Comprehensive Study Document

Multi-Modal Financial Advisor Chatbot

Table of Contents

1. Project Overview

1.1 Primary Goal and Purpose

The primary goal of the ewfx/aidhp-deep-agents project is to solve the challenge of hyper-personalized financial product recommendations by leveraging Generative AI and multi-modal data processing. The project aims to create an AI-driven digital financial advisor that dynamically adapts to users' evolving needs, moving beyond traditional, generic financial advice.

The core purpose of this project is to revolutionize digital banking by providing highly personalized, AI-driven financial planning that enhances user engagement, financial literacy, and decision-making through a smart, adaptive chatbot.

1.2 Key Use Cases and User Personas

The Multi-Modal Financial Advisor Chatbot is designed to serve various user personas with different financial needs, backgrounds, and goals. Some of the key personas include:

Persona	Description	Primary Goals	Use Cases
Young Professional	25-35 years old, early in career, moderate income, some debt (student loans)	Building credit, managing debt, starting investments	- Basic investment recommendations - Debt management strategies - Budgeting assistance
Family Planner	35-45 years old, established career, family responsibilities	College savings, retirement planning, insurance needs	- Education savings plans - Life insurance options - Long-term investment strategies

Pre-Retiree	50-65 years old, peak earning years, focused on retirement	Maximizing retirement savings, portfolio adjustments	- Retirement readiness assessment - Conservative investment shifts - Tax optimization strategies
Small Business Owner	Variable age, entrepreneurial, complex financial needs	Business growth, personal-business financial separation	- Business loan options - Retirement plans for small businesses - Tax planning for entrepreneurs

1.3 Business Problems and Expected Outcomes

Business Problems Addressed:

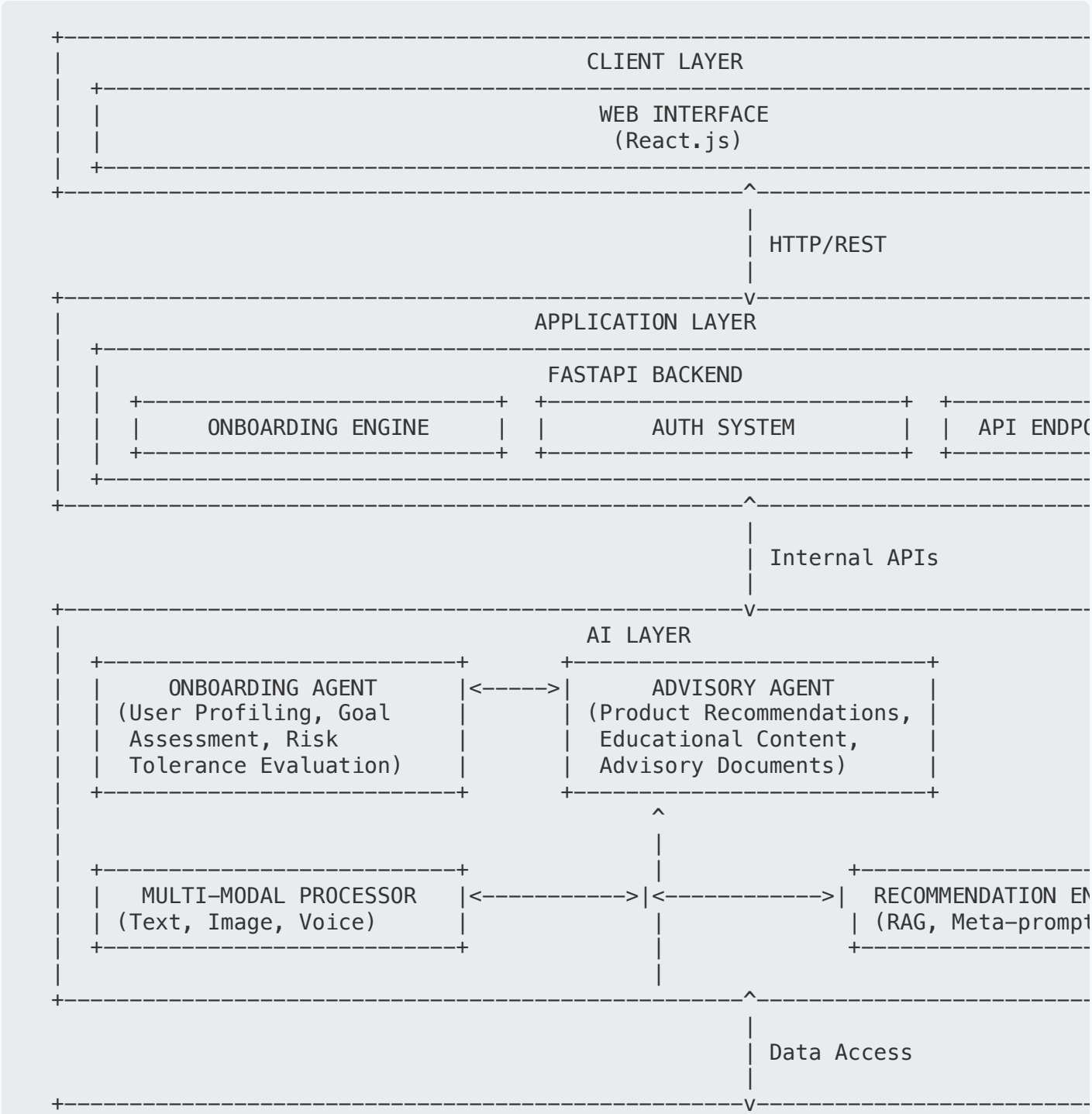
- Generic, one-size-fits-all financial advice that fails to address individual needs
- Low customer engagement with traditional financial advisory services
- Limited accessibility to personalized financial guidance for average consumers
- Inefficient customer onboarding processes in financial institutions
- Poor customer retention due to lack of personalization and follow-up
- Inconsistent quality of financial advice provided by human advisors

Expected Outcomes:

- Increased customer engagement with financial products through personalized recommendations
- Enhanced financial literacy among users through tailored educational content
- Higher conversion rates for financial product offerings
- Improved customer satisfaction and loyalty
- Reduction in customer service costs through automated, intelligent advisory services
- More inclusive financial advisory services reaching underserved demographics
- Data-driven insights into customer preferences and behaviors for financial institutions

2. High-Level Architecture

2.1 Architectural Diagram



```
|                                    DATA LAYER
|  +--------------------------+  +--------------------------+  +--------------
|  |        MONGODB           |  |       VECTOR STORE       |  |       REDIS C
|  | (User Profiles,          |  | (Embeddings for RAG)     |  | (Performance
|  |  Conversations,          |  |                          |  |
|  |  Financial Data)         |  |                          |  |
|  +--------------------------+  +--------------------------+  +--------------
+-------------------------------------------------------------------------------

+-------------------------------------------------------------------------------
|                                  ETHICAL AI LAYER
|  +--------------------------+  +--------------------------+  +--------------
|  |   EXPLAINABILITY TOOLS   |  |    BIAS DETECTION        |  |  TRANSPARENCY
|  +--------------------------+  +--------------------------+  +--------------
+-------------------------------------------------------------------------------
```

2.2 System Architecture Overview

The Multi-Modal Financial Advisor Chatbot employs a layered microservices architecture organized around business capabilities and separated by layers that handle specific aspects of the application. This architecture was chosen to ensure:

- **Modularity:** Each component can be developed, tested, and deployed independently.
- **Scalability:** Different services can be scaled according to demand.
- **Flexibility:** Multiple LLM providers can be integrated, ensuring resilience if one provider is unavailable.
- **Maintainability:** Components can be updated or replaced with minimal impact on the overall system.

The main architectural layers include:

2.2.1 Client Layer

The client layer consists of a React.js-based web interface that provides users with an interactive platform for chat interactions, document uploads, and viewing personalized financial content. This layer handles the presentation logic and user experience aspects of the application.

2.2.2 Application Layer

The application layer is built using FastAPI (Python) and serves as the main orchestration layer, handling API requests, authentication, and routing to appropriate services. The FastAPI backend integrates with various AI services and manages the overall flow of the application.

2.2.3 AI Layer

This layer contains the core intelligence of the system, including:

- **Dual-Agent System:** Two specialized LLM instances - an Onboarding Agent for user profiling and an Advisory Agent for generating personalized financial recommendations.
- **Multi-Modal Processor:** Handles diverse input types (text, image, and voice) to extract and interpret user information.
- **Recommendation Engine:** Uses Retrieval-Augmented Generation (RAG) and meta-prompts to generate personalized financial advice.

2.2.4 Data Layer

The data layer manages all persistent data storage needs of the application:

- **MongoDB:** Primary database for storing user profiles, conversations, and financial data.
- **Vector Store:** Specialized storage for embeddings used in the RAG system.
- **Redis Cache:** In-memory data structure store used for caching to enhance performance.

2.2.5 Ethical AI Layer

This cross-cutting layer implements tools and mechanisms to ensure the AI system operates ethically:

- **Explainability Tools:** Provide reasoning and justification for AI recommendations.
- **Bias Detection:** Monitors and mitigates biases in AI-generated financial advice.
- **Transparency Mechanisms:** Ensure users understand how their data influences recommendations.

2.3 Data Flow Between Components

The data flow within the Multi-Modal Financial Advisor Chatbot follows several key pathways:

2.3.1 Onboarding Flow

1. User inputs (text, uploaded documents) are received through the Web Interface and sent to the FastAPI Backend.
2. The Onboarding Engine routes the request to the Onboarding Agent in the AI Layer.
3. The Onboarding Agent processes the input, potentially utilizing the Multi-Modal Processor for non-text inputs.
4. The agent generates contextually appropriate questions to gather more user information.
5. User responses are used to build a comprehensive profile stored in MongoDB.
6. Once sufficient information is gathered, the system transitions to the advisory phase.

2.3.2 Advisory Flow

1. The Advisory Agent retrieves the user profile from MongoDB.
2. The Recommendation Engine generates personalized meta-prompts based on the user profile.
3. The Advisory Agent uses these meta-prompts to query appropriate LLMs for financial recommendations.
4. The RAG system enhances responses by retrieving relevant financial information from the Vector Store.
5. Generated recommendations are processed through the Ethical AI Layer to ensure they are unbiased and explainable.
6. The finalized recommendations and educational content are delivered to the user through the Web Interface.

2.3.3 Continuous Learning Flow

1. User interactions and feedback are captured through the Web Interface.
2. This interaction data is stored in MongoDB for future reference.
3. The system analyzes patterns in user behavior and feedback to refine recommendation strategies.
4. Updated user preferences and behavior patterns are incorporated into the user profile.
5. The Recommendation Engine adapts its strategies based on this continuous learning process.

Architecture Decision Rationale: The layered microservices approach was chosen over a monolithic architecture to facilitate independent scaling of components, especially the computationally intensive AI services. This design also allows for graceful degradation if certain services are unavailable and enables the system to switch between different LLM providers based on availability and specific use-case requirements.

3. Frontend Details

3.1 Frontend Framework and Libraries

The Multi-Modal Financial Advisor Chatbot frontend is built using modern web technologies to create an intuitive, responsive, and engaging user interface. The primary framework and supporting libraries include:

Technology	Version	Purpose
React.js	18.x	Core frontend library for building the user interface with reusable components
Material-UI	5.x	Component library providing consistent, professional UI elements following Material Design principles
React Router	6.x	Handling client-side routing between different views of the application
React Markdown	8.x	Rendering formatted advisory documents and educational content with proper styling
Axios	1.x	HTTP client for making API requests to the FastAPI backend
Redux Toolkit	1.9.x	State management for complex application state and global user data
React Query	4.x	Data fetching, caching, and state management for API interactions

3.2 UI Structure and Components

The user interface is structured around several key pages and components, each serving a specific purpose in the financial advisory workflow:

3.2.1 Main Pages

- **Landing Page**: Introduces the financial advisor chatbot and its capabilities, featuring:
  - Brief overview of the platform's functionality
  - Authentication options (login/register)
  - Testimonials and trust indicators
- **Authentication Pages**: Includes login and registration forms with:
  - Form validation for user inputs
  - Password strength indicators
  - OAuth integration for social login options
- **Onboarding Dashboard**: Guides new users through the initial profile setup:
  - Step-by-step onboarding process
  - Progress indicators
  - Interactive questionnaires for financial goal setting
- **Chat Interface**: The primary interaction point with the AI advisor:
  - Chat history display with user and bot messages
  - Message input area with support for text and file uploads
  - Typing indicators and status updates
  - Context-aware suggestions
- **Advisory Document Viewer**: Displays personalized financial advice documents:
  - Rich text formatting with Markdown support
  - Document categorization and filtering
  - Save and export options
- **Product Recommendation Dashboard**: Showcases personalized financial product recommendations:
  - Card-based product displays with key features
  - Comparison tools for multiple products
  - Integration with product-specific chatbots
- **User Profile Management**: Allows users to view and update their information:
  - Personal and financial information settings
  - Preference management
  - Privacy controls

3.2.2 Key Components

- **ChatComponent**: Handles the display and interaction within the chat interface:

```
// ChatComponent.jsx
import React, { useState, useEffect, useRef } from 'react';
import { Box, TextField, Button, Paper, Typography, CircularProgress } f
import { useSelector, useDispatch } from 'react-redux';
import { sendMessage, fetchChatHistory } from '../slices/chatSlice';

const ChatComponent = () => {
  const [message, setMessage] = useState('');
  const chatEndRef = useRef(null);
  const dispatch = useDispatch();
  const { messages, loading } = useSelector((state) => state.chat);
  const { user } = useSelector((state) => state.auth);

  useEffect(() => {
    // Fetch chat history when component mounts
    if (user) {
      dispatch(fetchChatHistory());
    }
  }, [dispatch, user]);

  useEffect(() => {
    // Scroll to bottom of chat when messages change
    chatEndRef.current?.scrollIntoView({ behavior: 'smooth' });
  }, [messages]);

  const handleSendMessage = (e) => {
    e.preventDefault();
    if (message.trim()) {
      dispatch(sendMessage({ content: message, sender: 'user' }));
      setMessage('');
    }
  };

  // Component rendering logic...
};
```

- **RecommendationCard**: Displays individual financial product recommendations:

```
// RecommendationCard.jsx
import React from 'react';
import { Card, CardContent, CardActions, Button, Typography, Chip, Box }
import { useNavigate } from 'react-router-dom';

const RecommendationCard = ({ product, matchScore }) => {
  const navigate = useNavigate();

  const handleLearnMore = () => {
    navigate(`/products/${product.id}`);
  };

  const handleChatAbout = () => {
    navigate(`/chat/product/${product.id}`);
  };

  return (


         85 ? "success" : matchScore > 70 ? "primary" : "default"}
        />


        {product.name}
        {product.description}
        {/* Additional product details... */}


        Learn More
        Chat About This Product


  );
};
```

- **DocumentViewer**: Renders personalized advisory documents using Markdown:

```
// DocumentViewer.jsx
import React from 'react';
import ReactMarkdown from 'react-markdown';
import { Paper, Typography, Divider, Box } from '@mui/material';
import remarkGfm from 'remark-gfm';

const DocumentViewer = ({ document }) => {
  if (!document) return null;

  return (

      {document.title}

        Generated on: {new Date(document.createdAt).toLocaleDateString()




          {document.content}



  );
};
```

3.3 Frontend-Backend Integration

The frontend integrates with the backend through a RESTful API client service that abstracts the communication details:

```
// apiService.js
import axios from 'axios';

const API_BASE_URL = process.env.REACT_APP_API_BASE_URL || 'http://localhost
```

```
// Create axios instance with default config
const apiClient = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});

// Request interceptor for adding auth token
apiClient.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);

// Response interceptor for error handling
apiClient.interceptors.response.use(
  (response) => response,
  (error) => {
    // Handle authentication errors
    if (error.response && error.response.status === 401) {
      localStorage.removeItem('token');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);

// API service methods
const apiService = {
  // Auth endpoints
  login: (credentials) => apiClient.post('/auth/token', credentials),
  register: (userData) => apiClient.post('/auth/register', userData),
```

```
  getCurrentUser: () => apiClient.get('/auth/me'),

  // Chat endpoints
  sendMessage: (message) => apiClient.post('/chat/message', message),
  getConversations: () => apiClient.get('/chat/conversations'),
  getConversation: (id) => apiClient.get(`/chat/conversations/${id}`),

  // Recommendations endpoints
  getRecommendations: () => apiClient.get('/recommendations'),
  getRecommendationHistory: () => apiClient.get('/recommendations/history'),
  provideFeedback: (feedback) => apiClient.post('/recommendations/feedback',

  // Onboarding endpoints
  startOnboarding: () => apiClient.post('/onboard/start'),
  updateOnboarding: (data) => apiClient.post('/onboard/update', data),
  completeOnboarding: () => apiClient.post('/onboard/complete'),

  // Advisory document endpoints
  getDocuments: () => apiClient.get('/advisory/documents'),
  getDocument: (id) => apiClient.get(`/advisory/documents/${id}`),

  // Product-specific chat
  sendProductMessage: (productId, message) =>
    apiClient.post(`/products/${productId}/chat`, message),

  // Image upload and analysis
  uploadImage: (formData) =>
    apiClient.post('/images/upload', formData, {
      headers: { 'Content-Type': 'multipart/form-data' }
    }),
};

export default apiService;
```

3.4 Design Principles and UI/UX Guidelines

The frontend design follows several key principles and guidelines to ensure a consistent and intuitive user experience:

3.4.1 Design System

- **Material Design Integration**: Follows Google's Material Design principles through Material-UI components.
- **Responsive Layout**: Uses CSS Grid and Flexbox to ensure proper rendering across devices of different screen sizes.
- **Consistent Typography**: Implements a hierarchical type system with defined font sizes, weights, and line heights.
- **Color System**: Utilizes a defined palette with primary, secondary, and accent colors, along with semantic colors for success, warning, and error states.

3.4.2 Accessibility Considerations

- WCAG 2.1 AA compliance for all interactive elements
- Proper contrast ratios for text and background colors
- Keyboard navigation support for all interactive elements
- Screen reader compatibility with ARIA labels and roles
- Focus management for improved keyboard accessibility

3.4.3 UX Patterns

- **Progressive Disclosure**: Information is revealed gradually, starting with essential elements and providing additional details on demand.
- **Contextual Help**: Tooltips and information icons provide explanations for financial terms and concepts.
- **Clear Feedback**: All user actions receive immediate visual feedback, with loading indicators for asynchronous operations.
- **Guided Journeys**: Step-by-step processes with clear progress indicators, especially for complex operations like onboarding.
- **Consistency**: Common actions maintain consistent positioning and behavior throughout the application.

Design Decision Highlight: The conversational UI was designed to balance the familiarity of messaging apps with the specialized needs of financial advisory. This involved using larger message bubbles to accommodate complex financial content and incorporating visual elements like charts and comparison tables directly within the chat flow to maintain context while presenting complex information.