

<https://doi.org/10.31891/csit-2026-3-5>

Serhii SAVCHENKO

PhD, Doctorate of Computer Engineering
& Information Systems Department,
Khmelnytskyi National University
<https://orcid.org/0000-0001-9706-5334>
e-mail: savchenkoso@khnmu.edu.ua

Received: 15/08/2026
Accepted: 23/09/2026
Published: 30/09/2026

© Copyright
2026 by the author(s)



This is an Open Access article distributed
under the terms of the [Creative Commons
CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/)

UDC 004.89: 336.76

DESIGN AND IMPLEMENTATION OF A RAG-BASED AI AGENT FOR ROBO-ADVISORS

This study focuses on the design and implementation of an agentic system based on Retrieval-Augmented Generation (RAG) architecture for automated financial advisory services (Robo-Advisors, RA). The relevance of this study lies in the knowledge limitations of large language models (LLMs) in domain-specific areas, such as the financial sector, where factual errors can lead to direct financial losses and regulatory risks. Based on a literature review, the class of tasks for a modern Robo-Advisor platform suitable for RAG-based solutions has been identified. The purpose of this research is to develop a RAG agent architecture for a Robo-Advisor service and verify its functionality on a working prototype. Functional requirements for an RA module were established to track the behaviour of financial assets within a specific investment portfolio. An analysis was conducted on the tools used to implement typical interaction scenarios with a RAG system (data acquisition, text preprocessing, vectorisation, data normalisation, LLM interaction, etc.). A software architecture was designed in which agent functions (obtaining stock prices, analysing financial news, automated reports, user communication) are reflected in separate software modules. The system is divided into an asynchronous indexing circuit and a synchronous user interaction circuit. The agent's memory is implemented as hybrid storage, where a relational database serves as the source of truth, and the vector index is a fully recoverable derivative artifact. A method for re-ranking news search results using a normalised time-of-publication coefficient is proposed. User interaction is implemented as a chatbot for a messenger. Pilot operation confirmed that the system meets the established requirements; identified shortcomings and limitations of the agent are described, along with methods for their resolution. The results of this study can be used to integrate RAG agents into full-scale Robo-Advisor services.

Keywords: Agentic AI, Large Language Model, Retrieval-Augmented Generation, Robo-Advisor, Investment Portfolio Optimisation, FinTech.

Introduction

A Robo-Advisor is a software platform that automates the investment management process. Contemporary Robo-Advisors are designed for direct use by investors, bypassing the need for financial experts. A systematic review [1] indicates that the fundamental functions of such systems include initial portfolio construction, asset allocation, and daily portfolio management (asset rebalancing) based on machine learning algorithms. During the initial portfolio formation, financial instrument recommendations are generated by taking into account the metrics of the investor's individual profile, such as risk tolerance, income level, investment expectations, and the planning horizon.

Over the course of the past several years, large language models have transitioned from being purely academic objects of inquiry into practical instruments that are being actively integrated into commercial and governmental products. Private enterprises and public institutions have commenced the vigorous implementation of chatbots and AI agents within their offerings. For instance, banking institutions are integrating intelligent consultants for customer service, healthcare facilities are deploying AI assistants for patient support, insurance providers are developing automated risk assessment systems, and retailers are offering personalised product and service recommendation systems.

The experience of Ukraine is exemplary regarding the implementation of AI in the public sector. In 2025, the "Diia" application introduced Diia.AI – the world's first national AI assistant, which was recognised as the winner of the European

Commission's SEMIC Best Cases Award 2025 in the "Government-to-Citizens" category [2]. The system demonstrates a CSAT satisfaction rate of over 80%, is capable of reducing the time required to resolve citizen enquiries to 3 minutes, and is projected to save citizens over 1 million hours per year [2]. Diia.AI is built on a commercial LLM from Google (Gemini Flash) combined with proprietary security filters and integration with state registers [3]. Currently, similar initiatives are being implemented in other countries. The US government actively uses chatbots in state agencies to interact with citizens [4]. The banking industry is also adopting AI for personalised financial services [5].

The fundamental technological challenge remains that general-purpose LLM models cannot be directly applied to specific tasks within particular business sectors. Base models lack access to up-to-date internal company data, are unfamiliar with specific industry regulatory requirements (such as International Financial Reporting Standards (IFRS) or National Bank of Ukraine regulations), and are unable to perform actions in external systems (CRM, ERP, banking infrastructure, etc.). For this reason, effective AI integration into industrial products (primarily regarding chatbots and agents built on LLMs) requires the implementation of interaction between the LLM and external information sources and services.

Related works

The study [6] identifies four stages in the development of automated financial consulting: traditional financial advisors (1950s), discount brokers (1970s), online trading (1990s), and Robo-Advisors themselves. The first RA platforms appeared in 2008 and spread rapidly in the USA and other countries starting from 2011. The authors of the study [6] describe four stages of RA platform development, from simple questionnaire-based recommendations to self-learning systems with automatic investment portfolio rebalancing (Figure 1).

An analysis of sources [6, 1] allows for a summary of the typical functionality of a modern RA system: (1) client profiling and risk tolerance assessment; (2) forecasting the returns of financial instruments; (3) portfolio structure optimisation; (4) monitoring and rebalancing; (5) the client interaction interface. This last component is increasingly implemented in a conversational format. The study [6] describes the architecture of a conversational RA that combines an asset allocation subsystem (Black-Litterman and Markowitz models) with a dialogue subsystem based on AIML (Artificial Intelligence Markup Language). RA platforms are also considered a tool for the democratisation of wealth management. In study [7], the examples of the Wealthfront and Betterment systems demonstrate that AI has simplified the creation of well-diversified portfolios for the general user.

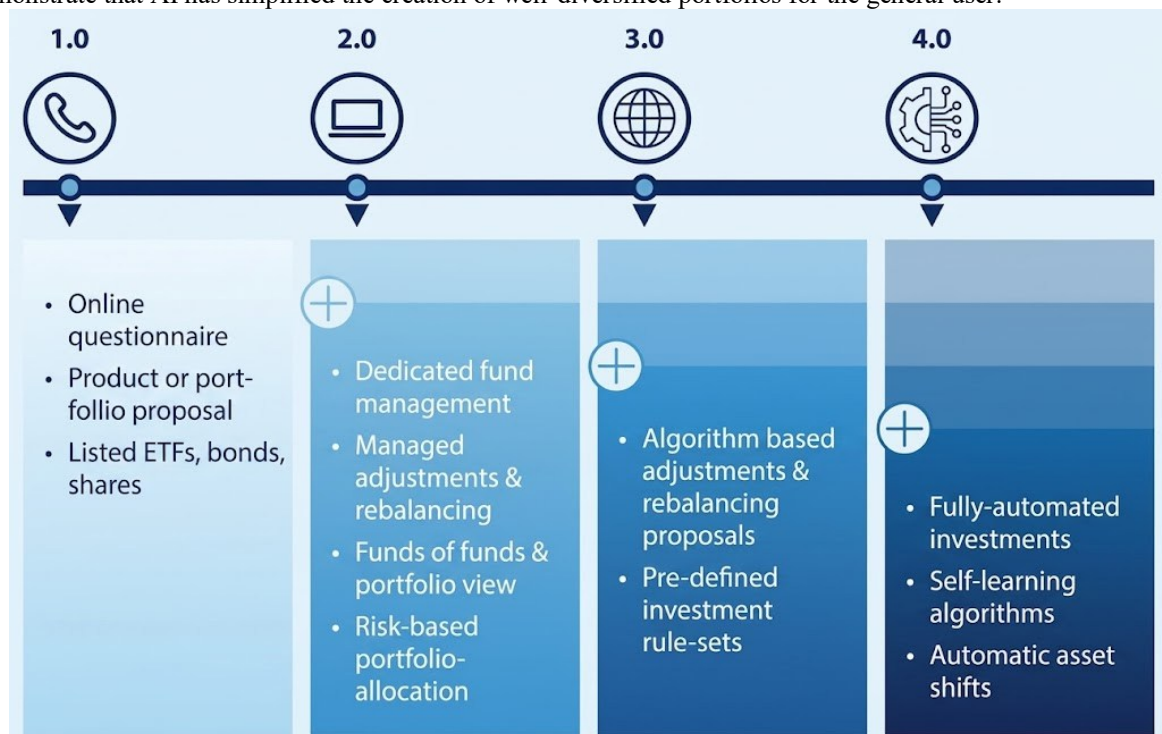


Fig. 1. Evolution of Robo-Advisor systems

Generally, AI applications in RA systems span the entire lifecycle of investment decision-making. In study [7], a comparative analysis of deep learning models, such as LSTM (Long Short-Term Memory) and RNN (Recurrent Neural Network), was conducted against classic time series processing methods (ARIMA). It was shown that neural network models are significantly better at reproducing market nonlinearity and volatility. NLP (Natural Language Processing) methods are used to analyse the sentiment of news and the reports of public companies listed on stock exchanges.

In study [8], the portfolio management task is formalised as a Markov decision process, and agents are trained using DQN (Deep Q-Network) and PPO (Proximal Policy Optimization) algorithms. Experimental comparison demonstrated the superiority of RL (Reinforcement Learning) strategies over traditional ones: the cumulative return of PPO was 132.4% compared to 98.5% for Modern Portfolio Theory (MPT), while the Sharpe ratio was 1.42 versus 1.12 [8]. The authors emphasise that the key advantage of RL is its capacity for sequential decision-making and adaptation to changing market regimes, which static optimisation models lack. At the same time, the authors point out the tendency of RL models towards overfitting and their high computational cost. In [1], it is highlighted that the lack of transparency in algorithms creates direct regulatory risks. Study [7] justifies the potential of hybrid models that combine machine analysis with expert judgement, along with the development of XAI (Explainable AI) methods. Table 1 summarises the areas of application for machine learning and AI methods in modern Robo-Advisor systems.

Table 1

Areas of AI Application in Robo-Advisor Systems

Application area	Typical methods and models	Sources
Forecasting returns and trends	LSTM, RNN, ensemble methods	[7]
Portfolio optimisation and rebalancing	DQN, PPO, Black–Litterman, MPT	[6, 8]
Client risk profiling	Clustering, recommender systems	[6, 1]
Unstructured data analysis	NLP, BERT, sentiment analysis	[7]
Fraud and anomaly detection	Unsupervised learning, stream processing	[7]
Client interaction	Dialogue systems, LLM assistants	[6, 7]

Key aspects of rag architecture

Retrieval-Augmented Generation (RAG) is an architectural approach that combines large language models with external knowledge sources that remain outside of its parameters and can be updated independently of the training process. Figure 2 presents the main components of a RAG system. A canonical RAG system contains three components [9]:

- retrieval module, which accesses a vector database or API;
- augmentation module, which selects and condenses relevant context;
- generative module (generation), which generates the response.

The operating principle of a RAG system is based on a two-phase pipeline. During the preparatory (offline) phase, documents from selected sources are segmented into chunks. For each chunk, an embedding model calculates a dense vector representation within a semantic space. These vectors, along with metadata, are stored in a vector database for subsequent nearest-neighbour searches. During the query processing (online) phase, the user's query is vectorised by the same model, after which the retrieval module selects fragments closest to the query based on cosine similarity. The augmentation module compresses and organises the selected context according to the constraints of the context window. Finally, the generative module synthesises a response, relying simultaneously on the provided context and the model's own parametric knowledge, and optionally accompanies it with references to the sources used.

RAG systems allow for the overcoming of two fundamental problems of LLMs: hallucinations and the limitation of knowledge by the training date [9, 10, 11]. For the financial sector, particularly for Robo-Advisors, both of these issues are critical. Factual errors during investment decision-making can lead to direct financial losses, reputational damage, and violations of regulatory requirements.

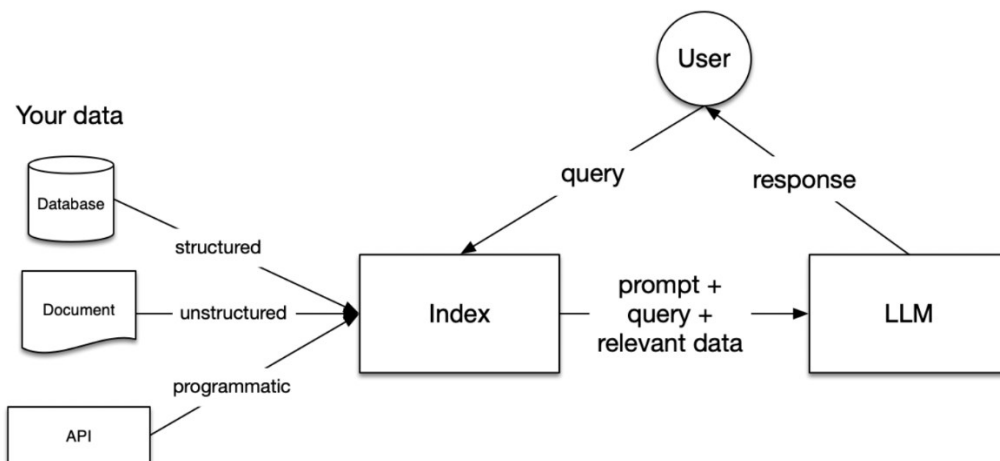


Fig 2. High-level architecture of RAG systems [12]

Empirical research confirms the effectiveness of using RAG in the financial sector. Study [13] describes an end-to-end real-time market analysis system that integrates SEC reports (Forms 10-K, 10-Q, 8-K), investor call transcripts, financial news, and social media data. Authors of another study [14] showed that optimised strategies for chunking financial documents reduce factual errors by 72.4%, while 93.7% of responses correctly rely on the latest available data. For Robo-Advisor systems, RAG is applied to the following tasks [13, 14, 15, 16]:

- data relevance (investment recommendations must be based on current reports and news, rather than on the static weights of the language model);
- explainability of recommendations (the ability to verify the primary sources underlying a particular investment recommendation increases client trust and reduces regulatory risks);
- personalisation (rewriting queries based on the client's risk profile, personalised search, and generation aligned with preferences);
- ensuring the security of personal and corporate data (RAG modules operate and store data within the company's internal infrastructure; only the minimum necessary dataset, which can be further anonymised, is transmitted to third parties or AI providers).

At the same time, certain engineering challenges still remain unresolved. The study [10] mentions context window limitations, the need for incremental vector index updates, and the lack of methodologies for evaluating RAG system efficiency. The article [13] highlights the issue of conflicting sources, while the authors of [14] note the complexity of integration with legacy banking systems. The reviewed literature does not present a comprehensive architecture for a RAG agent specifically for Robo-Advisor services.

Designing the architecture of a rag-based agentic system

In the experimental part of this research, a Robo-Advisor microservice was designed and developed to autonomously gather market information and provide the user with well-founded, referenced responses regarding their investment portfolio. The functional requirements include:

- the autonomous periodic collection of financial instrument quotes and related economic and political news;
- the processing, storage, and indexing of the collected data;
- the processing of arbitrary user queries in a dialogue mode, with the mandatory citation of the sources used;
- the periodic delivery of a concise market sentiment overview based on data from the financial instruments included in a specific user's investment portfolio.

Non-functional requirements include the use of Java and Python programming languages, deployment management via Docker Compose, and cost optimisation for the use of third-party AI providers (LLM requests and embedding generation). The system is deployed as four containers managed by Docker Compose: the main application, a relational database, a vector store, and a Python service for market data. Interaction with the messenger API is based on a long polling scheme, so the system does not require any incoming network ports, which significantly simplifies its security. Figure 3 illustrates the general architecture of the RAG-Based Agentic System.

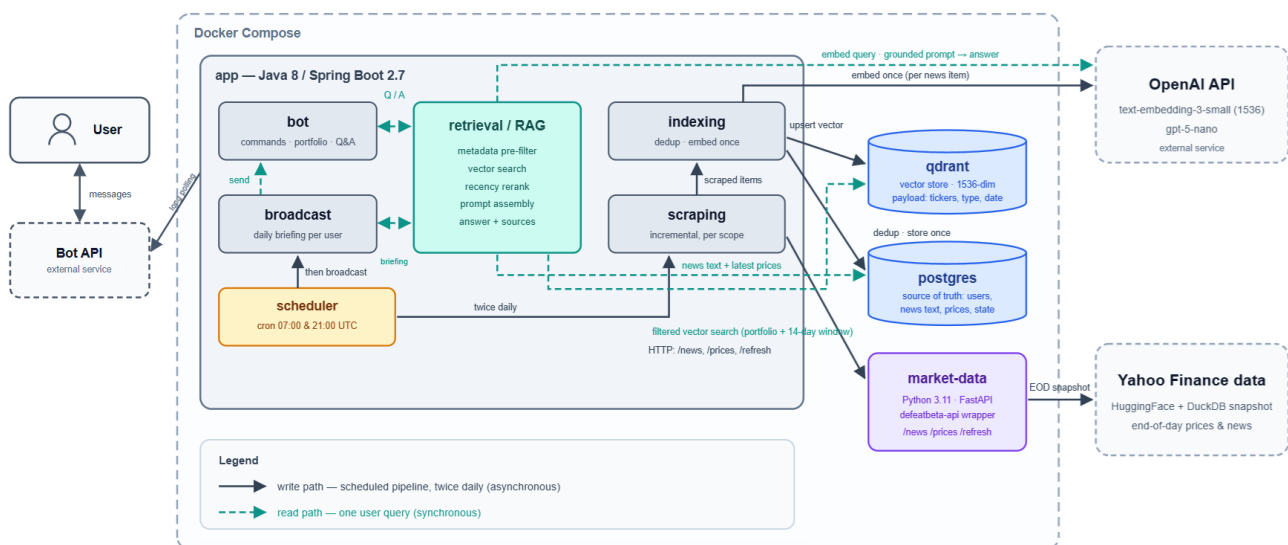


Fig. 3. Architecture of the agentic system

The system is designed as an autonomous agent that collects and processes information automatically. Overall, five basic agent tasks can be identified, each of which is implemented in a separate software module:

- perception – a data collection module that incrementally polls external sources for quotes and news;

- memory – an indexing module that forms a two-level knowledge repository;
- reasoning – a search and generation module that constructs a grounded context and formulates a response;
- action – a Telegram bot and distribution service that deliver the results to the user;
- planning – a scheduler that enables the agent to operate according to a schedule, rather than only at the time of a user request.

Personalisation in the system is implemented at the search level. The user's portfolio is not merely a text mention in the prompt, but a filter predicate for candidates in the vector storage. Mentioning tickers in the prompt does not guarantee that the model will not rely on irrelevant documents, whereas a filter at the index level makes the appearance of such documents in the context impossible. The system components are logically divided into two independent data processing circuits (Figure 4), which can be considered close to the CQRS (Command Query Responsibility Segregation) architectural pattern.

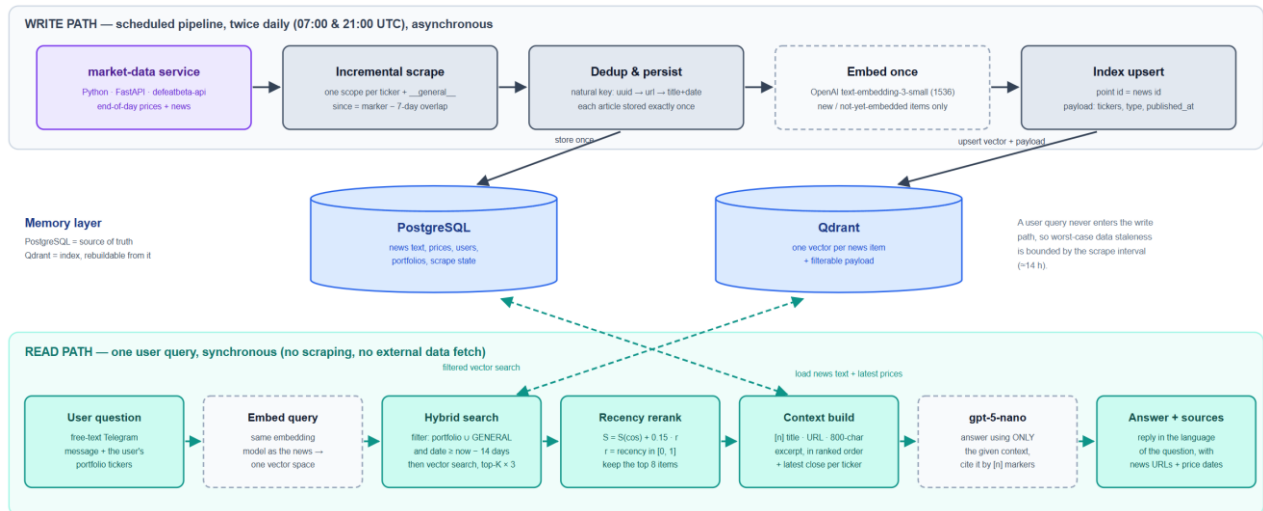


Fig. 4. Data processing loops (indexing and search retrieval)

The agent's memory is designed as hybrid storage with a clear separation of responsibilities. A relational database serves as the single source of truth. It stores users and their portfolios, full texts of news articles, price time series, and the operational status of the data scraper. The vector storage acts as a derivative index, containing only the vector for each news item and the minimum set of metadata required for filtering (news ID, list of related tickers, news type, and publication timestamp). Figure 5 illustrates the data storage scheme within the Docker container system.

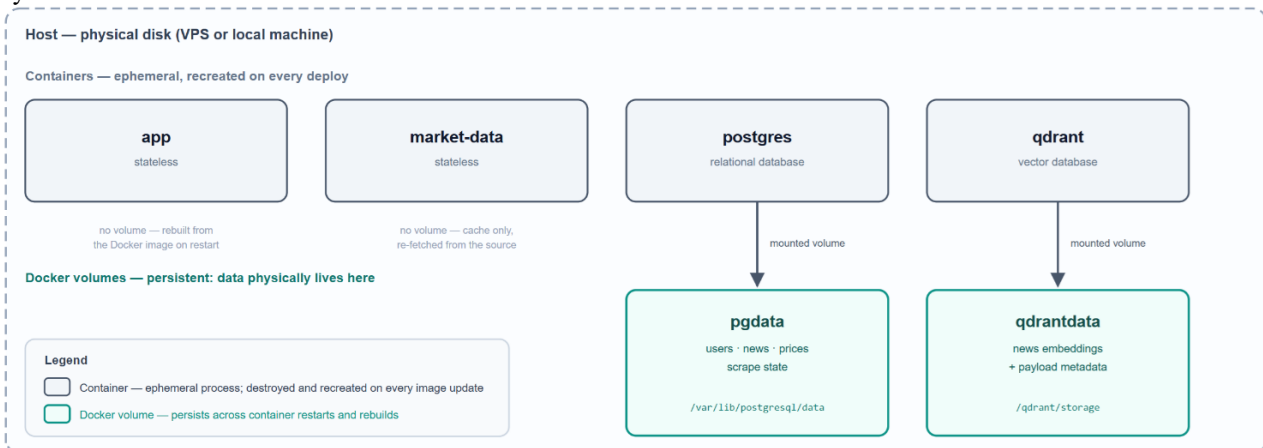


Fig. 5. Application container and data volume layout

Each news item is stored exactly once, even if it refers to multiple tickers (a many-to-many relationship). General market news has no such connections at all. Consequently, the vector representation is also calculated exactly once per news item, rather than for each connection. This directly fulfils the requirement to reduce the number of requests to the paid API.

The reading pipeline is designed as a three-stage retrieval pipeline. In the first stage, metadata filtering is performed using two components: a strict freshness condition (the publication date must not be older than a set window) and a disjunctive membership condition (the news is either market-wide or linked to at least one portfolio ticker). In the second stage, within the allowed subset, the cosine similarity between the query vector and the news

vectors is calculated. Here we request three times as many candidates as needed for the context. In the third stage, re-ranking is performed with a bias towards freshness, after which a fixed number of the best documents are selected. The need for the third stage is due to the nature of the metric itself: cosine similarity is indifferent to time, whereas in the financial sector, two semantically nearly identical news items of different ages have different values for the user. Strict ordering by date is an unacceptable opposite extreme, as it completely undermines semantic relevance. Therefore, an additive combination of both factors is applied, with a small weight given to the time component. Fresher documents win in near-tie situations, but they do not replace significantly more relevant, but older, documents.

The generation stage is designed to ensure result traceability. The selected documents are provided to the model as a numbered context block, supplemented with the latest known closing prices for each portfolio instrument. The system prompt requires the model to respond exclusively based on the provided context, cite the news items by their numbers, and clearly state if there is a lack of data. Alongside the text response, the agent returns a structured list of sources that is displayed to the user.

System implementation and software components

The core modules of the system are implemented in Java using the Spring Boot framework. PostgreSQL 14 DBMS is used as the relational database. The database schema is managed by Flyway, and Hibernate operates in validate mode, which prevents uncontrolled schema changes by the ORM and makes every structural modification an explicit, versioned artefact. Qdrant was chosen as the vector store, whose key advantage is the support for metadata (payload) filtering directly during the vector search. This enables hybrid search to be performed without an additional post-processing layer.

The OpenAI API was used as the AI provider. The text-embedding-3-small model (1,536 dimensions) is used for text vectorisation, while the gpt-5-nano model is used to generate responses. The market data retrieval service is implemented in Python 3.11, using FastAPI, the Uvicorn server, and the defeatbeta-api library. This library provides pre-processed data from Yahoo Finance (cached using HuggingFace and DuckDB). Additionally, the ftfy library is employed to correct corrupted text encoding (mojibake) in the primary sources.

The deployment is performed using Docker and Docker Compose. The main application is built with a multi-stage Dockerfile, and all credentials are passed exclusively through environment variables. Table 2 presents the main technologies used to develop the agent.

Table 2

The system technology stack

System components	Technology
Application service	Java, Spring Boot, Maven
Relational database	PostgreSQL 14, Flyway for database structure migrations
Vector database	Qdrant
Embeddings	OpenAI text-embedding-3-small, 1536 dimensions
Generative model	OpenAI gpt-5-nano
Market data source	Python, FastAPI, defeatbeta-api
User interface	Telegram bots API
Testing	JUnit 5, Mockito, Testcontainers
Deployment	Docker, Docker Compose (4 containers, 2 volumes)

The vector search filter is structured into two groups of conditions (Listing 1). The *must* group contains a strict condition for news recency. The publication timestamp must not be less than the threshold of *now - rag.recency-window-days* (14 days by default). Consequently, outdated news cannot be included in the response. The *should* group implements a relevance disjunction: the news is of a general market type, *OR* the list of its tickers intersects with the user's portfolio. As a result, the response always combines both personal and general market contexts, even if no news regarding a specific asset was published during the reporting period.

```
List<Map<String, Object>> must = new ArrayList<>();
must.add(condition("published_at", singletonMap("range", singletonMap("gte", cutoffEpoch))));

List<Map<String, Object>> should = new ArrayList<>();
should.add(condition("type", singletonMap("match",
    singletonMap("value", "GENERAL"))));
if (tickers != null && !tickers.isEmpty()) {
    should.add(condition("tickers", singletonMap("match", singletonMap("any", tickers))));
}

Map<String, Object> filter = new HashMap<>();
filter.put("must", must);
filter.put("should", should);
```

Listing 1. Building a hybrid filter for searching in a vector store

To retrieve the most relevant news during the processing of a user query, an adaptive model (Listing 2) is employed, taking into account the publication date and the vector similarity measure:

$$S = S_{cos} + w \cdot r, \text{ where } r = \frac{(t_{pub} - t_{cutoff})}{t_{now} - t_{cutoff}} \in [0, 1], w = 0.15. \quad (1)$$

Recency normalisation is performed based on the width of the search window: the oldest permissible document receives $r = 0$, and the most recent receives $r = 1$. This makes the adjustment invariant to the window width, meaning that changing the *rag.recency-window-days* parameter does not require the weight to be readjusted. The value of $w = 0.15$ was selected as a fraction of the cosine similarity range. It is sufficient to guarantee that among documents with similar S_{cos} values, the newer one is preferred, but it is too small to replace a significantly more relevant, although older, news item. To provide sufficient data for re-ranking, three times more candidates are requested from the vector store than are needed for the context. After sorting, the list is reduced to the *top-K* items, and the news articles are loaded from the relational database while preserving their rank order.

```
public List<News> retrieve(List<String> tickerSymbols, String question) {
    int topK = ragProperties.getRetrieval().getTopK();
    Instant now = Instant.now();
    long cutoff = now.minus(ragProperties.getRecencyWindowDays(),
        ChronoUnit.DAYS).getEpochSecond();

    float[] queryVector = openAiClient.embed(question);
    List<SearchHit> hits = vectorStore.search(queryVector, tickerSymbols, cutoff, topK * 3);
    if (hits.isEmpty()) {
        return new ArrayList<>();
    }
    double span = Math.max(1.0, now.getEpochSecond() - cutoff);
    hits.sort(Comparator.comparingDouble(h -> -combinedScore(h, cutoff, span)));
}

private double combinedScore(SearchHit hit, long cutoff, double span) {
    double recency = Math.max(0.0, Math.min(1.0, (hit.getPublishedAt() - cutoff) / span));
    return hit.getScore() + RECENCY_WEIGHT * recency; // RECENCY_WEIGHT = 0.15
}
```

Listing 2. Document reranking considering the publication date

The system is covered by two levels of testing. Unit tests use Mockito and completely isolate external dependencies. The integration test loads the full Spring context with a disposable PostgreSQL container (using Testcontainers) and verifies the consistency of the Flyway migration. Additionally, a set of administrative REST endpoints has been implemented. These allow for the unscheduled launch of the data collector, the viewing of status labels, the complete rebuilding of the vector index, and the execution of the full RAG response path for a user query.

Conclusions

The operation of the developed agent was practically tested over a four-week period. During the testing, several issues related to updating the data on financial instrument rates were identified. After rectifying all the discovered bugs, the agent fully meets the established functional requirements. The agent automatically updates the knowledge base on schedule, replies to arbitrary user enquiries in the language of the request, accompanies each answer with a list of used sources, and takes into account the composition of the specific investment portfolio.

The conducted literature review shows that a typical Robo-Advisor platform covers client profiling, return forecasting, portfolio optimisation and rebalancing, market monitoring, and client interaction. While the first three task groups are effectively solved using classical machine learning methods, the final two (handling unstructured data and user interaction) represent a task category where a RAG agent holds a significant advantage. At the same time, the reviewed studies focus either on financial analysis or portfolio optimisation, without proposing a complete agent architecture for a Robo-Advisor.

This study presents the architecture of a RAG agent, which can function either as a standalone product or as a module of a Robo-Advisor service. The developed system proved its relevance during practical testing. Trial operation confirmed compliance with all the specified functional requirements, and the defects identified during this process were successfully resolved.

The next stages of the research include the full integration of the agent into a Robo-Advisor and the expansion of data sources through public financial reports. There is also a need to create a formalised methodology for evaluating the quality of the agent's responses, which will allow for comparing the effectiveness of different models for response generation (LLM evaluation) and models for creating embeddings (RAG evaluation).

ADDITIONAL INFORMATION

DECLARATION ON THE USE OF GENERATIVE ARTIFICIAL INTELLIGENCE TOOLS

In preparing this work, the author used DeepL Translate and Kagi Translate for translation, grammar checking, and rephrasing. The author also used the Elicit and NotebookLM services for literature search and preliminary analysis of academic sources. After using these tools, the author carefully reviewed and edited the content and assumes full responsibility for the content of this publication.

REFERENCES

1. Martins, M. N., & Ashofteh, A. (2023). A systematic review on robo-advisors in FinTech. In CAPSI 2023 proceedings (Article 15, pp. 160–185). APSI – Associação Portuguesa de Sistemas de Informação. <https://doi.org/10.18803/capsi.v23.160-185>
2. European Commission. (2025). Diia.AI: Ukraine's national AI agent for government services. Public Sector Tech Watch. Retrieved March 25, 2026, from <https://interoperable-europe.ec.europa.eu/collection/public-sector-tech-watch/diiaai-ukraines-national-ai-agent-government-services>
3. Organisation for Economic Co-operation and Development. (2025). Diia AI assistant. OECD.AI Policy Observatory. Retrieved March 25, 2026, from <https://oecd.ai/en/dashboards/policy-initiatives/diia-ai-assistant>
4. Chen, T., & Gasco-Hernandez, M. (2025). Uncovering the results of AI chatbot use in the public sector: Evidence from US state governments. Public Performance & Management Review, 48(6), 1331–1356. <https://doi.org/10.1080/15309576.2024.2389864>
5. Sheth, J. N., Jain, V., Roy, G., & Chakraborty, A. (2022). AI-driven banking services: The next frontier for a personalised experience in the emerging market. International Journal of Bank Marketing, 40(6), 1248–1271. <https://doi.org/10.1108/IJBM-09-2021-0449>
6. Hasanah, E. N., Wiryo, S. K., & Koesrindartoto, D. P. (2023). Financial robo-advisor: Learning from academic literature. Jurnal Minds: Manajemen Ide dan Inspirasi, 10(1), 17–40. <https://doi.org/10.24252/minds.v10i1.33428>
7. Maureen, C. A. (2026). AI-powered financial forecasting: Methodologies, applications, and case studies. International Journal of Scientific Research and Modern Technology, 5(1), 128–133. <https://doi.org/10.38124/ijrmt.v5i1.1183>
8. Kanikanti, V. S. N. (2023). Reinforcement learning for optimizing investment portfolios in robo-advisory platforms. American International Journal of Computer Science and Technology, 5(4), 23–32. <https://doi.org/10.63282/3117-5481/AIJCS-V5I4P103>
9. Singh, A., Ehtesham, A., Kumar, S., Khoei, T. T., & Vasilakos, A. V. (2025). Agentic retrieval-augmented generation: A survey on agentic RAG. arXiv. <https://doi.org/10.48550/arXiv.2501.09136>
10. Zhao, X., Sun, T., Ren, S., Yang, J., & Liu, Y. (2025). RAG-based AI agents for enterprise software development: Implementation patterns and production deployment. Frontiers in Artificial Intelligence Research, 2(3), 501–520. <https://doi.org/10.71465/fair456>
11. George, R., Srinivasan, A. G., Joe, J. K., R, H. M., J, V., Kant, H., Vimalkanth, R., S, S., & Suresh, S. (2025). Enhancing financial RAG with agentic AI and Multi-HyDE: A novel approach to knowledge retrieval and hallucination reduction. Proceedings of the 10th Workshop on Financial Technology and Natural Language Processing (pp. 19–32). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2025.finnlp-2.3>
12. Theja, R. (2023, November 6). Evaluate RAG with LlamaIndex. OpenAI Cookbook. https://developers.openai.com/cookbook/examples/evaluation/evaluate_rag_with_llamaindex
13. Tailor, P. (2025). Retrieval-augmented generation (RAG) for real-time financial market analysis. The American Journal of Interdisciplinary Innovations and Research, 7(7), 137–144. <https://doi.org/10.37547/tajir/Volume07Issue07-12>
14. Prakash, S. (2025). The role of cloud-based vector databases and retrieval-augmented generation (RAG) for generative AI in financial markets analysis. World Journal of Advanced Engineering Technology and Sciences, 15(3), 1609–1618. <https://doi.org/10.30574/wjaets.2025.15.3.1034>
15. Li, X., Jia, P., Xu, D., Wen, Y., Zhang, Y., Zhang, W., Wang, W., Wang, Y., Du, Z., Li, X., Liu, Y., Guo, H., Tang, R., & Zhao, X. (2025). A survey of personalization: From RAG to agent. arXiv. <https://doi.org/10.48550/arXiv.2504.10147>
16. Murali, G., Kumar, T. S., Ahmed, S. F., Hemanth, U., & Subhash, P. V. S. (2026). FinAgent: A privacy-preserving multi-agent financial intelligence system with context-aware governance and GraphRAG reasoning. International Journal for Research in Applied Science & Engineering Technology, 14(3), 2590–2598. <https://doi.org/10.22214/ijraset.2026.77845>

Сергій САВЧЕНКО
Хмельницький національний університет

ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ RAG-ОРІЄНТОВАНОГО ШІ-АГЕНТА ДЛЯ СИСТЕМ ROBO-ADVISOR

Дана стаття присвячена проектуванню та прикладній реалізації агентної системи на основі архітектури Retrieval-Augmented Generation (RAG) для сервісів автоматизованого фінансового консультування (Robo-Advisors, RA). Актуальність дослідження полягає в обмеженості знань великих мовних моделей (LLM) у доменно-специфічних областях, таких як фінансова галузь, де фактологічна помилка може спричинити прямі грошові втрати та регуляторні ризики. За результатами огляду літератури визначено клас задач сучасної Robo-Advisor-платформи, які доцільно розв'язувати засобами RAG. Мета цього дослідження полягає у розробці архітектури RAG-агента для Robo-Advisor сервісу та перевірці його працездатності на діючому прототипі. Було сформовано функціональні вимоги RA-модуля для відстеження поведінки фінансових активів, що входять до певного інвестиційного портфелю. Проведено аналіз інструментів, за допомогою яких реалізуються типові сценарії взаємодії з RAG системою (отримання даних, попередня обробка текстової інформації, векторизація тексту, нормалізація даних, взаємодія з LLM тощо). Спроектовано програмну архітектуру, у якій агентні функції (отримання інформації про курси акцій, аналіз фінансових новин, автоматичні звіти, спілкування з кінцевим користувачем) відображено в окремих програмних модулях, а систему розділено на асинхронний контур індексації та синхронний контур взаємодії з користувачем. Пам'ять агента реалізовано як гібридне сховище, у якому реляційна база є джерелом істини, а векторний індекс є повністю відновлюваним похідним артефактом. Запропоновано метод переранжування результатів пошуку за новинами з нормованим коефіцієнтом часу їх публікації. Взаємодію з користувачем реалізовано у вигляді чат-бота для месенджера. Дослідна експлуатація підтвердила відповідність системи поставленим вимогам; описано виявлені недоліки та обмеження агента, а також способи їх усунення. Результати даного дослідження можуть бути використані для впровадження RAG-агента у повноцінні Robo-Advisor-сервіси.

Ключові слова: Agentic AI, Large Language Model, Retrieval-Augmented Generation, Robo-Advisor, Investment Portfolio Optimization, FinTech.