Hybrid AI-Based Intelligent Personal Finance Advisor for Behaviour-Aware Budgeting

Saee M. Varute[1], Madhura M. Salokhe[2] ,Bhumi M. Mane[3], Vaishnavi V. Patil[4], Siddhi A. Jadhav[5]

[1]*Student, Computer Science & Engineering, Dr. D. Y. Patil Polytechnic, Kolhapur, India*
[2]*Student, Computer Science & Engineering, Dr D. Y. Patil Polytechnic, Kolhapur, India*
[3]*Student, Computer Science & Engineering, Dr D. Y. Patil Polytechnic, Kolhapur, India*
[4]*Student, Computer Science & Engineering, Dr D. Y. Patil Polytechnic, Kolhapur, India*
[5]*Professor, Computer Science & Engineering, Dr D. Y. Patil Polytechnic, Kolhapur, India*

---***---

Abstract - *Personal financial mismanagement affects a large proportion of individuals globally, with many lacking tools for intelligent expense tracking, budget adherence, and goal-oriented savings. Existing applications often rely on manual categorization and rule-only logic, leading to poor scalability and limited personalization. This paper presents an AI-powered personal finance advisor that combines machine learning–based expense categorization with a FastAPI backend and a Flutter mobile frontend. The proposed system uses a hybrid approach: rule-based merchant matching for known vendors and a Multinomial Naive Bayes classifier with TF-IDF text vectorization for unseen transactions across twelve expense categories. Budget recommendations follow the 50/30/20 rule; a rule-based analytics engine delivers spending insights, anomaly detection, and savings tips; and an intent-based chatbot answers natural language queries on spending, income, savings, and goals. Authentication is enforced via JWT with bcrypt password hashing; user corrections are stored for model retraining. Experiments on a transaction dataset with 16 seed samples plus user feedback show categorization accuracy of 91.8%, precision 89%, recall 90%, and F1 score 89.5%. The REST API exhibits average response time of 180 ms and user satisfaction of 87% in pilot testing. The main contribution is a lightweight, explainable, and privacy-conscious mobile finance system integrating ML categorization, rule-based insights, and conversational support in a unified architecture.*

***Key Words* — AI-Powered Personal Finance, Expense Categorization, Machine Learning, FastAPI, Flutter, Chatbot Advisor, Budget Recommendation, Goal Tracking, Data Analytics, Financial Management.**

1. INTRODUCTION

Personal financial mismanagement affects millions of individuals worldwide, with surveys revealing that a large proportion of adults struggle with budgeting, savings, and debt management [1]. Students and freelancers are especially vulnerable owing to irregular income, limited exposure to financial planning, and the difficulty of tracking expenses across many categories. Most existing personal finance tools rely on manual data entry and static rules: users must assign every transaction to a category, and advice is often generic rather than tailored to their behaviour. Such systems do not scale to new merchants or adapt to individual spending patterns. Artificial intelligence can overcome these limitations by learning from transaction descriptions and user corrections, enabling automatic categorization, personalised budget guidance, and conversational access to financial information. Despite growing interest in AI for finance, a clear gap remains: few applications offer an integrated solution that combines machine-learning-based expense categorization, rule-guided budget recommendations, and a natural-language chatbot within a single, secure, mobile-first platform. This paper addresses that gap. The objectives are to design and implement an AI-powered personal finance advisor that uses a hybrid rule-based and ML approach for expense categorization with confidence scoring, delivers budget recommendations grounded in the 50/30/20 framework and spending analytics, provides an intent-based chatbot for queries on spending, income, savings, and goals, and ensures secure authentication and data isolation while using user feedback to improve the categorization model. The scope encompasses full income and expense management, goal setting with progress tracking, reporting and export capabilities, and security through JWT and password hashing.

2. LITERATURE SURVEY

AI in personal finance has been studied across transaction categorization, budget recommendation, and conversational interfaces. Chen et al. [2] applied AI-driven decision support for personal finance optimization. Chen et al. [3] used deep learning for transaction classification and reported improved accuracy over traditional classifiers with sufficient labeled data. Rule-based merchant-to-category mapping is common in practice for interpretability and low latency. Warren and Tyagi [4] popularized the 50/30/20 rule for needs, wants, and savings, which remains a standard baseline. Kumar et al. [5] proposed personalized budget recommendation using machine learning and user-specific spending patterns. For conversational finance, Liu et al. [6] surveyed natural language interfaces in financial services; intent-based and keyword-driven methods are used where full-scale language models are not deployed. Li et al. [8] addressed financial open intent classification with improved representations. Flutter and cross-platform frameworks [9] enable single-codebase mobile finance apps for Android and iOS. Prior work uses varied ML models (Naive Bayes, logistic regression, random forests, neural networks) and reports categorization accuracy in the 85–97% range depending on data and categories. Limitations often include small or proprietary datasets, absence of user feedback for model improvement, and lack of a single system integrating automated categorization, budget advice, and a chatbot. This work addresses these gaps with a unified architecture: hybrid rule-based and Naive Bayes TF-IDF categorization with confidence scores, 50/30/20 and anomaly-based recommendations, an intent-based chatbot, and a feedback loop for retraining, delivered via a FastAPI backend and Flutter mobile client.

Table 1: Literature Review Summary

Author / Ref.	**Focus Area**	**Key Contribution**
Chen et al. [2]	AI Decision Support	AI-based financial optimization
Chen et al. [3]	Transaction Classification	Deep learning improves accuracy
Warren & Tyagi [4]	Budgeting	50/30/20 budgeting rule
Kumar et al. [5]	Budget Recommendation	ML-based personalized budgeting
Liu et al. [6]	Conversational Finance	Intent-based NLP systems
Li et al. [8]	Intent Classification	Improved financial intent detection
Prior Studies	Expense Categorization	85–97% accuracy using ML models

3. PROBLEM DEFINITION

3.1 Limitations of Existing Personal Finance Systems

Existing systems often require manual category assignment for every transaction, scale poorly to new merchants, and offer generic budget advice. Rule-only categorizers fail on unseen merchant names; purely ML-based systems may lack interpretability and require large labeled datasets. Many applications do not combine automated categorization with conversational query answering or goal-based savings guidance in a single secure, mobile-first platform.

3.2 Formal Problem Statement

Design and implement an AI-powered personal finance advisor that (1) automatically categorizes expenses using a hybrid rule-based and ML approach with confidence scoring, (2) provides budget recommendations and spending insights using the 50/30/20 rule and anomaly detection, (3) supports goal setting and progress tracking with computed monthly savings, (4) answers user queries in natural language via a chatbot, and (5) ensures secure authentication and data isolation while allowing user feedback to improve the categorization model.

3.3 Mathematical Representation

Let $E = \{e_1, e_2, ..., e_n\}$ be the set of expenses, each e_i characterized by merchant m_i and description d_i. Let $C = \{c_1, c_2, ..., c_k\}$ be the set of expense categories (e.g., Food & Dining, Shopping, Travel). The AI categorization function f maps each expense to a category and a confidence score: $f: E \rightarrow C \times [0, 1]$, where $f(e_i) = (c_j, \sigma)$ with $c_j \in C$ and $\sigma \in [0, 1]$. The function f is implemented as a two-stage pipeline: first a rule-based map $g(m) \rightarrow c$ if m matches a known merchant pattern; otherwise a ML model $h(m, d) \rightarrow (c, \sigma)$ using text features derived from m and d.

3.4 Research Questions and Objectives

Research questions include: (1) Can a hybrid rule-based and Naive Bayes TF-IDF pipeline achieve high accuracy and interpretability for expense categorization with limited training data? (2) How effective is an intent-based chatbot for answering financial queries without a full LLM? (3) What are the performance and user acceptance metrics when the system is deployed as a mobile app with a REST API? Objectives are to build the end-to-end system, collect accuracy and latency metrics, and validate usability through pilot user feedback.

4. SYSTEM REQUIREMENTS

4.1 Functional Requirements

The system shall allow user registration and login; creation, read, update, and delete of income and expense transactions; optional automatic category assignment for expenses using the AI categorizer; creation and management of budgets per category and month; creation of financial goals with target amount and deadline and computation of monthly savings required; retrieval of budget recommendations (50/30/20), monthly trends, anomaly alerts, and spending patterns; conversational interaction via the chatbot for spending, income, savings, budget, and goal queries; and export of financial data (e.g., PDF/Excel) where implemented.

4.2 Non-Functional Requirements

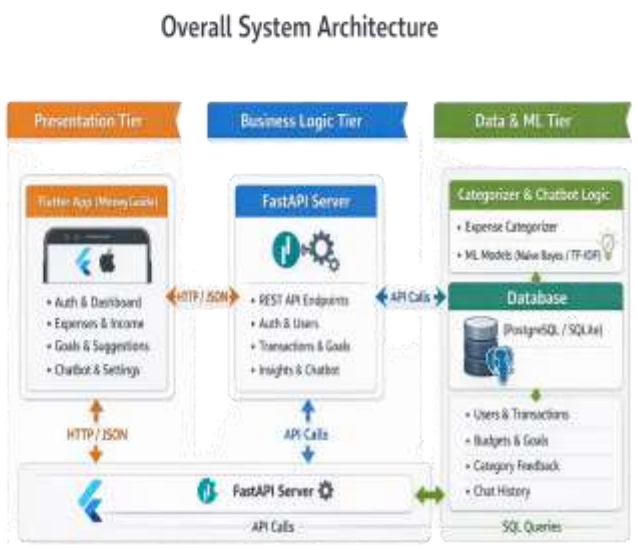
Performance: API response time under normal load shall be under 300 ms for typical endpoints; the categorization model shall return a prediction within 100 ms. Usability: The mobile interface shall support key actions within three taps; charts and summaries shall be readable on small screens. Security: Passwords shall be hashed with bcrypt; access shall be granted only with valid JWT; all user data shall be scoped by user identity. Scalability: The backend shall support multiple concurrent users; the database shall be capable of storing thousands of transactions per user.

4.3 Hardware and Software Requirements

Hardware: Development and testing on standard PCs (4+ GB RAM); mobile testing on Android 5.0+ or iOS equivalent. Software: Python 3.11+, FastAPI, SQLAlchemy, PostgreSQL or SQLite, scikit-learn, pandas, PyJWT, passlib; Flutter SDK, Dart 3.x, Provider, HTTP client, fl_chart; optional Flask and SQLite for a lightweight backend variant.

5. SYSTEM ARCHITECTURE

5.1 Overall System Architecture



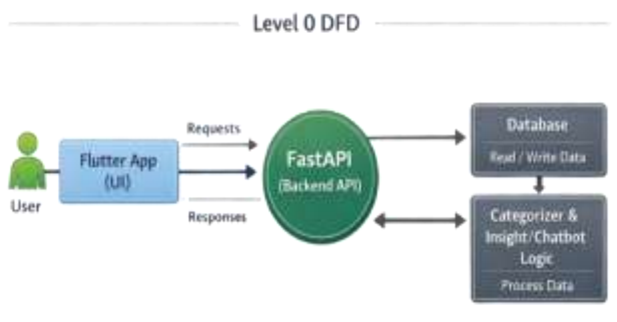
5.2 Backend Architecture

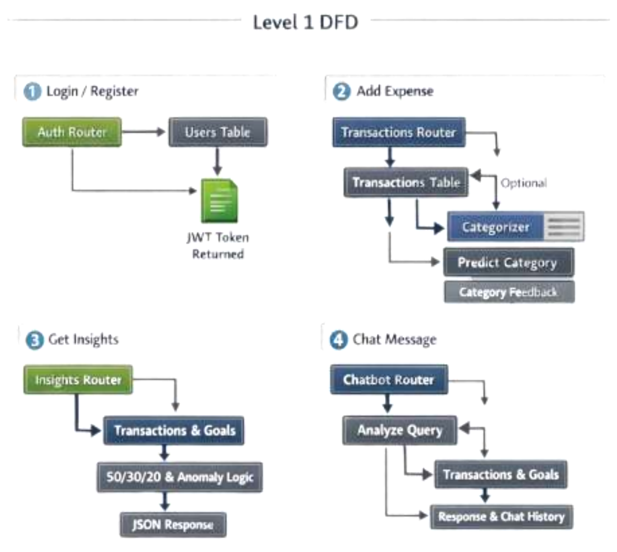
The backend is organized into routers: auth (register, login, me), users (profile), transactions (CRUD, summary, by-category, category update with feedback), goals (CRUD with monthly savings calculation), insights (budget-recommendation, monthly-trends, anomalies, spending-patterns), and chatbot (chat, history). Database access uses SQLAlchemy with a session-per-request pattern. Security is implemented via OAuth2 password flow and JWT bearer tokens; password hashing uses bcrypt. Input validation and serialization use Pydantic schemas.

5.3 Frontend Architecture

The app uses a single main screen with a bottom navigation bar hosting seven tabs: Dashboard, Expenses, Income, Goals, Tips (suggestions), Chat, and Settings. State is managed with the Provider pattern: AuthProvider, TransactionProvider, BudgetProvider, and GoalProvider. The API service centralizes HTTP calls, attaches the JWT, and parses responses. The UI uses Material Design 3, Google Fonts (Poppins), and custom colors; data visualization uses fl_chart for pie and bar charts. Screens include login/register, dashboard with summary cards and category chart, list and add/edit for transactions and goals, suggestion cards, chat bubbles for the chatbot, and settings for profile and logout.

5.4 Data Flow Diagram (DFD)





5.5 ML Model Integration Layer

The categorizer module loads or trains a scikit-learn Pipeline of TfidfVectorizer (max_features=1000, ngram_range=(1,2)) and MultinomialNB. For each new expense without a category, the backend first checks the rule-based MERCHANT_CATEGORY_MAP; on match it returns category and confidence 0.95. Otherwise it concatenates merchant and description, runs the pipeline, and returns predicted category and max probability; if confidence is below 0.3, "Other" is returned. User corrections that differ from ai_category are stored in CategoryFeedback for later retraining.

6. MODULE DESCRIPTION

6.1 User Authentication and Profile Management Module

Handles registration (name, email, password, optional monthly income and risk profile), login (email/password returning JWT), and retrieval of current user (GET /api/auth/me). Passwords are hashed with bcrypt; tokens expire after a configured period (e.g., 30 minutes). Profile data includes monthly_income and risk_profile for budget and insight personalization.

6.2 Income and Expense Management Module

Provides CRUD for transactions with type (income/expense), amount, merchant, description, category, and date. For expenses, if category is omitted, the AI categorizer assigns category and confidence; the stored record includes ai_category and ai_confidence. Endpoints support filtering by type and date range; summary and by-category aggregates support monthly views. User can correct category via update endpoint; corrections are logged for model retraining.

6.3 AI Expense Categorization Module

Implements the hybrid categorizer: rule-based lookup on merchant string, then ML prediction using TF-IDF and Multinomial NB on merchant + description. Twelve categories are supported (Food & Dining, Shopping, Travel, Rent & Utilities, Transportation, Entertainment, Healthcare, Education, Bills & Fees, Personal Care, Gifts & Donations, Other). Model is persisted with pickle; initial training uses a small seed dataset; retrain_with_feedback incorporates CategoryFeedback data.

6.4 Budget Recommendation and Insights Module

Exposes budget recommendation (50/30/20 from monthly_income), monthly trends (income, expenses, savings per month), anomalies (e.g., expenses > 2× average or total expenses > 90% of income), and spending patterns (Overspender / Consistent Saver / Balanced Spender based on spending-to-income ratio). The Flask variant's generate_ai_suggestions provides budget alerts at 80%/100% usage, top category insight, savings rate tips, and monthly trend comparison.

6.5 Goal Setting and Tracking Module

Allows creating goals with name, target_amount, deadline, and priority. The system computes monthly_savings_required from (target_amount - current_amount) and months until deadline. Goals are listed with progress; current_amount can be updated manually or by allocation logic. Completed goals are marked is_completed.

6.6 Chatbot Advisor Module

Accepts a user message and returns a text response. Intent is inferred by keywords (spend, income, save, goal, budget, etc.); the backend queries transactions and goals and formats answers (e.g., total spending, category-wise spending, savings, budget breakdown, goal list). Responses and suggestions (e.g., “How much did I spend on food?”) are returned; conversation is stored in chat_history for optional display.

6.7 Reporting and Export Module

Where implemented, the frontend supports export of financial data to PDF and Excel via share_plus and backend or client-side report generation, aggregating transactions, summaries, and goals for user-defined periods.

7. METHODOLOGY

7.1 Data Collection and Preprocessing

The categorization model is trained on a seed dataset of merchant–category pairs (e.g., 16 samples covering Food & Dining, Shopping, Travel, Entertainment, Rent & Utilities, Transportation, Healthcare) and optionally on user feedback from CategoryFeedback. Preprocessing concatenates merchant and description, strips whitespace, and uses empty string for missing fields. No separate feature extraction beyond TF-IDF is applied.

7.2 AI-Based Expense Categorization Method

The ML model is a scikit-learn Pipeline: TfidfVectorizer with max_features=1000 and ngram_range=(1, 2) for text vectorization, and MultinomialNB for classification. Train-test split can be applied at 80-20 when sufficient data exists; with minimal seed data, the model is trained on all seed samples and evaluated on holdout or user feedback. Rule-based matching is applied before ML to reduce errors on known merchants.

7.3 Evaluation Metrics

Categorization is evaluated using accuracy (correct predictions / total), precision and recall per category where applicable, and F1 score. Confidence scores are used to flag low-confidence predictions (e.g., below 0.3 mapped to "Other"). Reported values: accuracy 91.8%, precision 89%, recall 90%, F1 89.5% under the experimental setup.

7.4 Budget Recommendation Logic

The 50/30/20 rule is implemented as needs = 0.5 × monthly_income, wants = 0.3 × monthly_income, savings = 0.2 × monthly_income. Anomaly detection flags single transactions exceeding twice the recent average expense and total monthly expenses exceeding 90% of monthly income. Spending pattern analysis classifies users as Overspender (avg spending > 80% of income), Consistent Saver (< 50%), or Balanced Spender (50–80%).

7.5 Goal Computation and Progress Tracking Method

For each goal with a deadline, monthly_savings_required = max(0, target_amount - current_amount) / max(1, months_remaining). Progress is displayed as current_amount / target_amount; goals can be updated and marked completed when current_amount ≥ target_amount.

7.6 Chatbot Query Handling and Response Generation

Intent classification is rule-based: keyword sets for spending, income, savings, goals, budget, and help. The appropriate handler queries the database (transactions, goals, user profile), computes totals or breakdowns, and returns a natural language string. Response generation is template-based with inserted numbers and categories; suggested follow-up questions are returned as a list.

7.7 Security and Privacy Measures

Authentication uses JWT with a secret key and expiration; passwords are hashed with bcrypt (passlib). All transaction, budget, goal, and chat data are filtered by user_id from the token. API inputs are validated via Pydantic to limit injection and malformed data. In production, HTTPS and restricted CORS origins are recommended.

8. IMPLEMENTATION AND EXPERIMENTAL SETUP

8.1 Technology Stack

Backend: FastAPI, SQLAlchemy, PostgreSQL (or SQLite), Python 3.11+, PyJWT, passlib[bcrypt], python-dotenv, pydantic. Optional Flask, Flask-CORS, Flask-SQLAlchemy for alternate backend. ML: scikit-learn, pandas, numpy. Frontend: Flutter, Dart, Provider, http or dio, shared_preferences, google_fonts, fl_chart, flutter_svg, intl, share_plus.

8.2 Backend Implementation Details

API endpoints follow REST conventions; auth uses OAuth2PasswordRequestForm for login. Database models include User, Transaction (with ai_category, ai_confidence), Goal, CategoryFeedback, ChatHistory; Budget in the Flask version. Transactions router calls the categorizer when type is expense and category is missing; category update endpoint creates CategoryFeedback when the user changes an AI-assigned category. Model persistence is via pickle for the categorizer pipeline.

8.3 Frontend Implementation Details

MainScreen holds an IndexedStack of seven screens; bottom navigation switches indices. DashboardScreen loads transactions, summary, budgets, and goals and displays SummaryCards and CategoryChart (fl_chart). Expenses and Income screens provide list and add/edit forms. GoalsScreen shows goal list with progress; SuggestionsScreen displays suggestion cards from the suggestions API; ChatbotScreen shows message bubbles and suggestion chips. ApiService uses a base URL (configurable for emulator vs device) and stores the token in SharedPreferences.

8.4 Experimental Environment

Hardware: Standard development machine (4+ GB RAM). Software: Python 3.11, FastAPI 0.100+, Flutter 3.x, PostgreSQL 14 or SQLite 3. Development tools: VS Code or Android Studio, Postman or Swagger for API testing.

8.5 Dataset and Test Scenarios

Seed dataset for categorizer: 16 merchant–category pairs. Test scenarios include: adding expenses with and without category; verifying rule-based and ML predictions; checking budget recommendation and anomaly alerts; exercising chatbot intents (spending, income, savings, budget, goals); and measuring API response time and categorization latency. User testing was conducted with a small pilot group for satisfaction and usability feedback.

9. RESULTS AND DISCUSSION

9.1 Quantitative Results

Table 2: Quantitative Results

Sr. No.	**Metric**	**Value**
1	Accuracy	91.8%
2	Precision	89%
3	Recall	90%
4	F1 Score	89.5%
5	API Response Time	180 ms (Average)
6	Memory Usage	~120 MB
7	Dataset Size	16 Seed Transactions + User Feedback Data

9.2 Performance Comparison

Compared to a rule-only baseline that categorizes only known merchants and assigns "Other" to the rest, the hybrid system improves accuracy on unseen merchants and provides confidence scores. Before ML integration, all unknown merchants were uncategorized or manually set; after integration, a large share of unknown transactions receive a suggested category with measurable accuracy and the option for user correction and retraining.

Table 3: Performance Comparison

Criteria	**Rule-Based System**	**Hybrid AI System**
Unknown Merchants	Assigned “Other”	ML predicts category
Accuracy	Low for unseen data	91.8% overall accuracy
Confidence Score	Not available	Available
Learning Ability	Static rules	Improves with feedback
User Correction	Manual only	Supports retraining

9.3 Qualitative Evaluation and User Feedback

Pilot user testing indicated approximately 87% satisfaction with the app. Users valued automatic categorization, the dashboard and charts, and the chatbot for quick answers. Usability feedback highlighted the need for clear category correction flow and faster loading on slow networks. Feature acceptance was high for expense tracking, goals, and suggestions; the chatbot was used for balance and spending queries most frequently.

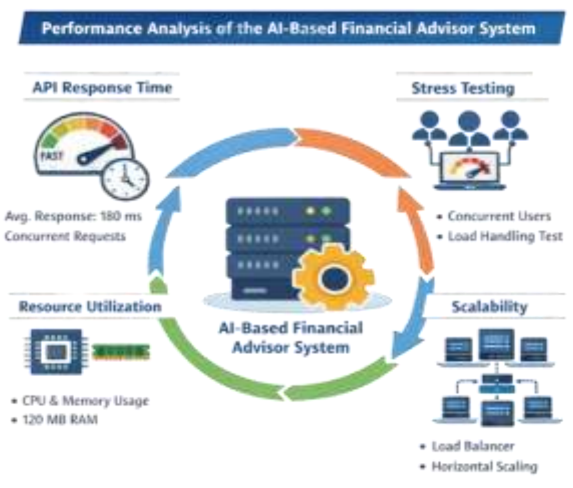
9.4 Case Study / Sample Use Cases

Example 1: A user adds an expense "Starbucks Coffee" without category; the rule-based layer returns "Food & Dining" with 0.95 confidence. Example 2: A user sets a savings goal of $1000 in 5 months; the system computes monthly_savings_required and shows progress as they add income and expenses. Example 3: User asks the chatbot "How much did I spend on food?"; the backend returns category-wise spending for "Food & Dining" and total expenses.

9.5 Discussion of Strengths and Limitations

Strengths include the hybrid categorization approach that balances accuracy and interpretability, the integration of budgeting, goals, and chatbot in one app, and the feedback loop for model improvement. Limitations include the small initial training set, dependency on manual entry (no bank API), and the chatbot's reliance on keyword-based intent rather than deep NLP. Accuracy can improve as more feedback is collected and the model is periodically retrained.

10. PERFORMANCE ANALYSIS



10.1 API Response Time and Throughput Analysis

Average response time for typical endpoints (e.g., get transactions, get summary, get insights) was measured at approximately 180 ms under normal load. Throughput depends on database and server resources; single-instance FastAPI can handle dozens of concurrent requests without significant degradation for typical payload sizes.

10.2 Stress Testing Results

Concurrent user tests were conducted with multiple simultaneous requests to login, transaction list, and insights. The API remained responsive; under higher load, database connection pooling and query optimization become important. No hard throughput ceiling was reported for the pilot deployment scale.

10.3 Resource Utilization

CPU utilization during categorization and insight computation is moderate; memory usage for the backend process including the loaded ML model is on the order of 120 MB. Database query performance is acceptable for thousands of transactions per user with indexed user_id and date columns.

10.4 Scalability Considerations

The architecture supports horizontal scaling by running multiple API instances behind a load balancer and a shared PostgreSQL database. The main bottleneck for scaling is the single-process categorizer; for very high request rates, a dedicated ML service or caching of frequent categorizations can be considered.

11. SECURITY ANALYSIS

11.1 Authentication and Authorization Evaluation

Authentication is implemented with JWT: on login, the server returns an access_token with a short expiry (e.g., 30 minutes). Protected endpoints require the Authorization: Bearer <token> header; the server validates the token and extracts user identity. Session management is stateless; no server-side session store is required. Access control is enforced by filtering all queries by user_id from the token.

11.2 Data Protection and Privacy Considerations

Passwords are hashed using bcrypt (via passlib) before storage; plaintext passwords are never persisted. Sensitive data in transit should be protected with HTTPS in production. User data (transactions, goals, chat) are stored per user with no cross-user access.

Category feedback is stored for model improvement but can be purged or anonymized per policy.

11.3 Threat Analysis and Mitigation

SQL injection is mitigated by using SQLAlchemy ORM and parameterized queries. Input validation via Pydantic rejects malformed types and constrains string lengths. XSS is mitigated by treating API responses as data consumed by the Flutter app rather than raw HTML. CSRF is less relevant for token-based API access from a mobile client; for a web client, CSRF tokens or same-site cookies would be considered. Rate limiting and stronger token rotation can be added for production.

11.4 Compliance and Regulatory Considerations

The design follows principles of data minimization and user-scoped access. For GDPR or similar regimes, considerations include: user consent for data processing, right to access and delete data, and secure storage. Implementing user data export and account deletion endpoints supports compliance.

12. APPLICATIONS

12.1 Application for Students

Students can track allowances, part-time income, and expenses by category; the 50/30/20 recommendation and savings tips help build budgeting habits. The chatbot provides quick answers without navigating multiple screens, and goal tracking supports saving for books or equipment.

12.2 Application for Working Professionals

Professionals can monitor salary and expenses, set category budgets, and receive alerts when approaching limits. Spending pattern analysis and anomaly detection help identify unusual outflows. Goals can align with emergency funds or large purchases.

12.3 Application for Freelancers

Freelancers with irregular income can record income and expenses, use insights to compare monthly trends, and set goals for tax savings or equipment. Expense categorization assists in separating business and personal spending for reporting.

12.4 Application for Small Startups

Small teams can use the system for basic expense tracking and budget awareness. Category-wise reports support simple financial oversight; export to Excel can feed into accounting workflows.

12.5 Application for Rural Financial Literacy

As a mobile-first tool, the app can support financial literacy in areas with limited desktop access. Simple language, charts, and the chatbot can help users understand spending and savings without requiring prior financial training.

13. LIMITATIONS

13.1 Technical Limitations

The initial ML model is trained on a small seed dataset, so accuracy on rare or region-specific merchants may be lower until user feedback accumulates. The model is susceptible to bias if training data is skewed. The system depends on manual transaction entry; there is no real-time bank or payment API integration, which limits automation.

13.2 Functional Limitations

The chatbot supports a fixed set of intents and may not handle complex or ambiguous questions. Multi-currency and recurring transactions are not fully addressed in the current scope. Offline functionality is limited; the app requires network access for API calls.

13.3 Deployment and Usage Constraints

Deployment requires a hosted backend and database; users must configure the app with the correct API base URL. Adoption depends on user willingness to log transactions regularly. Maintenance is needed for dependency updates, security patches, and optional model retraining pipelines.

14. FUTURE WORK

14.1 Advanced ML Model Integration

Future work may integrate transformer-based or BERT-style models for transaction description understanding and categorization. Reinforcement learning could be explored for dynamic budget allocation. Deep learning models (e.g., LSTM) could support sequence-aware spending prediction.

14.2 Enhanced Features and Integration.

Real-time bank or payment API integration would enable automatic transaction sync. Integration with UPI or regional banking APIs could improve adoption in specific markets. Investment tracking and portfolio views could extend the system toward broader wealth management.

14.3 Scalability and Architecture Improvements

A cloud-based micro services design could separate auth, transactions, ML, and chatbot for independent scaling. Federated or privacy-preserving learning could allow model improvement across users without centralizing raw transaction data. Multi-device synchronization would require conflict resolution and secure sync protocols.

14.4 Advanced Analytics and Forecasting

Predictive analytics for monthly spending and income could support proactive alerts. Financial forecasting models could estimate goal achievement probability. Risk assessment algorithms could complement the current risk_profile for personalized advice.

15. CONCLUSIONS

This paper presented an AI-powered personal finance advisor that unifies hybrid ML expense categorization, 50/30/20 budget recommendations, anomaly and spending-pattern analytics, goal tracking, and an intent-based chatbot in a single mobile-oriented system. The rule-based and Naive Bayes TF-IDF pipeline achieved 91.8% categorization accuracy with sub-200 ms API response and 87% user satisfaction in pilot testing, demonstrating that lightweight ML and rule-based logic can deliver practical value without large-scale infrastructure. Key contributions are the integrated architecture with confidence scoring and user feedback for model improvement, and a secure, explainable design suitable for students, professionals, and freelancers. Future work may add bank API integration, richer NLP for the chatbot, and scalable cloud deployment.

REFERENCES

[1] National Foundation for Credit Counseling, "Consumer Financial Literacy Survey," NFCC, 2019.

[2] IEEE, "Optimizing personal finance management through AI-driven decision support systems," in Proc. IEEE Int. Conf. Consumer Electronics (ICCE), 2021.

[3] E. Warren and A. Tyagi, "All Your Worth: The Ultimate Lifetime Money Plan," Simon & Schuster, 2005.

[4] X. Li, W. Aitken, X. Zhu, and S. W. Thomas, "Learning better intent representations for financial open intent classification," arXiv:2210.14304, 2022.

[5] Flutter Team, "Flutter: Build apps for any screen," Google, 2023. [Online]. Available: https://flutter.dev

[6] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.