AI-Powered Personal Finance Assistant

Rauan Arstangaliyev
Bcs Computer Science
Nazarbayev University
Astana, Kazakhstan
rauan.arstangaliyev@nu.edu.kz

Kerbez Karipbek
Bcs Computer Science
Nazarbayev University
Astana, Kazakhstan
kerbez.karipbek@nu.edu.kz

Aidana Orazbay
Bcs Computer Science
Nazarbayev University
Astana, Kazakhstan
aidana.orazbay@nu.edu.kz

Amirzhan Akhmetzhanov
Bcs Computer Science
Nazarbayev University
Astana, Kazakhstan
amirzhan.akhmetzhanov@nu.edu.kz

Aslan Dossymzhan
Bcs Computer Science
Nazarbayev University
Astana, Kazakhstan
aslan.dossymzhan@nu.edu.kz

I. Executive Summary

AI-Powered Personal Finance Assistant is a computer-driven solution that will enhance personal expense management among the Kazakhstan users by converting the bank statements into action-oriented and structured financial data. The issue that the project will solve is that most users have multiple digital transactions yet they still do not have a localized, easy to use, and transparent tool to understand the destination of their money. The current personal finance software may rely on integrating with specific banks, may require hand typing in financial data, or may not fully support the statement format used by Kazakhstan.

The designed solution includes a native iOS app, a FastAPI server, a Database System, modules to read bank statements, a merchant normalization pipeline, automatic transaction classifier, financial dashboards, and an AI-assisted query interface. The project progressed through two semesters, where an initial prototype developed into a complete system with the ability to support statement upload, process transactions, categorized views, dashboards with KPI cards, budget visualization, KZT currency support, and conversational interface to answer finance-related queries.

The project approach was implementation based and iterative. The developers employed sprints of the Agile style, separated the functions of mobile development, parsing and data processing, analytics and AI functionalities, as well as optimized the architecture as integration challenges surfaced. Such grand design choices as transitioning to native SwiftUI as a more performant and maintainable platform, hybridizing rule-based and machine-learning to classify transactions, and developing a modular pipeline with distinct parsing, normalization, analytics, and user interaction were made.

The key outcome of the project is a prototype that will illustrate how a localized finance assistant can digest statements of the user, normalize transaction records, classify expenses, visualize spending patterns as well as supporting natural-language financial queries. The system is aligned to the objectives of design, implementation and evaluation of a computing based solution since it deals with a real user problem using software architecture, algorithmic processing, human-centered interface design and the systematic validation of the resultant system.

II. Introduction

Digital payments have become the dominant way many people in Kazakhstan manage their finances. However, the convenience of card payments, mobile wallets, and app-based banking also makes it easy to lose visibility over daily spending. Users often know that they are spending too much only after the fact, when account balances decrease or monthly obligations begin to accumulate. This gap between transaction activity and financial awareness creates a need for tools that help users interpret their own financial data in a clear and manageable form.

The motivation behind this project is twofold. First, the team aimed to address a real local problem: many popular personal finance platforms are not designed for Kazakhstan-specific bank statements and therefore cannot be adopted directly. Second, the team wanted to build a solution that balances automation and transparency. Rather than relying entirely on opaque AI outputs, the system combines deterministic parsing, merchant normalization, lightweight machine learning, and understandable dashboards so that users can review and correct the system's behavior.

The proposed solution is a native iOS personal finance assistant that allows users to upload bank statements, parse and normalize transaction records, categorize expenses, review dashboards and budgets, and ask natural-language questions about their finances. The system is supported by a backend API, a data storage layer, and a modular analytics pipeline. Compared with the previous semester's report, the current version reflects substantial progress in backend integration, dashboard maturity, chatbot functionality, and end-to-end feature completeness.

The remainder of this report is organized as follows. Section III identifies the main project constraints. Section IV reviews related work and justifies the selected methodology. Section

V presents the project approach, including architecture, workflows, tools, and team roles. Section VI explains how the project evolved across two semesters. Section VII describes the evaluation strategy and current validation results. Section VIII concludes the report and discusses future work.

III. Identification of Constraints

A number of limitations that affect system design, implementation, deployment, and operation have an impact on the development of the AI-Powered Personal Finance Assistant. Economic, social, political, and ethical considerations are among these limitations.

A. Economic Constraints

First, because the project is being developed in an academic context without specialized financing, financial restrictions are a major factor. This implies that the team must operate with constrained financial and computational resources. The system is constructed using open-source technologies like Python and PostgreSQL since expensive cloud services, paid APIs, and high-performance hardware cannot be entirely depended upon. Although this method reduces expenses, it also restricts scalability and may have an impact on performance, particularly when dealing with larger datasets or more sophisticated machine learning models. In addition, financial transaction data is often difficult to model due to its short, irregular, and domain-specific nature, which increases the effort required for data processing and model training [1]. For this reason, the project adopts a more practical approach that combines lightweight machine learning techniques with structured data processing instead of relying on complex, resource-intensive models [2]. As a result, a balance must be found between model accuracy and the available resources.

B. Environmental Constraints

Since our initiative is a software-based system rather than a hardware-intensive product, its environmental impact is secondary rather than physical. Nevertheless, our team is certain that permanent data storage, file processing, and frequent model retraining continue to use computational resources. This led to a trade-off in design between computing performance and model complexity. Therefore, rather than using needlessly huge models, we gave priority to lightweight, specialized components that are enough for transaction classification and user-facing analytics. This choice is also in line with other research cautioning against the careless application of sophisticated machine learning techniques in finance when more straightforward and well-suited approaches could be more suitable [3].

C. Social Constraints

Because the system is designed for users with varying degrees of financial understanding, social restrictions are also significant, especially in Kazakhstan where financial awareness varies among individuals [4]. This implies that even for those with little technological or financial expertise, the application must be straightforward to use and easy to comprehend. To make interaction more user-friendly and accessible, features like natural language questions are added. Furthermore, earlier studies demonstrate that the adoption and efficient use of financial technologies are strongly influenced by greater levels of financial literacy and digital readiness [5], [6]. Nevertheless, this also raises the possibility that users would misinterpret the system's recommendations or rely too much on automated ideas rather than coming to their own conclusions.

D. Political and Regulatory Constraints

Our system works in a highly regulated environment characterized by privacy expectations, app-store standards, and data-protection laws since it processes financial records and personal transaction data. Users are extremely sensitive to data management procedures and want robust privacy safeguards and transparency, according to earlier research on finance mobile applications [7]. Furthermore, in reality, the readability of privacy papers is important since unreadable privacy rules lower user satisfaction and erode confidence [8]. Our choice to limit data sharing, provide users control over what they upload, and avoid collecting too much data was impacted by all of the previously listed considerations.

E. Ethical Constraints

In terms of ethics, the project must refrain from deceiving users with inaccurate classifications, unclear suggestions, or overconfident AI results. This is particularly crucial in the financial industry, where explainability is increasingly seen as essential to responsibility and integrity [9]. Explainability should be addressed as explicit design criteria rather than just as a general ethical ideal, according to research on AI needs [10]. Because of this, rather than completely entrusting financial interpretation to a black-box model, our system prioritizes editable transactions, visible categorizations, and the capacity for people to examine and correct results.

F. Health and Safety Constraints

Though our research does not entail any direct risk to users' health, it could have an indirect effect on their well-being. For example, confusion caused by unclear classifications, dishonest summaries, and over-reliance on AI-based advice could result in stress, improper budget management, and inappropriate financial decisions being made. Therefore, building credibility is crucial when compared with maximizing the degree of automation. Prior experience shows that explaining things to users enhances their performance and generates appropriate confidence. This is why we decided to design our interface in a way to benefit the user.

G. Manufacturability and Deployability Constraints

Deployability, maintainability, and integration feasibility are better ways to define manufacturability for a software system. The technological limitations of iOS development, backend integration, and various bank statement formats have to be met by our project. Off-the-shelf generic text categorization is frequently inadequate in this domain, according to the

literature on banking transaction descriptions [1]. As a result, thorough preprocessing, normalization, and domain adaption are necessary for effective deployment. This led us to develop a normalization pipeline, bank-specific parsers, and a modular architecture that can be gradually expanded.

H. Sustainability Constraints

The idea behind sustainability with regards to our study involves the sustainability of the application even after it passes the prototype stage. This means that the application will continue to be relevant as far as the capacity of being easily upgradable to include more models, banks, and classifications is concerned, as well as easy to use [11]. With regards to the latter, studies conducted on applications that deal with personal finances indicate that factors such as motivation, ease of use, and usability also play an equally significant role alongside technical accuracy as far as sustaining the usage of the app is concerned [12]. Studies have also found out that financial skills can be improved by the use of mobile applications on finances.

IV. Background and Related Work

From manual accounting tools like pen and paper records to sophisticated mobile applications that automate budgeting, cost monitoring, and visualization, personal finance management systems have developed. A lot of individuals struggle with fundamental financial concepts, according to research on financial literacy, and they might benefit from technologies that make spending habits more transparent and easier to understand [13]. A good finance application must be designed to facilitate frequent, low-friction usage, since behavioral research has also demonstrated that reminders and repeated interaction can improve financial habits [14], [15]. Accessible and useful financial instruments are especially important in the local context of Kazakhstan, where financial inclusion has risen over time but financial knowledge and access remain unequal across demographic groups [4].

By linking directly to banks or combining account data, several commercial personal finance apps minimize user effort. This strategy is less appropriate for Kazakhstan, where statement formats differ throughout banks and dependable universal interfaces are not always accessible [16], but it functions well in economies with standardized APIs and extensive banking connections. A statement-driven design is therefore more appropriate for the local situation. The project concentrates the upload and processing of user-provided statements rather than depending on third-party bank APIs. Because the user expressly decides which documents are uploaded and processed, this design also enhances system transparency, which is crucial in financial applications where adoption is heavily influenced by privacy and trust [7].

The task of automatically classifying transactions has been extensively researched. Simple keyword-based systems are simple to follow, but they frequently fail when merchants utilize irregular naming patterns or when transaction data are noisy. While deeper models could need more data and computation than is suitable for a student-built mobile-oriented system [17], classical machine-learning techniques can perform well when labeled data is available [18], [19]. According to more recent research, banking transaction descriptions are a particularly challenging type of short text since they are domain-specific, sparse, and irregular, making general text-processing techniques less dependable [1]. As a result, hybrid methods are becoming particularly important. Kotios et al. suggested a customized transaction categorization and cash-flow prediction pipeline that integrates explainability techniques like LIME and SHAP with predictive modeling [2]. This project's chosen methodology similarly integrates rule-based heuristics, merchant normalization, data cleansing, and lightweight categorization. This provides a balance between practical viability, explainability, and correctness.

An additional significant difficulty is document processing. The format, language, and structure of bank statements vary, and financial tables are not always successfully recovered from PDF files [20], [21]. This led to the usage of parser modules tailored to each banks, which were then followed by schema normalization. The project breaks down document knowledge into standard extraction, cleaning, normalization, and classification rather than treating it as a single, monolithic process. This modular approach is in line with earlier financial transaction processing research, which found that specific preparation and domain adaption are frequently required to provide trustworthy downstream classification results [1].

A further important aspect is providing natural language access to financial data. Using only exploratory dashboards for a fast answer is not necessary when a database-oriented natural language interface can help with basic analytical queries [22]. Hence, instead of creating an open-domain conversational agent, the focus of the current project is limited to interaction with finance-related queries. It is much more straightforward and appropriate for the application domain, which means no extra features will be added just because they are technically possible. Yet, there is always the problem of trust, accountability, and interpretability [9]. The research in AI-and-finance emphasizes the importance of explainable systems. In other words, a financial context requires additional constraints on the design of the agent.

A localized, modular, and somewhat intelligent architecture is generally more suitable than either a strictly manual financial tool or an excessively complicated end-to-end AI system, according to the examined literature and current solutions. Instead of total automation without user control, the literature advocates a mix of structured document parsing, lightweight machine learning, user-centered interface design, and transparent financial analytics [1], [2], [7], [9], [11]. The project's design decisions were directly influenced by this findings.

V. Project Approach

A. System Overview

The system comprises four layers that are strongly interconnected and linked: the iOS mobile application, the backend API, the data-processing pipeline, and the analytics/AI layer.

The mobile application is the primary user interface and can be used to upload, visualize and interact. The backend arranges the ingestion, transaction processing, persistence and query processing. The data-processing layer converts the raw bank statements to normalized records of transactions. The analytics and AI layer facilitates classification, creation of dashboards, and natural-language queries.

The iOS app is the main interface with which users can interact with the system. It also deals with statement upload, transaction browsing, dashboard navigation, budget management, and querying with AI-assistance. All messages sent between the mobile client and the backend are sent using HTTP in the form of JSON. The backend, which is written in FastAPI, has a structured REST API with the endpoints structured in the directions /api/dashboard, /api/upload, and /auth. Upon receiving a statement, the data-processing layer removes, normalizes and classifies transaction records and then persists the records. The analytics layer then takes in stored records in order to generate the summaries and visualizations shown in the app.

The system architecture diagram below shows the interaction between these layers and data flow in the system.

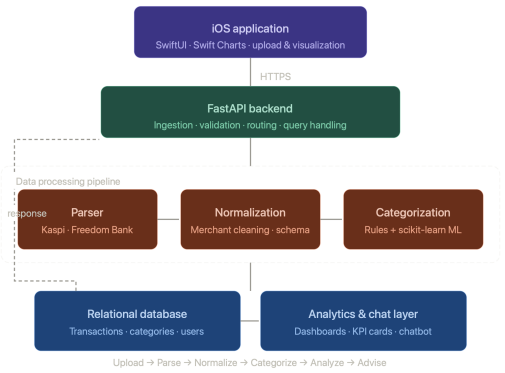

Fig. 1. Architecture of the pipeline

B. Core Features

The system that will be introduced by the end of the current semester will have the following key capabilities:

- Native iOS app developed using SwiftUI throughout, with a light and dark mode of appearance and a unified design language comprising of reusable themed parts.
- Upload flow End-to-end statement uploading flow enables users to choose a PDF on their device, upload it to the backend, and get it back in the form of parsed transactions.
- Parser kazakh bank statements (specifically Kaspi bank statements and Freedom bank statements), each with different parsing logic.
- Merchant normalization pipeline that cleans raw description of transactions, eliminates encoding artifacts, and identifies entries to canonical merchant identities.
- Hybrid transaction categorization engine that uses deterministic rule-based matching with high-confidence records and a trained machine-learned classifier with ambiguous records.
- Financial dashboards with spending trends and category breakdowns, bank-level comparisons, KPI summary cards, and month-over-month comparisons — all of which can respond to period and bank and transaction-type filters.
- KZT-native currency formatting All monetary values are shown in the form of the tenge symbol and locale suitable group separators.
- Chatbot interface powered by AI to allow users to query the chatbot with natural-language queries like, How much did I spend on food this month? and receive answers based on their processed transaction data.
- Backend database integration that enables permanent storage of users, parsed transactions, category assignments, and aggregated result of analytics.
- Automated test coverage at the unit, integration and end-to-end testing, and verifying that parsing is correct, API responses, and core flows are valid in the UI.

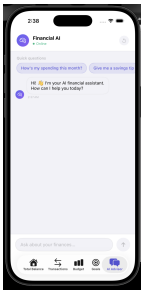

Fig. 2. App Chat View

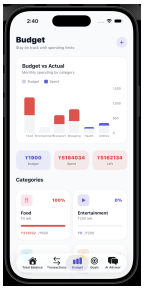

Fig. 3. App Budget View

C. Architecture and Processing Workflow

The project follows the workflow shown below:

Upload → Parse → Normalize → Categorize → Analyze → Advise

This process commences when an individual makes a statement using the mobile application. The backend checks the file, and forwards it to the relevant parsing module. The information obtained by the parser includes raw transaction data including date, amount, merchant description and

Fig. 4. App Goals View

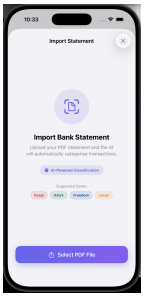

Fig. 5. Import Bank Statement View

currency. The normalization step then normalizes merchant strings, formats and record fields into a single schema. The categorization module then categorizes expenses based on a hybrid model. After the storage of structured transaction data, it is followed by the analytics layer generating spending summaries, category views, KPI cards, and budget-related products. Chatbot utilizes the processed data in responding to domain-specific financial queries.

D. Representative Use Cases

Statement Upload and Processing. A user picks a bank statement in the iOS application. This file is uploaded to the backend, processed, standardized, stored and then formatted back to the user as formatted transactions. This use case is a solution to the fundamental issue of converting the raw banking documents into the useful financial records.

Automatic Categorization. Transactions are automatically grouped as spending categories, like food, transport, utilities or entertainment, after upload. Heuristics that are based on rules are used when the deterministic signals are strong but the ambiguous cases are classified by the use of classification model. The user would then be able to view the assigned labels via the transaction view and dashboard.

Financial Querying. User queries are processed like: What did I spend this month? or What was my highest spending category? The chatbot deciphers the query, and displays the corresponding aggregates based on the processed financial information of the user.

E. Third-Party Components and Tools

The project involves a number of external tools and libraries. In the backend, FastAPI is used to create the API, and processing of incoming data and interactions are supported with Python-based tools: `pdfplumber`, `PyPDF2` and `pandas`. Experiments and classification of machine-learning is based on `scikit-learn`. SwiftUI is used to develop interfaces on the mobile side and Swift Charts can be used to visualize financial data. GitHub was used to control version control and collaborative development, and Jira, Telegram, and frequent meetings with advisors facilitated sprint planning and communication.

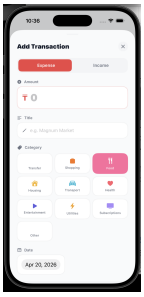

Fig. 6. Add Transactions View

F. Team Roles and Collaboration

The team divided responsibilities according to subsystem ownership while still collaborating on integration and validation.

- **Kerbez Karipbek** headed dashboards and analytics, such as spending patterns, category-based, and budget visualization.
- **Rauan Arstangaliyev** was in charge of AI chatbot orientation and the conversational query support.
- **Amirzhan Akhmetzhanov** managed transaction parsing, normalization of merchants and the data pipeline that was used to perform categorization.
- **Aidana Orazbay** and **Aslan Dossymzhan** headed native iOS development, screen impl, navigation, and upload workflows, visualization, and backend integration on the mobile side.

The development process was structured into two week sprints that were run using Jira. Every sprint was preceded by a planning session where the scope of tasks and assigning them was done, and a review at the end where functionality completed was shown against the acceptance criteria. The group was utilizing GitHub to do version control, and feature branches were reviewed using pull requests and merged into the main branch. Each merge was tested to run integration tests to identify regressions early.

A special Telegram group was used to communicate on a daily basis and the advisor meetings were held weekly to review progress. A common notebook was used to follow design decisions, API contracts and open issues. In cases where a component relied on an interface that was owned by a different member - e.g. when the iOS dashboard views relied on the analytics endpoints of a backend - the contract was documented in a common API specification before either

party started implementation, which helped reduce integration strain at milestone boundaries.

This organisational structure allowed the team to work in practical terms as a unit of solution development based on computers: individual members used professional judgement in his or her domain, cross-functional dependencies were solved in explicit contracts and frequent synergisation and the system was bound together in a functional application by disciplined milestone-based integration.

VI. Project Execution

A. Semester 1: Problem Framing and Prototype Formation

The first semester was characterized by defining the scope of the problem, studying the tools used in personal finance, and beginning to create the initial prototypes. The initial efforts were on the mobile interface, the first database schema and the earliest efforts to categorize transactions. Our aim was not to create a full system at this stage, we had to make sure that a general workflow was technically viable and could be divided into separate components which team members could work on simultaneously.

Among the lessons we got early on was that the decision of mobile framework had its repercussions that we had not expected. We have started with React Native because most of the team were familiar with web development. It performed well with simple screens, but was slow with data-intensive screens, and interacting with iOS-native apps such as file handling was too complex. the decision proved a waste of time and we had to redo it in the second semester.

B. Semester 2: Integration and Feature Maturity

Semester 2 was on converting individual prototypes to a single application. We took the mobile application and transferred it to SwiftUI, linked it to the back-end, reformatted the dashboards to incorporate KPI cards, implemented KZT currency formatting, combined data across various bank sources, connected the chatbot, and developed tests. At the conclusion of the semester, a user would be able to start with an uploaded raw bank statement and view categorized transactions on a dashboard and ask the chatbot questions about how they spend their money.

This was not as much exploratory as in the first semester. We were not trying to see whether things could work, we were making things work together. It also implied that now more time was spent debugging integration problems between modules than creating new functionality itself.

C. Key Design Decisions

Migration from React Native to SwiftUI. This was the largest architectural migration that we made. SwiftUI was faster, provided easier access to native iOS features and, more importantly, was less friction that we continued to experience with React Native. It involved rewriting the majority of the frontend, however, it eliminated a category of issues that we were continually fixing.

Hybrid Categorization Strategy. We did not rely on only one classification approach. Instead, we used a lightweight machine learning model along with rule-based logic. The rules address the obvious ones: transfer to ”Glovo” is food delivery, payment to ”2GIS Taxi” is transport. The ML model corrects ambiguous or noisy descriptions in case the rules are not sufficient. This trade-off enabled the system to be more practical in light of the scanty labeled data at our disposal.

Localized Statement-First Pipeline. Kazakh banks lack open APIs to directly access transaction data. Instead of creating something that requires integrations that we are not allowed to access, we created the system based on user-posted PDF and CSV statements. It introduces a manual process to the user, yet that is the point, it works with local banks.

D. Problems Encountered and Resolutions

The parsing part of bank statements proved to be the most difficult task in the whole project since Kaspi and Freedom Bank statements have quite different structures, including page format, columns and fields structure, and formatting styles in general. This means that a parsing algorithm which is working for one bank does not work for the other. Moreover, there are inconsistencies even within a single bank: different date formats, different truncations for business names, different field merges, etc. The development of these algorithms took considerable time from the second semester of the project.

Ambiguity in business identification. There is no standardized list of business names in local bank statements. In particular, the same merchant can be referred to as ”IP Asanov A.B.” in one case and just ”Asanov” in another one. In order to merge such transactions we used business name matching with scoring, considering additional contextual attributes, but some corner cases still caused errors.

Imbalanced Training Data. We observed that labeled examples were heavily concentrated on several classes like food and transport, while some others, e.g. healthcare, were much fewer in numbers. In order to improve the quality of categorizer we increased the size of labeled data manually and used assisted labeling techniques.

Subsystem Integration Complexity. All subsystems were initially built as separate services (mobile application, parsers, normalization pipeline, categorization, dashboards and chatbot). Integration of all these components implied proper handling of the data format and contract definition, so some cycles of debugging were necessary. Any change in any of them led to changes in another.

E. Teamwork and Project Management

For project management purposes we decided to use sprints with length from two to four weeks in the style of Agile methodology. Weekly check-in was used as a way of coordination, Telegram chat for day-to-day communication, Jira was responsible for issue tracking and GitHub for version control. Regular meetings once in two weeks with our advisor were used to maintain proper trajectory in case there appeared a

blocker or disagreement. The responsibility of the subsystems was distributed among teammates, but end-to-end workflows were discussed jointly.

VII. Evaluation

A. Evaluation Goals

The AI-Powered Personal Finance Assistant system was analyzed to determine whether it actually resolves the problem formulated in the introduction section: assisting Kazakhstani users in transforming unstructured bank statement data into structured personal finance data. Thus, it is worth noting that during testing, not only the individual functional aspects of each component were considered; the focus was on the entire computer-based system and its ability to sustain the overall user process workflow, including uploading a local bank statement file, extracting transactions, classifying expenditures, examining the dashboard, and interacting with the chatbot for financial inquiries.

The evaluation was guided by the following questions:

1) Can the system extract structured transaction records from supported Kazakhstan bank statements?
2) Can the categorization pipeline assign meaningful categories to transaction descriptions?
3) Can the dashboard layer compute financial summaries quickly enough for interactive use?
4) Can the full statement-processing pipeline transform PDF statements into normalized and classified transaction outputs?
5) What limitations remain before the system can be evaluated as a production-ready personal finance assistant?

These questions have been selected based on the value of the project, which lies in the capability of the system to convert data from local bank statements into something that can be analyzed and searched for by the users. Thus, the evaluation will consider factors such as parsing efficiency, categorization, computation in the dashboards, and overall process time.

B. Evaluation Methodology

The assessment involved both technical validation and work-flow examination. First, the parser component was evaluated for accuracy using the PDF bank statements and their corresponding CSV fixtures present in the project repository. The collection consisted of six PDF files; four were sourced from Kaspi Bank while the remaining two were from Freedom Bank. Each PDF was validated against its corresponding CSV fixture based on row counts and transaction field completeness.

Secondly, the categorization algorithm was assessed using the labeled transactions dataset provided in the project. After filtering out instances where descriptions or categories were missing, we retained a total of 7,199 samples for analysis. This dataset was then stratified into an 80% training set and a 20% hold-out set using a predetermined random seed value. Consequently, we ended up with 5,759 samples in the training set and 1,440 samples in the testing set.

Lastly, we evaluated the performance of the dashboard by measuring the amount of time required for computing the summary statistics from the processed transactions. We used 1,903 transactions as inputs for the benchmarkProcess. These computations involved the global sums, sum by categories, bank-level summaries, and group-type summaries. The goal here is to assess the computation module behind the dashboard and not the entire process including rendering on the iOS application.

Lastly, we evaluated the end-to-end process by executing the PDF to output pipeline using the six available bank statement PDF files. The steps of the PDF to output pipeline include detecting the bank, parsing the document, normalizing the data, and classifying the transactions. The OKED network fetching process was deliberately excluded from this benchmark due to its dependency on network requests and hence its inherent instability.

C. Evaluation Dataset

The evaluation used three main datasets and file groups:

- There are six types of bank statement in PDF format: four from Kaspi Bank and two from Freedom Bank.
- The CSV fixture data has been parsed, resulting in a total of 8,063 rows of transactions across the six statements.
- There is also a labeled dataset for categorization containing 7,202 rows before filtering, out of which 7,199 are usable labeled examples.

The classes in the categorization dataset included transfer, purchase, transport, supermarket, restaurant/cafeteria, subscription, and marketplace. The classification dataset was imbalanced, with transfer and purchase as the two most frequent classes. As such, it was deemed appropriate to use macro-F1 score in conjunction with accuracy since the macro-F1 score is more representative.

D. Quantitative Results

Table I summarizes the main quantitative evaluation results.

TABLE I
Evaluation Metrics

Metric	Result
Parsing quality	100% core-field completeness; 8,063 parsed transactions; 6 PDFs
Categorization quality	95.42% accuracy; 95.94% macro-F1; 1,440 test samples
Dashboard performance	83 ms with CSV loading; 55 ms warm calculation
End-to-end processing	13.5 s per statement; 6 PDFs processed in 81.0 s; OKED skipped
User feedback	No structured questionnaire collected

E. Parsing Evaluation

The parser was tested using six PDF bank statements supported by the project: four of Kaspi Bank and two of Freedom Bank. For each of the six files, the number of transactions generated by the parser coincided precisely with the numbers provided by the prepared CSV fixtures. Altogether, the output of the parser amounted to 8,063 transaction entries.

Regarding the completeness check, the following core fields were considered: date, amount, original amount as text, currency, operation type, transaction description, name of the bank, and the name of the source file. All 8,063 parsed transactions included non-null values for all core fields. As a result, the completeness rate of core fields became 100

This proves that the parser is reliable when processing the current formats of bank statements. However, it should not be interpreted as an assertion about the general parsing accuracy of any Kazakhstan bank statement. In fact, the validation procedure applies only to the formats of Kaspi Bank and Freedom Bank available in the project's dataset.

F. Categorization Evaluation

The model was assessed by applying the model to the labeled data provided in the repository. Rows containing missing values in both description columns and category columns were discarded, leaving a dataset containing 7,199 samples. The dataset was then randomly partitioned into training samples and test samples, comprising 5,759 and 1,440 samples, respectively, through stratified sampling.

The results on the hold-out test set are shown above. Specifically, the model achieves an overall accuracy of 95.42% and macro F1 of 95.94%. Given that the distribution of the target variable is highly unbalanced, the significance of macro F1 becomes clear – the distribution of transactions in different categories is heavily skewed toward larger categories such as transfers and purchases, and macro F1 captures the performance of the classifier in all categories.

Regarding specific performance, the classifier performed reasonably well for transport, supermarket, subscription, and marketplace categories. It also achieved low relative precision and recall for restaurant and café categories, which can be expected due to the noisy nature of the merchant descriptions used in the classification process and some descriptions being identical to general purchases.

On balance, the findings from the classification support the decision to choose the lightweight machine learning classifier for categorizing the transactions. This classifier performs well within the acceptable scope of use for the student-designed mobile financial assistant.

G. Dashboard Performance Evaluation

The dashboard computations were benchmarked using a preprocessed data file containing 1,903 transactions. The process times for loading the transaction data and computing the main dashboard statistics were considered, which include totals, categorization, bank aggregation, and trend summaries.

In total, including the CSV loading time, the average duration of the computations was 83 milliseconds. Excluding the CSV loading time from the computation, the average duration of the dashboard statistics was 55 milliseconds. Both times indicate that the data analysis layer does not impose high computational costs, and thus can be used in the prototype stage of the application development process. Given that the time required remains less than one second, it is likely that no performance issues will arise during calculations on this scale.

However, it should be noted that the benchmarking procedure only accounts for the time spent on calculations in the data layer of the analytics pipeline. The benchmarking process did not include the visual rendering time of the dashboard inside the iOS application.

H. End-to-End Pipeline Evaluation

Testing of the entire process workflow involved executing the bank statement processing for all six sample PDF documents. In this regard, we used the full process chain to detect the bank, parse and normalize the data, and classify transactions contained therein. In order not to affect the process performance with any additional network-dependent operations, we excluded fetching of OKED from the execution plan.

This resulted in 2,103 successfully parsed and classified transactions from the six bank statements, with the process completing in around 81 seconds or 13.5 seconds on average for one document. Thus, it can be said that the project has already gone past the initial stages and reached the state of delivering some useful output.

As a prototype, the performance is acceptable due to the nature of statements uploading being an occasional task rather than a real-time process. On the other hand, the obtained outcomes show what needs to be optimized to make further development more efficient, including speed improvements for parsing, batch execution, as well as reducing duplicate operations.

I. User-Centered Validation

Since there was no formal user survey conducted, there is no user feedback rating number. The importance of this step is that during the Senior Project Evaluation, it should be evident that the system will be comprehended by its possible users.

At the moment, the created workflow includes such typical tasks as uploading statements, viewing transaction data, summarizing transactions based on categories, viewing dashboard values, and asking finance-related questions. While internal analysis showed that these typical user actions can be successfully done using the system, an external test would give a better understanding of the usability and usefulness of the application.

One of the effective approaches to the further testing might be a task-based research where users perform actions like uploading statements, figuring out the most expensive category for themselves, analyzing their spending per month, and asking a chatbot about finance-related questions. Users then give a five-point rating for each parameter.

J. Discussion

In conclusion, our prototype approach solves the original problem through a sequence of stages: the parsing module parses and extracts information from supported local bank statements; the classification task produces highly accurate

category assignment on the out-of-sample evaluation; and the computation stage within the dashboard generates summaries rapidly. Overall, an end-to-end test case has demonstrated how real PDF statements could be converted to normalized, categorized data in less than 30 seconds.

Our project could be considered as a computer-based solution for the following reasons: combining multiple tasks such as document processing, data normalization, machine learning classification, computing, and financial interpretation of information. Thus, the prototype developed during the course provides much more than just data storage: it transforms the raw input data into valuable output that allows users to manage their finances better and be aware of their spending habits.

However, there are definite limitations to our prototype, including the following: support for Kaspi and Freedom Bank statement types only; classification using a dataset with imbalance issues; computation time benchmark without mobile rendering time; the absence of OKED network fetching in the end-to-end test case; lack of questionnaire data from external users.

Despite its limitations, the evaluation confirms the system's functionality and compliance with the project objectives. This indicates the technical feasibility of developing an application for localized personal finance assistants that first analyzes financial statements before making recommendations based on the user's specific circumstances in Kazakhstan.

VIII. Conclusion and Future Work

This project presented the design and implementation of a localized, AI-supported personal finance assistant for Kazakhstan users. Across two semesters, the team developed the system from an early prototype into an integrated mobile and backend solution that supports statement ingestion, transaction normalization, automatic categorization, dashboards, budget visualization, and conversational financial querying.

The main contribution of the project is not only the individual features, but the way they were combined into one computing-based workflow. The application demonstrates that a statement-first architecture can serve as a practical alternative when direct bank integrations are limited. It also shows that combining deterministic processing, lightweight machine learning, and human-readable financial views is an effective strategy for building a transparent finance assistant.

Future work can extend the project in several directions. First, the system can support more banks and more statement formats, including broader localization support. Second, the chatbot can become more robust through stronger prompt design, safer answer generation, and tighter grounding in the structured transaction database. Third, evaluation can be expanded through a larger user study and more comprehensive performance measurement. Finally, the project can be strengthened for real-world deployment through more advanced authentication, stronger production-grade security, and long-term data-governance mechanisms.

References

[1] S. García-Méndez, M. Fernández-Gavilanes, J. Juncal-Martínez, F. J. González-Castaño, and O. Barba-Seara, "Identifying banking transaction descriptions via support vector machine short-text classification based on a specialized labelled corpus," *IEEE Access*, vol. 8, pp. 61 642–61 655, 2020.

[2] D. Kotios, G. Makridis, G. Fatouros, and D. Kyriazis, "Deep learning enhancing banking services: a hybrid transaction classification and cash flow prediction approach," *Journal of Big Data*, vol. 9, p. 100, 2022.

[3] H. Wasserbacher and M. Spindler, "Machine learning for financial forecasting, planning and analysis: recent developments and pitfalls," *Digital Finance*, vol. 4, pp. 63–88, 2022.

[4] K. Kapparov, "Financial inclusion and financial literacy in kazakhstan," Asian Development Bank Institute, Tech. Rep. 876, 2018.

[5] R. A. AlSuwaidi and C. Mertzanis, "Financial literacy and fintech market growth around the world," *International Review of Financial Analysis*, vol. 95, p. 103481, 2024.

[6] G. B. Ferilli, E. Palmieri, S. Miani, and V. Stefanelli, "The impact of fintech innovation on digital financial literacy in europe: Insights from the banking industry," *Research in International Business and Finance*, vol. 69, p. 102218, 2024.

[7] O. Haggag *et al.*, "An analysis of privacy regulations and user concerns for financial applications," *Information and Software Technology*, 2025, online ahead of print.

[8] H. Xia, C. Xu, J. Z. Zhang, S. M. Jasimuddin, and X. Li, "The influence of readability of financial app privacy policy on enterprise performance," *Journal of Database Management*, vol. 35, no. 1, 2024.

[9] P. Weber, K. V. Carl, and O. Hinz, "Applications of explainable artificial intelligence in finance—a systematic review of finance, information systems, and computer science literature," *Management Review Quarterly*, vol. 74, no. 2, pp. 867–907, 2024.

[10] N. Balasubramaniam, M. Kauppinen, A. Rannisto, K. Hiekkanen, and S. Kujala, "Transparency and explainability of ai systems: From ethical guidelines to requirements," *Information and Software Technology*, vol. 159, p. 107197, 2023.

[11] P. Bitrián, I. Buil, and S. Catalán, "Making finance fun: the gamification of personal financial management apps," *International Journal of Bank Marketing*, vol. 39, no. 7, pp. 1310–1332, 2021.

[12] D. French, D. McKillop, and E. Stewart, "The effectiveness of smartphone apps in improving financial capability," *The European Journal of Finance*, vol. 26, no. 4-5, pp. 302–318, 2020.

[13] A. Lusardi and O. S. Mitchell, "The economic importance of financial literacy: Theory and evidence," *Journal of Economic Literature*, vol. 52, no. 1, pp. 5–44, 2014.

[14] D. Karlan, M. McConnell, S. Mullainathan, and J. Zinman, "Getting to the top of mind: How reminders increase saving," *Management Science*, vol. 62, no. 12, pp. 3393–3411, 2016.

[15] R. H. Thaler and C. R. Sunstein, *Nudge: Improving Decisions About Health, Wealth, and Happiness*. Yale University Press, 2008.

[16] World Bank Group, "Kazakhstan: Financial sector assessment program—technical note on fintech," World Bank Group, Tech. Rep., 2020.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] V. Singh and P. Best, "Machine learning for automated transaction categorization," *Journal of Financial Technology*, vol. 7, no. 2, pp. 145–162, 2019.

[19] Y. Wang, Q. Chen, J. Zhang, L. Cao, J. Yang, and C. Zhang, "A cost-sensitive ensemble method for class-imbalanced datasets," *Abstract and Applied Analysis*, vol. 2017, p. 5402832, 2017.

[20] R. Smith, "An overview of the tesseract ocr engine," in *Ninth International Conference on Document Analysis and Recognition*, vol. 2. IEEE, 2007, pp. 629–633.

[21] S. Smith and J. Cordeiro, "Document analysis challenges in financial services," in *Proceedings of the 2009 10th International Conference on Document Analysis and Recognition*. IEEE, 2009, pp. 1243–1247.

[22] F. Li and H. V. Jagadish, "Constructing an interactive natural language interface for relational databases," *Proceedings of the VLDB Endowment*, vol. 8, no. 1, pp. 73–84, 2014.