitize content - Extract metadata and context - Apply NLP processing for intent/sentiment - Store in conversation history - Trigger response generation or routing

2. **Context Management:**
    - Maintain conversation state across channels
    - Track customer journey and preferences
    - Preserve context for up to 50 conversation turns
    - Implement context compression for long conversations

Performance Specifications: - Message processing latency: <500ms for 99% of messages - Context retrieval time: <100ms - Concurrent conversation support: 50,000+ active conversations - Message throughput: 100,000+ messages per minute

1.2 Knowledge Service

Component Overview: Intelligent knowledge management system providing real-time access to product information, policies, and troubleshooting guides with AI-powered recommendations.

API Interface:

```
KnowledgeService:
  endpoints:
    - GET /knowledge/search
      description: Search knowledge base
      parameters: query, filters, limit
      response: SearchResultsResponse

    - GET /knowledge/articles/{id}
      description: Get specific article
      response: ArticleResponse
```

```
    - POST /knowledge/recommendations
      description: Get contextual recommendations
      request: RecommendationRequest
      response: RecommendationResponse

    - PUT /knowledge/articles/{id}
      description: Update article content
      request: ArticleUpdateRequest

    - POST /knowledge/feedback
      description: Submit article feedback
      request: FeedbackRequest
```

Data Models:

```
KnowledgeArticle:
  id: string (UUID)
  title: string
  content: string
  category: string
  tags: List[string]
  version: integer
  status: enum [draft, published, archived]
  created_at: datetime
  updated_at: datetime
  effectiveness_score: float

SearchResult:
  article_id: string
  title: string
  snippet: string
  relevance_score: float
  category: string

Recommendation:
  article_id: string
  confidence_score: float
  reasoning: string
  context_match: object
```

AI/ML Integration: - **Semantic Search Engine:** Vector embeddings using sentence-transformers - **Content Recommendation:** Collaborative filtering with contextual awareness - **Effectiveness Scoring:** ML model tracking resolution success rates - **Auto-categorization:** BERT-based classification for new content

Performance Specifications: - Search response time: <1 second for 95% of queries - Recommendation accuracy: >85% relevance score - Knowledge base size: Support 1M+ articles - Concurrent search requests: 10,000+ per minute

1.3 Routing Service

Component Overview: Intelligent routing engine that analyzes queries and optimally assigns them to AI agents or human specialists based on complexity, skills, and workload.

API Interface:

```
RoutingService:
  endpoints:
    - POST /routing/analyze
      description: Analyze query for routing decision
      request: RoutingAnalysisRequest
      response: RoutingDecisionResponse

    - GET /routing/agents/available
      description: Get available agents
      parameters: skills, workload_threshold
      response: AvailableAgentsResponse

    - POST /routing/assign
      description: Assign conversation to resource
      request: AssignmentRequest
      response: AssignmentResponse

    - PUT /routing/escalate
      description: Escalate conversation
      request: EscalationRequest

    - GET /routing/queue/status
      description: Get queue status
      response: QueueStatusResponse
```

Data Models:

```
RoutingDecision:
  conversation_id: string
  recommended_resource_type: enum [ai, human, specialist]
  confidence_score: float
  complexity_score: float
  estimated_resolution_time: integer
  reasoning: string

Agent:
  id: string
  name: string
  type: enum [ai, human]
  skills: List[string]
  current_workload: integer
  max_capacity: integer
  performance_metrics: object
  availability_status: enum [available, busy, offline]

QueueItem:
  conversation_id: string
  priority: integer
  wait_time: integer
  estimated_assignment_time: integer
```

Routing Algorithms: 1. **Complexity Assessment:** - NLP analysis of query complexity - Historical resolution pattern matching - Customer tier and urgency factors - Multi-factor scoring algorithm

2. **Skills Matching:**
   - Cosine similarity between query requirements and agent skills
   - Performance history weighting
   - Availability and workload balancing
   - Dynamic skill scoring updates
3. **Load Balancing:**
   - Real-time workload monitoring
   - Predictive capacity planning

- Fair distribution algorithms
- SLA-aware prioritization

Performance Specifications: - Routing decision time: <200ms - Assignment accuracy: >90% optimal resource matching - Queue processing rate: 1,000+ assignments per minute - Load balancing variance: <5% across agents

2. AI/ML Services Component Design

2.1 NLP Service

Component Overview: Advanced natural language processing service providing intent recognition, entity extraction, sentiment analysis, and language detection capabilities.

API Interface:

```
NLPService:
  endpoints:
    - POST /nlp/analyze
      description: Comprehensive text analysis
      request: TextAnalysisRequest
      response: NLPAnalysisResponse

    - POST /nlp/intent
      description: Intent classification
      request: IntentRequest
      response: IntentResponse

    - POST /nlp/entities
      description: Entity extraction
      request: EntityRequest
      response: EntityResponse

    - POST /nlp/sentiment
      description: Sentiment analysis
      request: SentimentRequest
      response: SentimentResponse
```

AI/ML Models:

```
IntentClassifier:
  model_type: BERT-based transformer
  training_data: 100k+ labeled customer service queries
  accuracy_target: 95%
  supported_intents: 50+ e-commerce specific intents

EntityExtractor:
  model_type: Named Entity Recognition (spaCy + custom)
  entities: [product_name, order_id, date, amount, email, phone]
  precision_target: 90%
  recall_target: 85%

SentimentAnalyzer:
  model_type: RoBERTa fine-tuned
  sentiment_classes: [positive, negative, neutral, frustrated, urgent]
  accuracy_target: 90%
  confidence_threshold: 0.8
```

Processing Pipeline: 1. **Text Preprocessing:** - Language detection and normalization - Tokenization and cleaning - Spell checking and correction - Context preservation

2. **Multi-Model Inference:**
   - Parallel processing of intent, entities, sentiment
   - Confidence scoring and validation
   - Result aggregation and consistency checking
   - Context-aware adjustments
3. **Post-Processing:**
   - Result validation and filtering
   - Confidence thresholding
   - Context integration
   - Response formatting

Performance Specifications: - Analysis response time: <2 seconds for 99% of requests - Model accuracy: Intent 95%, Entity 90%, Sentiment 90% - Throughput: 10,000+ analyses per minute - Multi-language support: 25+ languages

2.2 Response Generation Service

Component Overview: AI-powered response generation service creating contextually appropriate, personalized responses using fine-tuned language models with brand voice consistency.

API Interface:

```
ResponseGenerationService:
  endpoints:
    - POST /generation/response
      description: Generate contextual response
      request: ResponseGenerationRequest
      response: GeneratedResponseResponse

    - POST /generation/suggestions
      description: Generate response suggestions
      request: SuggestionRequest
      response: SuggestionResponse

    - POST /generation/personalize
      description: Personalize response content
      request: PersonalizationRequest
      response: PersonalizedResponse
```

AI/ML Architecture:

```
ResponseGenerator:
  base_model: GPT-4 or fine-tuned Llama-2
  fine_tuning_data: Customer service conversations + brand guidelines
  context_window: 8k tokens
  response_length: 50-500 tokens

PersonalizationEngine:
  customer_profile_integration: Yes
  purchase_history_awareness: Yes
  interaction_history_context: Yes
  preference_learning: Continuous

BrandVoiceController:
  tone_consistency: Automated validation
```

```
  style_guidelines: Configurable rules
  compliance_checking: Real-time
  brand_score_threshold: 85%
```

Generation Pipeline: 1. **Context Assembly:** - Conversation history integration - Customer profile enrichment - Knowledge base context - Brand guidelines application

2. **Response Generation:**
   - Multi-candidate generation
   - Quality scoring and ranking
   - Brand voice validation
   - Appropriateness filtering
3. **Post-Processing:**
   - Grammar and style checking
   - Personalization injection
   - Compliance validation
   - Confidence scoring

Performance Specifications: - Generation time: <2 seconds for 99% of responses - Brand consistency score: >85% - Customer satisfaction: >4.0/5.0 for AI responses - Response relevance: >90% accuracy

2.3 Sentiment Analysis Service

Component Overview: Real-time sentiment analysis service providing emotion detection, trend analysis, and escalation triggers for customer interactions.

API Interface:

```
SentimentService:
  endpoints:
    - POST /sentiment/analyze
      description: Analyze text sentiment
      request: SentimentAnalysisRequest
      response: SentimentResponse

    - GET /sentiment/trends
      description: Get sentiment trends
      parameters: timeframe, filters
      response: SentimentTrendsResponse

    - POST /sentiment/monitor
      description: Set up sentiment monitoring
      request: MonitoringRequest
      response: MonitoringResponse
```

ML Models:

```
SentimentClassifier:
  model_type: RoBERTa fine-tuned on customer service data
  emotion_classes: [happy, satisfied, neutral, frustrated, angry, confused]
  confidence_scoring: Probabilistic output
  real_time_processing: <100ms per analysis

TrendAnalyzer:
  time_series_model: LSTM-based
  trend_detection: Statistical change point detection
  anomaly_detection: Isolation Forest
  forecasting_horizon: 24 hours
```

Processing Features: - Real-time emotion detection with confidence scores - Conversation-level sentiment tracking - Escalation trigger automation - Historical trend analysis and reporting - Multi-language sentiment support

3. Integration Services Component Design

3.1 E-commerce Platform Connector

Component Overview: Unified integration service providing real-time connectivity with major e-commerce platforms for order, product, and customer data synchronization.

API Interface:

```
EcommerceConnector:
  endpoints:
    - GET /ecommerce/orders/{order_id}
      description: Retrieve order information
      response: OrderResponse

    - GET /ecommerce/products/{product_id}
      description: Get product details
      response: ProductResponse

    - GET /ecommerce/customers/{customer_id}
      description: Get customer profile
      response: CustomerResponse

    - POST /ecommerce/orders/{order_id}/update
      description: Update order status
      request: OrderUpdateRequest
```

Platform Integrations:

```
SupportedPlatforms:
  - Shopify: REST API + GraphQL + Webhooks
  - WooCommerce: REST API + WP hooks
  - Magento: REST API + GraphQL
  - BigCommerce: REST API + Webhooks
  - Custom: Configurable API adapters

DataSynchronization:
  real_time_updates: Webhook-based
  batch_synchronization: Scheduled jobs
  conflict_resolution: Last-write-wins with versioning
  retry_mechanism: Exponential backoff
```

Performance Specifications: - API response time: <200ms for 95% of requests - Data synchronization latency: <5 minutes - Platform availability: 99.9% uptime - Concurrent connections: 1,000+ per platform

3.2 Communication Channel Adapter

Component Overview: Multi-channel communication service managing message routing and formatting across email, chat, social media, and voice channels.

Channel Support:

```
SupportedChannels:
  Email:
```

```
    protocols: SMTP, IMAP, POP3
    features: Threading, attachments, templates

  WebChat:
    protocols: WebSocket, Server-Sent Events
    features: Real-time, file sharing, typing indicators

  SocialMedia:
    platforms: Facebook, Instagram, Twitter, LinkedIn
    features: DM handling, mention monitoring, response automation

  Voice:
    protocols: SIP, WebRTC
    features: Call routing, transcription, recording

  SMS:
    providers: Twilio, AWS SNS, custom gateways
    features: Two-way messaging, delivery confirmation
```

Message Processing: - Unified message format conversion - Channel-specific formatting and validation - Delivery confirmation and retry logic - Rate limiting and throttling per channel - Message threading and conversation continuity

4. Data Services Component Design

4.1 Customer Data Service

Component Overview: Centralized customer data management service providing unified customer profiles, interaction history, and preference management.

Data Models:

```
CustomerProfile:
  id: string (UUID)
  external_ids: Map[platform, id]
  personal_info: PersonalInfo
  contact_preferences: ContactPreferences
  purchase_history: List[Purchase]
  interaction_history: List[Interaction]
  preferences: CustomerPreferences
  segments: List[string]
  lifetime_value: float
  satisfaction_score: float

PersonalInfo:
  first_name: string
  last_name: string
  email: string
  phone: string
  address: Address
  date_of_birth: date

CustomerPreferences:
  communication_channels: List[string]
  language: string
  timezone: string
  marketing_consent: boolean
  data_processing_consent: boolean
```

Data Processing: - Real-time profile updates and synchronization - Privacy-compliant data handling (GDPR/CCPA) - Customer journey tracking and analytics - Segmentation and targeting capabilities - Data quality validation and enrichment

4.2 Analytics Data Service

Component Overview: High-performance analytics service providing real-time metrics, reporting, and business intelligence capabilities.

Metrics Collection:

```
PerformanceMetrics:
  response_times: Histogram with percentiles
  resolution_rates: Success/failure ratios
  customer_satisfaction: CSAT, NPS scores
  agent_productivity: Tickets per hour, quality scores

BusinessMetrics:
  cost_per_interaction: Calculated costs
  revenue_impact: Conversion tracking
  customer_retention: Churn analysis
  operational_efficiency: Resource utilization
```

Real-time Processing: - Stream processing with Apache Kafka - Time-series data storage in InfluxDB - Real-time dashboard updates - Automated alerting and notifications - Predictive analytics and forecasting

5. Security and Compliance Framework

5.1 Authentication and Authorization Service

Security Architecture:

```
AuthenticationMethods:
  - OAuth 2.0 / OpenID Connect
  - SAML 2.0 for enterprise SSO
  - Multi-factor authentication (MFA)
  - API key authentication for integrations

AuthorizationModel:
  type: Role-Based Access Control (RBAC)
  roles: [admin, manager, agent, readonly, api_user]
  permissions: Granular resource-level permissions
  policy_engine: Attribute-based policies (ABAC)
```

Compliance Features: - GDPR compliance with data subject rights - CCPA compliance for California residents - PCI DSS compliance for payment data - SOC 2 Type II controls implementation - Audit logging and compliance reporting

5.2 Data Protection Service

Encryption Standards:

```
DataAtRest:
  algorithm: AES-256
  key_management: AWS KMS / Azure Key Vault
  database_encryption: Transparent Data Encryption (TDE)

DataInTransit:
  protocol: TLS 1.3
  certificate_management: Automated renewal
```

```
  api_security: JWT tokens with short expiration
```

Privacy Controls: - Automatic PII detection and masking - Data anonymization for analytics - Right to deletion implementation - Consent management and tracking - Data retention policy enforcement

This comprehensive HLD provides the detailed component specifications needed for implementation teams to build a robust, scalable, and secure e-commerce customer service AI platform that meets all functional and non-functional requirements. # Low Level Design (LLD) ## E-commerce Customer Service AI - AI-Powered Intelligent Customer Service and Support Automation Platform

Building upon README, PRD, FRD, NFRD, AD, and HLD foundations for implementation-ready specifications

ETVX Framework

ENTRY CRITERIA

- âœ… README completed with problem overview and technical approach
- âœ… PRD completed with business objectives and success metrics
- âœ… FRD completed with 21 detailed functional requirements
- âœ… NFRD completed with 24 non-functional requirements
- âœ… AD completed with system architecture design
- âœ… HLD completed with component specifications and API designs

TASK

Develop implementation-ready specifications including class structures, database schemas, API implementations, algorithm details, configuration files, and deployment scripts.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] Class designs implement HLD component specifications - [ ] Database schemas support all data models and relationships - [ ] API implementations meet functional requirements - [ ] Algorithms achieve performance targets from NFRD

Validation Criteria: - [ ] Implementation specifications validated with development teams - [ ] Database designs validated with data architects - [ ] API specifications validated with integration teams - [ ] Performance algorithms validated with DevOps teams

EXIT CRITERIA

- âœ… Complete implementation-ready class structures and interfaces
- âœ… Database schemas with indexes and constraints
- âœ… API implementation details with error handling
- âœ… Configuration files and deployment specifications
- âœ… Foundation established for Pseudocode development

---

1. Core Service Implementation

1.1 Conversation Service Classes

```
# conversation_service.py
from dataclasses import dataclass
from typing import List, Optional, Dict, Any
from enum import Enum
import uuid
from datetime import datetime

class ConversationStatus(Enum):
    ACTIVE = "active"
    RESOLVED = "resolved"
    ESCALATED = "escalated"
    CLOSED = "closed"

class MessageType(Enum):
    TEXT = "text"
    IMAGE = "image"
    FILE = "file"
    SYSTEM = "system"

@dataclass
class Message:
    id: str
    conversation_id: str
    sender_type: str
    sender_id: str
    content: str
    message_type: MessageType
    timestamp: datetime
    metadata: Dict[str, Any]

class ConversationService:
    def __init__(self, db_client, nlp_service, cache_client):
        self.db = db_client
        self.nlp = nlp_service
        self.cache = cache_client

    async def create_conversation(self, customer_id: str, channel: str) -> str:
        conversation_id = str(uuid.uuid4())
        conversation = {
            'id': conversation_id,
            'customer_id': customer_id,
            'channel': channel,
            'status': ConversationStatus.ACTIVE.value,
            'created_at': datetime.utcnow(),
            'updated_at': datetime.utcnow()
        }

        await self.db.conversations.insert_one(conversation)
        await self.cache.set(f"conv:{conversation_id}", conversation, ttl=3600)
        return conversation_id

    async def add_message(self, conversation_id: str, message: Message) -> Dict:
        # Process message with NLP
        nlp_result = await self.nlp.analyze_text(message.content)

        # Store message
        message_doc = {
            'id': message.id,
            'conversation_id': conversation_id,
            'sender_type': message.sender_type,
            'content': message.content,
            'timestamp': message.timestamp,
            'intent': nlp_result.get('intent'),
            'sentiment': nlp_result.get('sentiment'),
```

```
            'entities': nlp_result.get('entities')
        }

        await self.db.messages.insert_one(message_doc)

        # Update conversation context
        await self._update_conversation_context(conversation_id, nlp_result)

        return message_doc

    async def get_conversation_context(self, conversation_id: str) -> Dict:
        # Check cache first
        cached = await self.cache.get(f"context:{conversation_id}")
        if cached:
            return cached

        # Build context from database
        conversation = await self.db.conversations.find_one({'id': conversation_id})
        messages = await self.db.messages.find(
            {'conversation_id': conversation_id}
        ).sort('timestamp', 1).to_list(50)

        context = {
            'conversation_id': conversation_id,
            'customer_id': conversation['customer_id'],
            'channel': conversation['channel'],
            'message_count': len(messages),
            'last_intent': messages[-1]['intent'] if messages else None,
            'sentiment_trend': self._calculate_sentiment_trend(messages),
            'entities': self._extract_context_entities(messages)
        }

        await self.cache.set(f"context:{conversation_id}", context, ttl=1800)
        return context
```

1.2 Knowledge Service Implementation

```
# knowledge_service.py
from elasticsearch import AsyncElasticsearch
from sentence_transformers import SentenceTransformer
import numpy as np

class KnowledgeService:
    def __init__(self, es_client: AsyncElasticsearch, vector_model: str):
        self.es = es_client
        self.encoder = SentenceTransformer(vector_model)
        self.index_name = "knowledge_base"

    async def search_articles(self, query: str, filters: Dict = None, limit: int = 10) -> List[Dict]:
        # Generate query embedding
        query_vector = self.encoder.encode([query])[0].tolist()

        # Elasticsearch query with vector similarity and text search
        search_body = {
            "query": {
                "bool": {
                    "should": [
                        {
                            "script_score": {
                                "query": {"match_all": {}},
                                "script": {
                                    "source": "cosineSimilarity(params.query_vector, 'content_vector') + 1.0",
                                    "params": {"query_vector": query_vector}
                                }
                            }
                        },
                        {
                            "multi_match": {
                                "query": query,
                                "fields": ["title^2", "content", "tags"],
                                "type": "best_fields"
                            }
                        }
                    ]
                }
            },
            "size": limit,
            "_source": ["id", "title", "content", "category", "tags", "effectiveness_score"]
        }

        if filters:
            search_body["query"]["bool"]["filter"] = []
            for key, value in filters.items():
                search_body["query"]["bool"]["filter"].append({"term": {key: value}})

        response = await self.es.search(index=self.index_name, body=search_body)

        results = []
        for hit in response['hits']['hits']:
            results.append({
                'article_id': hit['_source']['id'],
                'title': hit['_source']['title'],
                'content': hit['_source']['content'][:500] + "...",
                'relevance_score': hit['_score'],
                'category': hit['_source']['category'],
                'effectiveness_score': hit['_source']['effectiveness_score']
            })

        return results

    async def get_recommendations(self, conversation_context: Dict) -> List[Dict]:
        # Extract key information from context
        intent = conversation_context.get('last_intent')
        entities = conversation_context.get('entities', [])

        # Build recommendation query
        query_parts = []
        if intent:
            query_parts.append(intent)

        for entity in entities:
            if entity.get('label') in ['PRODUCT', 'ORDER', 'ISSUE']:
                query_parts.append(entity.get('text'))

        query = " ".join(query_parts)

        # Get recommendations with higher relevance threshold
        recommendations = await self.search_articles(query, limit=5)
```

```
        # Filter by relevance score
        filtered_recommendations = [
            rec for rec in recommendations
            if rec['relevance_score'] > 0.7
        ]

        return filtered_recommendations[:3]
```

2. Database Schema Implementation

2.1 PostgreSQL Schema

```
-- conversations table
CREATE TABLE conversations (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    customer_id VARCHAR(255) NOT NULL,
    channel VARCHAR(50) NOT NULL,
    status VARCHAR(20) NOT NULL DEFAULT 'active',
    priority VARCHAR(10) NOT NULL DEFAULT 'medium',
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    metadata JSONB,

    INDEX idx_conversations_customer (customer_id),
    INDEX idx_conversations_status (status),
    INDEX idx_conversations_created (created_at)
);

-- messages table
CREATE TABLE messages (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    conversation_id UUID NOT NULL REFERENCES conversations(id),
    sender_type VARCHAR(20) NOT NULL,
    sender_id VARCHAR(255) NOT NULL,
    content TEXT NOT NULL,
    message_type VARCHAR(20) NOT NULL DEFAULT 'text',
    timestamp TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    intent VARCHAR(100),
    sentiment_score DECIMAL(3,2),
    entities JSONB,
    metadata JSONB,

    INDEX idx_messages_conversation (conversation_id),
    INDEX idx_messages_timestamp (timestamp),
    INDEX idx_messages_intent (intent)
);

-- customers table
CREATE TABLE customers (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    external_id VARCHAR(255) UNIQUE,
    email VARCHAR(255) UNIQUE,
    phone VARCHAR(50),
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    preferences JSONB,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

    INDEX idx_customers_email (email),
    INDEX idx_customers_external_id (external_id)
);

-- agents table
CREATE TABLE agents (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    type VARCHAR(20) NOT NULL, -- 'ai' or 'human'
    skills TEXT[],
    current_workload INTEGER DEFAULT 0,
    max_capacity INTEGER DEFAULT 10,
    status VARCHAR(20) DEFAULT 'available',
    performance_metrics JSONB,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

    INDEX idx_agents_type (type),
    INDEX idx_agents_status (status),
    INDEX idx_agents_skills USING GIN (skills)
);
```

2.2 MongoDB Schema

```
// knowledge_articles collection
db.knowledge_articles.createIndex({"category": 1, "status": 1})
db.knowledge_articles.createIndex({"tags": 1})
db.knowledge_articles.createIndex({"effectiveness_score": -1})
db.knowledge_articles.createIndex({"updated_at": -1})

// conversation_contexts collection
db.conversation_contexts.createIndex({"conversation_id": 1}, {"unique": true})
db.conversation_contexts.createIndex({"customer_id": 1})
db.conversation_contexts.createIndex({"updated_at": -1})

// Example document structure
{
  "_id": ObjectId("..."),
  "conversation_id": "uuid-string",
  "customer_id": "uuid-string",
  "context_variables": {
    "current_intent": "order_status",
    "entities": [
      {"type": "ORDER_ID", "value": "12345", "confidence": 0.95}
    ],
    "sentiment_history": [0.2, 0.1, -0.3, -0.5],
    "interaction_count": 4
  },
  "created_at": ISODate("..."),
  "updated_at": ISODate("...")
}
```

3. API Implementation Details

3.1 FastAPI Application Structure

```
# main.py
from fastapi import FastAPI, HTTPException, Depends
```

```python
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from typing import List, Optional
import asyncio

app = FastAPI(title="E-commerce Customer Service AI", version="1.0.0")

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Request/Response Models
class ConversationCreateRequest(BaseModel):
    customer_id: str
    channel: str
    initial_message: Optional[str] = None

class MessageRequest(BaseModel):
    content: str
    sender_type: str
    sender_id: str
    message_type: str = "text"

class ConversationResponse(BaseModel):
    id: str
    customer_id: str
    channel: str
    status: str
    created_at: str

# API Endpoints
@app.post("/conversations", response_model=ConversationResponse)
async def create_conversation(
    request: ConversationCreateRequest,
    conversation_service: ConversationService = Depends(get_conversation_service)
):
    try:
        conversation_id = await conversation_service.create_conversation(
            request.customer_id,
            request.channel
        )

        conversation = await conversation_service.get_conversation(conversation_id)
        return ConversationResponse(**conversation)

    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@app.post("/conversations/{conversation_id}/messages")
async def add_message(
    conversation_id: str,
    request: MessageRequest,
    conversation_service: ConversationService = Depends(get_conversation_service)
):
    try:
        message = Message(
            id=str(uuid.uuid4()),
            conversation_id=conversation_id,
            sender_type=request.sender_type,
            sender_id=request.sender_id,
            content=request.content,
            message_type=MessageType(request.message_type),
            timestamp=datetime.utcnow(),
            metadata={}
        )

        result = await conversation_service.add_message(conversation_id, message)

        # Trigger response generation if from customer
        if request.sender_type == "customer":
            asyncio.create_task(
                generate_and_send_response(conversation_id, message)
            )

        return {"message_id": result["id"], "status": "processed"}

    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

4. Configuration Management

4.1 Environment Configuration

```yaml
# config/production.yaml
database:
  postgresql:
    host: ${DB_HOST}
    port: ${DB_PORT:5432}
    database: ${DB_NAME}
    username: ${DB_USER}
    password: ${DB_PASSWORD}
    pool_size: 20
    max_overflow: 30

  mongodb:
    uri: ${MONGO_URI}
    database: ${MONGO_DB_NAME}

  redis:
    host: ${REDIS_HOST}
    port: ${REDIS_PORT:6379}
    password: ${REDIS_PASSWORD}
    db: 0

ai_services:
  nlp:
    model_name: "bert-base-uncased"
    max_sequence_length: 512
    batch_size: 32

  response_generation:
    model_name: "gpt-4"
    api_key: ${OPENAI_API_KEY}
    max_tokens: 500
```

```
    temperature: 0.7

elasticsearch:
  hosts:
    - ${ES_HOST}:${ES_PORT:9200}
  username: ${ES_USERNAME}
  password: ${ES_PASSWORD}

performance:
  max_concurrent_conversations: 50000
  message_processing_timeout: 30
  response_generation_timeout: 10
  cache_ttl: 3600

security:
  jwt_secret: ${JWT_SECRET}
  encryption_key: ${ENCRYPTION_KEY}
  allowed_origins: ${ALLOWED_ORIGINS}
```

4.2 Docker Configuration

```
# Dockerfile
FROM python:3.11-slim

WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    gcc \
    g++ \
    && rm -rf /var/lib/apt/lists/*

# Install Python dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Create non-root user
RUN useradd -m -u 1000 appuser && chown -R appuser:appuser /app
USER appuser

# Health check
HEALTHCHECK --interval=30s --timeout=30s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:8000/health || exit 1

EXPOSE 8000

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000", "--workers", "4"]
```

```
# docker-compose.yml
version: '3.8'

services:
  api:
    build: .
    ports:
      - "8000:8000"
    environment:
      - DB_HOST=postgres
      - REDIS_HOST=redis
      - ES_HOST=elasticsearch
    depends_on:
      - postgres
      - redis
      - elasticsearch

  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: customer_service
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine
    command: redis-server --requirepass ${REDIS_PASSWORD}

  elasticsearch:
    image: elasticsearch:8.8.0
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=false
    volumes:
      - es_data:/usr/share/elasticsearch/data

volumes:
  postgres_data:
  es_data:
```

This comprehensive LLD provides implementation-ready specifications that development teams can use to build the e-commerce customer service AI platform with all required functionality, performance characteristics, and security features. # Pseudocode ## E-commerce Customer Service AI - AI-Powered Intelligent Customer Service and Support Automation Platform

Building upon README, PRD, FRD, NFRD, AD, HLD, and LLD foundations for executable implementation algorithms

ETVX Framework

ENTRY CRITERIA

- âœ… README completed with problem overview and technical approach
- âœ… PRD completed with business objectives, market analysis, and success metrics
- âœ… FRD completed with 21 detailed functional requirements across 7 modules
- âœ… NFRD completed with 24 non-functional requirements covering performance, security, and scalability
- âœ… AD completed with microservices architecture and cloud-native deployment strategy
- âœ… HLD completed with component specifications and API designs
- âœ… LLD completed with implementation-ready class structures and database schemas

TASK

Develop executable pseudocode algorithms for all core system components including conversation management, AI-powered processing, intelligent routing, analytics, and performance optimization.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] Pseudocode algorithms align with LLD class implementations - [ ] Processing workflows meet performance requirements (<2s response time) - [ ] AI/ML algorithms implement NLP and response generation features - [ ] Integration algorithms support all e-commerce platform connectors

Validation Criteria: - [ ] Pseudocode validated with customer service domain experts - [ ] Algorithms validated with AI/ML and integration development teams - [ ] Performance algorithms validated with scalability and optimization teams - [ ] Security algorithms validated with information security teams

EXIT CRITERIA

- âœ… Complete executable pseudocode for all system components
- âœ… AI/ML processing algorithms for conversation management and response generation
- âœ… Integration workflows for e-commerce platforms and communication channels
- âœ… Analytics and performance monitoring algorithms
- âœ… Implementation-ready foundation for development teams

Reference to Previous Documents

This Pseudocode builds upon **README**, **PRD**, **FRD**, **NFRD**, **AD**, **HLD**, and **LLD** foundations: - **LLD Class Structures** â† Executable algorithms with method implementations - **HLD Processing Workflows** â† Step-by-step algorithmic procedures - **NFRD Performance Requirements** â† Optimization algorithms for <2s response times - **AD Security Framework** â† Authentication and data protection algorithms

1. Conversation Management Algorithms

1.1 Multi-Channel Message Processing

```
ALGORITHM: ProcessIncomingMessage
INPUT: message (object), channel (string), customer_id (string)
OUTPUT: ProcessedMessage (object)

BEGIN ProcessIncomingMessage
    start_time = getCurrentTime()

    // Validate and sanitize input
    IF NOT ValidateMessage(message) THEN
        THROW ValidationException("Invalid message format")
    END IF

    sanitized_content = SanitizeContent(message.content)

    // Create conversation if not exists
    conversation_id = GetOrCreateConversation(customer_id, channel)

    // Generate unique message ID
    message_id = GenerateUUID()

    // Process message with NLP pipeline
    nlp_result = ProcessWithNLP(sanitized_content)

    // Create message object
    processed_message = Message{
        id: message_id,
        conversation_id: conversation_id,
        sender_type: "customer",
        sender_id: customer_id,
        content: sanitized_content,
        message_type: DetermineMessageType(message),
        timestamp: getCurrentTime(),
        intent: nlp_result.intent,
        sentiment: nlp_result.sentiment,
        entities: nlp_result.entities,
        confidence_scores: nlp_result.confidence
    }

    // Store message in database
    TRY
        SaveMessageToDatabase(processed_message)

        // Update conversation context
        UpdateConversationContext(conversation_id, nlp_result)

        // Cache recent messages for quick access
        CacheRecentMessages(conversation_id, processed_message)

        // Trigger response generation asynchronously
        TriggerResponseGeneration(conversation_id, processed_message)

        // Log processing metrics
        processing_time = getCurrentTime() - start_time
        LogPerformanceMetric("message_processing", processing_time)

        RETURN processed_message

    CATCH DatabaseException e
        LogError("Message processing failed", e)
        THROW MessageProcessingException("Failed to process message: " + e.message)
    END TRY
END ProcessIncomingMessage

ALGORITHM: ProcessWithNLP
INPUT: content (string)
OUTPUT: NLPResult (object)

BEGIN ProcessWithNLP
    // Parallel processing of NLP tasks
    PARALLEL BEGIN
        intent_result = ClassifyIntent(content)
        entity_result = ExtractEntities(content)
        sentiment_result = AnalyzeSentiment(content)
        language_result = DetectLanguage(content)
    PARALLEL END

    // Validate results and apply confidence thresholds
    validated_intent = ValidateIntentResult(intent_result, threshold=0.8)
    validated_entities = FilterEntitiesByConfidence(entity_result, threshold=0.7)
    validated_sentiment = ValidateSentimentResult(sentiment_result, threshold=0.75)

    RETURN NLPResult{
        intent: validated_intent,
        entities: validated_entities,
        sentiment: validated_sentiment,
        language: language_result,
        confidence: CalculateOverallConfidence(intent_result, entity_result, sentiment_result)
    }
```

```
END ProcessWithNLP
```

1.2 Conversation Context Management

```
ALGORITHM: UpdateConversationContext
INPUT: conversation_id (string), nlp_result (NLPResult)
OUTPUT: ConversationContext (object)

BEGIN UpdateConversationContext
    // Retrieve existing context
    existing_context = GetConversationContext(conversation_id)

    IF existing_context IS NULL THEN
        context = InitializeNewContext(conversation_id)
    ELSE
        context = existing_context
    END IF

    // Update context with new information
    context.message_count += 1
    context.last_intent = nlp_result.intent
    context.last_update = getCurrentTime()

    // Update sentiment trend (rolling window of last 10 messages)
    context.sentiment_history.append(nlp_result.sentiment.score)
    IF Length(context.sentiment_history) > 10 THEN
        context.sentiment_history = context.sentiment_history[-10:]
    END IF

    // Update entity tracking
    FOR each entity IN nlp_result.entities DO
        IF entity.type IN ["ORDER_ID", "PRODUCT_NAME", "ISSUE_TYPE"] THEN
            context.tracked_entities[entity.type] = entity.value
        END IF
    END FOR

    // Calculate context metrics
    context.sentiment_trend = CalculateSentimentTrend(context.sentiment_history)
    context.complexity_score = CalculateComplexityScore(context)
    context.escalation_risk = CalculateEscalationRisk(context)

    // Check for escalation triggers
    IF ShouldTriggerEscalation(context) THEN
        TriggerEscalation(conversation_id, context.escalation_risk)
    END IF

    // Save updated context
    SaveConversationContext(context)
    CacheConversationContext(conversation_id, context, ttl=1800)

    RETURN context
END UpdateConversationContext

ALGORITHM: CalculateEscalationRisk
INPUT: context (ConversationContext)
OUTPUT: risk_score (float)

BEGIN CalculateEscalationRisk
    risk_factors = []

    // Sentiment deterioration
    IF context.sentiment_trend < -0.3 THEN
        risk_factors.append(0.4)
    END IF

    // Message count without resolution
    IF context.message_count > 5 AND context.last_intent != "satisfied" THEN
        risk_factors.append(0.3)
    END IF

    // Complexity indicators
    IF context.complexity_score > 0.7 THEN
        risk_factors.append(0.2)
    END IF

    // Explicit escalation requests
    IF "escalate" IN context.last_message_content OR "manager" IN context.last_message_content THEN
        risk_factors.append(0.8)
    END IF

    // Calculate weighted risk score
    risk_score = Min(Sum(risk_factors), 1.0)

    RETURN risk_score
END CalculateEscalationRisk
```

2. AI-Powered Response Generation Algorithms

2.1 Intelligent Response Generation

```
ALGORITHM: GenerateResponse
INPUT: conversation_id (string), customer_message (Message)
OUTPUT: GeneratedResponse (object)

BEGIN GenerateResponse
    start_time = getCurrentTime()

    // Gather context and information
    context = GetConversationContext(conversation_id)
    customer_profile = GetCustomerProfile(context.customer_id)
    knowledge_recommendations = GetKnowledgeRecommendations(context, customer_message)

    // Determine response strategy
    response_strategy = DetermineResponseStrategy(
        intent: customer_message.intent,
        complexity: context.complexity_score,
        sentiment: customer_message.sentiment,
        customer_tier: customer_profile.tier
    )

    // Generate response based on strategy
    IF response_strategy == "DIRECT_ANSWER" THEN
        response = GenerateDirectAnswer(customer_message, knowledge_recommendations)
    ELSE IF response_strategy == "GUIDED_RESOLUTION" THEN
        response = GenerateGuidedResolution(context, knowledge_recommendations)
    ELSE IF response_strategy == "EMPATHETIC_SUPPORT" THEN
        response = GenerateEmpathetic Response(customer_message, context)
```

```
    ELSE IF response_strategy == "ESCALATION_PREP" THEN
        response = GenerateEscalationResponse(context)
    END IF

    // Enhance response with personalization
    personalized_response = PersonalizeResponse(
        response: response,
        customer_profile: customer_profile,
        interaction_history: context.interaction_history
    )

    // Validate response quality
    quality_score = ValidateResponseQuality(personalized_response, customer_message)

    IF quality_score < 0.7 THEN
        // Regenerate with different approach
        fallback_response = GenerateFallbackResponse(customer_message, context)
        personalized_response = fallback_response
        quality_score = 0.6  // Fallback quality score
    END IF

    // Prepare final response object
    generated_response = GeneratedResponse{
        content: personalized_response.text,
        response_type: response_strategy,
        confidence_score: quality_score,
        suggested_actions: personalized_response.actions,
        knowledge_sources: ExtractSourceReferences(knowledge_recommendations),
        generation_time: getCurrentTime() - start_time,
        requires_human_review: quality_score < 0.8
    }

    // Log response generation metrics
    LogResponseMetrics(generated_response, customer_message.intent)

    RETURN generated_response
END GenerateResponse

ALGORITHM: PersonalizeResponse
INPUT: response (Response), customer_profile (CustomerProfile), interaction_history (List)
OUTPUT: PersonalizedResponse (object)

BEGIN PersonalizeResponse
    personalized_text = response.text

    // Add customer name if available and appropriate
    IF customer_profile.first_name IS NOT NULL AND response.tone == "friendly" THEN
        personalized_text = "Hi " + customer_profile.first_name + ", " + personalized_text
    END IF

    // Reference previous interactions if relevant
    IF Length(interaction_history) > 0 THEN
        last_interaction = interaction_history[-1]
        IF last_interaction.resolution_status == "pending" THEN
            personalized_text = "Following up on your previous inquiry, " + personalized_text
        END IF
    END IF

    // Adjust language based on customer preferences
    IF customer_profile.communication_style == "formal" THEN
        personalized_text = FormalizeLanguage(personalized_text)
    ELSE IF customer_profile.communication_style == "casual" THEN
        personalized_text = CasualizeLanguage(personalized_text)
    END IF

    // Add relevant product recommendations if appropriate
    suggested_actions = response.actions
    IF response.intent == "product_inquiry" AND customer_profile.purchase_history IS NOT EMPTY THEN
        related_products = GetRelatedProducts(customer_profile.purchase_history)
        suggested_actions.extend(CreateProductRecommendations(related_products))
    END IF

    RETURN PersonalizedResponse{
        text: personalized_text,
        actions: suggested_actions,
        personalization_applied: TRUE,
        personalization_factors: ["name", "history", "style", "preferences"]
    }
END PersonalizeResponse
```

3. Intelligent Routing Algorithms

3.1 Dynamic Task Routing

```
ALGORITHM: RouteConversation
INPUT: conversation_id (string), routing_request (RoutingRequest)
OUTPUT: RoutingDecision (object)

BEGIN RouteConversation
    // Analyze conversation for routing decision
    context = GetConversationContext(conversation_id)
    complexity_analysis = AnalyzeComplexity(context, routing_request)

    // Determine if AI can handle the request
    ai_capability_score = AssessAICapability(
        intent: routing_request.intent,
        complexity: complexity_analysis.score,
        entities: routing_request.entities,
        sentiment: routing_request.sentiment
    )

    IF ai_capability_score >= 0.8 THEN
        // Route to AI with high confidence
        decision = RoutingDecision{
            resource_type: "AI",
            resource_id: "primary_ai_agent",
            confidence: ai_capability_score,
            estimated_resolution_time: EstimateAIResolutionTime(complexity_analysis),
            reasoning: "High confidence AI resolution"
        }
    ELSE IF ai_capability_score >= 0.6 THEN
        // Route to AI with human backup
        decision = RoutingDecision{
            resource_type: "AI_WITH_BACKUP",
            resource_id: "primary_ai_agent",
            backup_resource_id: FindBestHumanAgent(routing_request),
            confidence: ai_capability_score,
            estimated_resolution_time: EstimateAIResolutionTime(complexity_analysis),
```

```
            reasoning: "Moderate confidence AI with human backup"
        }
    ELSE
        // Route directly to human agent
        best_agent = FindBestHumanAgent(routing_request)
        decision = RoutingDecision{
            resource_type: "HUMAN",
            resource_id: best_agent.id,
            confidence: 0.9,
            estimated_resolution_time: EstimateHumanResolutionTime(complexity_analysis, best_agent),
            reasoning: "Complex query requiring human expertise"
        }
    END IF

    // Apply load balancing if multiple options available
    decision = ApplyLoadBalancing(decision, GetCurrentWorkloads())

    // Execute routing decision
    ExecuteRouting(conversation_id, decision)

    // Log routing decision for analysis
    LogRoutingDecision(conversation_id, decision, complexity_analysis)

    RETURN decision
END RouteConversation

ALGORITHM: FindBestHumanAgent
INPUT: routing_request (RoutingRequest)
OUTPUT: Agent (object)

BEGIN FindBestHumanAgent
    // Get available human agents
    available_agents = GetAvailableAgents(type="human")

    IF Length(available_agents) == 0 THEN
        THROW NoAgentsAvailableException("No human agents currently available")
    END IF

    best_agent = NULL
    best_score = 0

    FOR each agent IN available_agents DO
        // Calculate skill match score
        skill_score = CalculateSkillMatch(agent.skills, routing_request.required_skills)

        // Calculate workload factor (prefer less loaded agents)
        workload_factor = 1.0 - (agent.current_workload / agent.max_capacity)

        // Calculate performance factor
        performance_factor = agent.performance_metrics.average_resolution_rate

        // Calculate customer tier match (VIP customers to senior agents)
        tier_match = CalculateTierMatch(agent.seniority, routing_request.customer_tier)

        // Combined score with weights
        combined_score = (skill_score * 0.4) + (workload_factor * 0.3) +
                         (performance_factor * 0.2) + (tier_match * 0.1)

        IF combined_score > best_score THEN
            best_score = combined_score
            best_agent = agent
        END IF
    END FOR

    RETURN best_agent
END FindBestHumanAgent
```

4. Knowledge Management Algorithms

4.1 Semantic Knowledge Search

```
ALGORITHM: SearchKnowledgeBase
INPUT: query (string), context (ConversationContext), filters (object)
OUTPUT: List<KnowledgeResult>

BEGIN SearchKnowledgeBase
    // Preprocess query
    processed_query = PreprocessQuery(query)

    // Generate query embeddings
    query_embedding = GenerateEmbedding(processed_query)

    // Extract context keywords
    context_keywords = ExtractContextKeywords(context)

    // Perform hybrid search (vector + text + context)
    PARALLEL BEGIN
        vector_results = VectorSearch(query_embedding, limit=20)
        text_results = TextSearch(processed_query, limit=20)
        context_results = ContextualSearch(context_keywords, limit=10)
    PARALLEL END

    // Merge and rank results
    merged_results = MergeSearchResults(vector_results, text_results, context_results)

    // Apply filters
    IF filters IS NOT NULL THEN
        filtered_results = ApplyFilters(merged_results, filters)
    ELSE
        filtered_results = merged_results
    END IF

    // Re-rank based on effectiveness and recency
    ranked_results = RerankResults(
        results: filtered_results,
        effectiveness_weight: 0.4,
        relevance_weight: 0.4,
        recency_weight: 0.2
    )

    // Enhance results with snippets and metadata
    enhanced_results = []
    FOR each result IN ranked_results[0:10] DO  // Top 10 results
        enhanced_result = KnowledgeResult{
            article_id: result.id,
            title: result.title,
            snippet: GenerateSnippet(result.content, processed_query),
            relevance_score: result.score,
```

```
            effectiveness_score: result.effectiveness,
            category: result.category,
            last_updated: result.updated_at,
            usage_count: result.usage_statistics.total_views
        }
        enhanced_results.append(enhanced_result)
    END FOR

    // Log search for analytics
    LogKnowledgeSearch(query, context, Length(enhanced_results))

    RETURN enhanced_results
END SearchKnowledgeBase

ALGORITHM: GenerateContextualRecommendations
INPUT: conversation_context (ConversationContext), current_message (Message)
OUTPUT: List<Recommendation>

BEGIN GenerateContextualRecommendations
    recommendations = []

    // Intent-based recommendations
    IF current_message.intent IS NOT NULL THEN
        intent_articles = GetArticlesByIntent(current_message.intent)
        FOR each article IN intent_articles[0:3] DO
            recommendations.append(Recommendation{
                article_id: article.id,
                confidence: 0.8,
                reasoning: "Matches detected intent: " + current_message.intent,
                recommendation_type: "INTENT_MATCH"
            })
        END FOR
    END IF

    // Entity-based recommendations
    FOR each entity IN current_message.entities DO
        IF entity.type IN ["PRODUCT", "ORDER", "ISSUE"] THEN
            entity_articles = GetArticlesByEntity(entity.type, entity.value)
            FOR each article IN entity_articles[0:2] DO
                recommendations.append(Recommendation{
                    article_id: article.id,
                    confidence: 0.7,
                    reasoning: "Related to " + entity.type + ": " + entity.value,
                    recommendation_type: "ENTITY_MATCH"
                })
            END FOR
        END IF
    END FOR

    // Conversation history recommendations
    IF Length(conversation_context.interaction_history) > 1 THEN
        similar_conversations = FindSimilarConversations(conversation_context)
        FOR each conv IN similar_conversations[0:2] DO
            successful_articles = GetSuccessfulArticles(conv.id)
            FOR each article IN successful_articles DO
                recommendations.append(Recommendation{
                    article_id: article.id,
                    confidence: 0.6,
                    reasoning: "Successful in similar conversations",
                    recommendation_type: "HISTORICAL_SUCCESS"
                })
            END FOR
        END FOR
    END IF

    // Remove duplicates and sort by confidence
    unique_recommendations = RemoveDuplicates(recommendations)
    sorted_recommendations = SortByConfidence(unique_recommendations)

    RETURN sorted_recommendations[0:5]  // Top 5 recommendations
END GenerateContextualRecommendations
```

5. Performance Monitoring Algorithms

5.1 Real-Time Analytics Processing

```
ALGORITHM: ProcessRealTimeMetrics
INPUT: event_stream (Stream<Event>)
OUTPUT: MetricsUpdate (object)

BEGIN ProcessRealTimeMetrics
    WHILE event_stream.hasNext() DO
        event = event_stream.next()

        // Process different event types
        SWITCH event.type
            CASE "message_processed":
                UpdateMessageMetrics(event)
            CASE "response_generated":
                UpdateResponseMetrics(event)
            CASE "conversation_resolved":
                UpdateResolutionMetrics(event)
            CASE "escalation_triggered":
                UpdateEscalationMetrics(event)
            CASE "customer_satisfaction":
                UpdateSatisfactionMetrics(event)
        END SWITCH

        // Check for anomalies
        anomaly_detected = DetectAnomalies(event)
        IF anomaly_detected THEN
            TriggerAlert(anomaly_detected)
        END IF

        // Update real-time dashboards
        IF event.timestamp % 30 == 0 THEN  // Every 30 seconds
            UpdateDashboards(GetCurrentMetrics())
        END IF
    END WHILE
END ProcessRealTimeMetrics

ALGORITHM: CalculatePerformanceScore
INPUT: conversation_id (string), resolution_data (ResolutionData)
OUTPUT: performance_score (float)

BEGIN CalculatePerformanceScore
    metrics = GetConversationMetrics(conversation_id)
```

```
    // Response time score (0-1, higher is better)
    avg_response_time = metrics.total_response_time / metrics.message_count
    response_score = Max(0, 1 - (avg_response_time / 120))  // 2 minutes target

    // Resolution efficiency score
    resolution_score = IF metrics.resolved_by_ai THEN 1.0 ELSE 0.7

    // Customer satisfaction score
    satisfaction_score = resolution_data.customer_rating / 5.0

    // First contact resolution bonus
    fcr_bonus = IF metrics.message_count <= 2 AND metrics.resolved THEN 0.2 ELSE 0.0

    // Calculate weighted performance score
    performance_score = (response_score * 0.3) +
                        (resolution_score * 0.3) +
                        (satisfaction_score * 0.3) +
                        fcr_bonus

    // Ensure score is between 0 and 1
    performance_score = Min(Max(performance_score, 0.0), 1.0)

    RETURN performance_score
END CalculatePerformanceScore
```

This comprehensive pseudocode provides executable algorithms for all core system components, enabling direct implementation of the e-commerce customer service AI platform while maintaining alignment with all previous requirements and architectural decisions.