Vol.7 No.1 Tahun 2026
Jurnal Kecerdasan Buatan, Komputasi dan Teknologi Informasi
https://ejournal.unuja.ac.id/index.php/core
E-ISSN : 2774-7875 and P-ISSN : 2775-0124

Python-Based Telegram FAQ Chatbot Using Text Retrieval

Mochammad Faid[1], Sudriyanto[2], Moh. Sukron[3], Syaiful[3]
[1,2,3,4] Information Technology Study Program, Universitas Nurul Jadid, Probolinggo, Indonesia

Article Info

Article history:

Received : 05 24, 2026
Revised : 05 30, 2026
Accepted : 06 15, 2026

Keywords:

Telegram chatbot;
FAQ automation;
text retrieval;
Python;
difflib.

Abstract

This study presents the implementation of a Telegram FAQ chatbot developed in Python to address the difficulty of providing rapid and consistent responses to recurring public-service questions. The system stores question-and-answer pairs in memory, extracts FAQ entries from free-form text, and retrieves the closest answer using text similarity through Python's difflib library. The prototype also provides inline menus, an interactive quiz, FAQ reset and update commands, and an optional large language model refinement stage constrained by the retrieved context. The research followed an implementation-oriented prototyping approach comprising requirements identification, code analysis, system design, implementation review, and scenario-based functional inspection. Static analysis identified 14 principal functions, 273 built-in FAQ pairs, and 100 quiz questions. The implementation analysis indicates that lightweight retrieval can support a practical and resource-efficient FAQ automation prototype while maintaining answers within the available knowledge base. Nevertheless, production deployment requires stronger secret management, persistent storage, access control, logging, and empirical evaluation using real user questions. The study contributes a replicable architecture for controlled FAQ automation on Telegram and establishes priorities for subsequent reliability and usability testing.

Corresponding Author:

Mochammad Faid
mfaid@unuja.ac.id

INTRODUCTION

The rapid development of digital services has changed the way people access information. Users no longer rely solely on face-to-face services, email, or websites, but increasingly expect information to be delivered quickly through messaging applications they already use in their daily lives. In public services, education, administration, and business, users frequently submit repetitive questions concerning account registration, data updates, service requirements, verification procedures, transactions, schedules, and feature usage. When these questions are handled entirely by human staff, organizations may face increased workloads, delayed responses, and difficulties in maintaining consistent information. (Talenggoran et al. 2025) reported that information services relying on manual responses often encounter repeated inquiries and delayed assistance, particularly when questions are submitted outside official working hours.

Chatbots offer a potential solution to these challenges. A chatbot is a computer program designed to interact with users through natural language, either in text or voice form. Chatbot technology has evolved from simple rule-based and pattern-matching systems into more advanced approaches involving natural language processing, machine learning, and

large language models (Adamopoulou & Moussiades, 2020). Despite these advances, Frequently Asked Questions chatbots remain relevant, particularly for services that require concise, structured, and authoritative responses.

Telegram was selected as the communication platform in this study because it provides a Bot API that enables developers to build automated conversational systems relatively easily. The Telegram Bot API supports commands, message handlers, inline keyboards, callback queries, document delivery, and automated notifications (Telegram, n.d.). Telegram is also accessible across multiple devices and allows developers to build chatbot services without creating a standalone messaging application (Alfaiz and Maryam 2021). Used the Telegram Bot API and Python to develop a student orientation registration system. The system supported registration, data management, class assignment, and data export. (Firdaus and Najiyah 2024) further integrated a Telegram Bot with Google Sheets and Looker Studio to support daily reporting, user validation, data retrieval, and report visualization. These studies show that Telegram Bots can function not only as conversational interfaces but also as gateways for broader information management processes.

Python was used as the programming language in this study because of its relatively simple syntax and its extensive support for chatbot development and text processing. The `python-telegram-bot` library provides components for handling commands, text messages, interactive buttons, and callbacks through classes such as `ApplicationBuilder`, `CommandHandler`, `MessageHandler`, and `CallbackQueryHandler` (python-telegram-bot Authors, n.d.). Python also provides the `difflib` module, which can be used to compare text similarity and identify strings that most closely match a user query (Python Software Foundation, n.d.). The suitability of Telegram as a chatbot platform is also supported by previous studies. Astari et al. (2023) developed the "Tanya Zaid" Telegram chatbot as an interactive learning medium that allowed users to access learning materials through menus, buttons, and keyword-based queries. Ferelestian et al. (2023) likewise developed a Telegram chatbot for student information services using Wit.ai and reported that the system could process user requests and provide appropriate responses with an accuracy of 97%. Although these studies employed different development tools, both demonstrate that Telegram can support structured conversational flows and rapid information delivery. These features make Python and Telegram suitable for developing a lightweight FAQ chatbot prototype that is easy to understand, maintain, and extend.

Telegram Bots have been applied in a variety of domains. (Sudiatmika and Dewi 2020) developed an e-learning system based on a Telegram Bot for delivering learning materials, assignments, and academic scores. (Hidayat et al. 2021) used the Telegram Quiz Bot to support English listening instruction. (Agustini et al. 2025) developed an interactive quiz platform that integrated a Telegram Bot with generative artificial intelligence, while (Maharani et al. 2026) used Telegram Bot-based interactive multimedia to improve physics learning outcomes. In the context of community services, (Mulyana et al. 2023) implemented a Telegram chatbot to distribute event schedules, manage registrations, send notifications, and collect feedback. (Umam et al. 2025) also developed a Telegram chatbot to provide information about mobile phone specifications and prices. These implementations demonstrate that Telegram Bots are flexible enough to support information services, education, administration, and community interaction.

This study aims to describe the architecture of a Python-based Telegram FAQ chatbot, explain the text retrieval process, identify the functions available in the prototype, and analyze its limitations and development requirements before production deployment. The main contribution of this study is a lightweight, transparent, and replicable chatbot architecture that does not require model training. The system also preserves responses from a controlled FAQ knowledge base, making it suitable as an initial prototype for information services that require fast and consistent answers.

DOI: https://doi.org/10.33650/coreai.v6i2

METHOD

This study employed an implementation-oriented research design using a prototyping approach, focusing on the development and analysis of a Python-based Telegram FAQ chatbot. The research object was the program source code containing FAQ retrieval, user-interaction management, menus, callbacks, and quiz features. The research stages comprised requirements analysis, system design, implementation, and testing. These stages are consistent with the work of Riyanto and Asmunin (n.d.), who developed an academic information chatbot through requirements analysis, system design, implementation, and testing, as well as Umam et al. (2025), who applied user-needs analysis, conversational-flow design, Telegram API implementation, and black-box testing. The research data consisted of the program source code, default FAQ text, Standard Operating Procedure documents, procurement-service materials, and a collection of quiz questions. The data were obtained from the implementation artifacts and examined through documentation and static inspection of the code structure, data representation, principal functions, handler registration, user-state transitions, FAQ question-and-answer pairs, and message-processing flow.

The system was implemented using the python-telegram-bot library to handle commands, text messages, interactive buttons, and callbacks through components such as ApplicationBuilder, CommandHandler, MessageHandler, and CallbackQueryHandler (python-telegram-bot Authors, n.d.). The text retrieval process employed the difflib.get_close_matches function to compare user queries with the questions stored in the FAQ knowledge base and select the closest textual match (Python Software Foundation, n.d.). Before the matching process, the input text was normalized by converting it to lowercase, removing unnecessary characters, and standardizing whitespace. The system was analyzed descriptively and structurally by tracing the process from the receipt of a user message, handler execution, and FAQ comparison to response generation. Testing included an examination of the text retrieval logic, user-state transitions, quiz flow, and functional behavior using a black-box testing approach. Network integration testing was separated from logic-level inspection because full chatbot operation required an active Telegram bot token and an internet connection. The procedure was conducted chronologically. First, the service problem and user interaction requirements were identified. Second, the program structure and knowledge-base format were inspected. Third, the overall architecture and state transitions were mapped. Fourth, the FAQ parser and retrieval logic were reconstructed as readable code listings. Fifth, scenario-based functional inspection was performed for the primary commands and interaction states. Finally, the implementation was evaluated using coverage of required functions, static counts, retrieval behavior, and identified operational risks. Figure 1 summarizes this workflow

Table 1. Research stages and outputs

Stage	ACTIVITY	Output
Requirements identification	Analyze the FAQ service objectives, user needs, and expected interactions.	System requirements
Code and data analysis	Inspect functions, FAQ data, Telegram handlers, and quiz-state logic.	Component map and static metrics
System design	Model the input, retrieval, response, menu, update, and quiz flows.	Architecture and process design
Implementation review	Document the Python functions and event-handler relationships.	Implementation description
Functional inspection	Trace representative scenarios for commands, retrieval, updates, AI refinement, and quiz interaction.	Scenario coverage findings
Evaluation	Assess strengths, limitations, and production-readiness requirements.	Recommendations for development

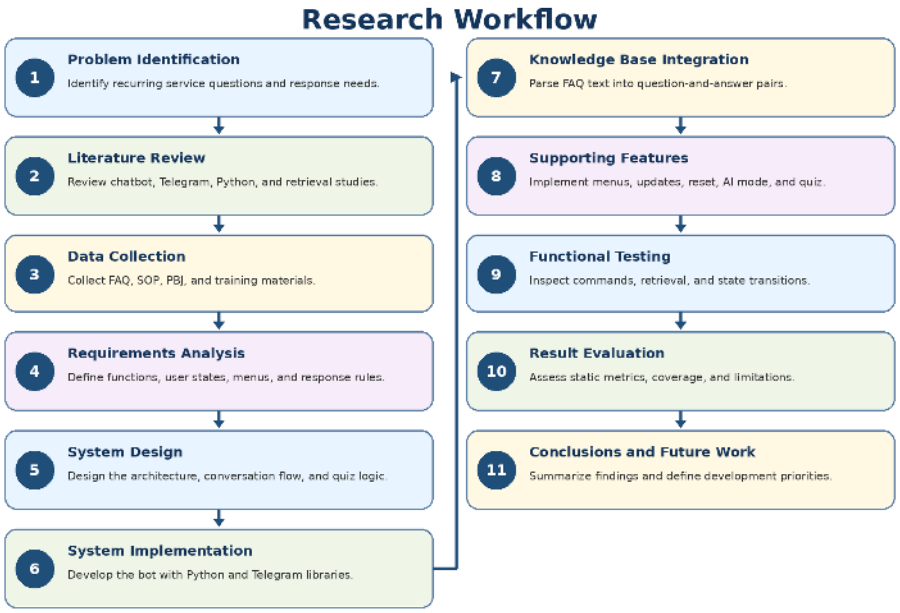

Figure. 1. Research workflow for the Telegram FAQ chatbot implementation.

When a message reaches the text handler, the system first evaluates the current user state. A quiz-state message is processed as an answer and immediately scored, whereas an FAQ-update state activates the parser and replaces or extends the knowledge base. Under normal conditions, the message is passed to the retrieval function. If a matching FAQ is found, the answer is returned directly or refined using the optional language-model stage. If no candidate satisfies the matching rules, the bot informs the user that the requested information is not available.

Table 2. General system architecture

Component	Python Implementation	Function
Telegram interface	ApplicationBuilder, CommandHandler, MessageHandler, CallbackQueryHandler	Receives commands, text messages, and inline-button callbacks.
Knowledge base	default_faq, faq_raw_text, faq_pairs	Stores the built-in FAQ content and user-provided updates.
FAQ parser	extract_faq_pairs()	Converts free-form FAQ text into question-and-answer pairs.
Answer retrieval	best_answer()	Selects the FAQ question most similar to the user input.
AI refinement	llm_refine()	Rephrases the retrieved answer while remaining constrained by its context.
Menu and quiz	main_menu(), aibot(), start_quiz(), handle_text()	Provides menus, chatbot mode, quiz state, answer checking, and scoring.

Table 3. Main interaction scenarios

Input condition	System process	Expected output
The user enters /start.	Initialize the default FAQ and display the main menu.	Greeting message and inline menu.
The user selects Ask the Bot.	Enable answer-refinement mode in user_data.	Instruction to submit a question.
The user submits a service question.	best_answer() retrieves the closest FAQ entry.	FAQ answer or an unavailable-answer notice.

DOI: https://doi.org/10.33650/coreai.v6i2

The user submits new FAQ text.	extract_faq_pairs() parses questions and answers.	Updated FAQ collection.
The user selects the quiz.	start_quiz() creates the quiz state and displays a question.	Reply keyboard containing answer choices.
The user answers a quiz question.	handle_text() checks the option and updates the score.	Correct/incorrect feedback and the next question.

RESULTS AND ANALYSIS

The chatbot was implemented in a single Python source file. The program imports Telegram classes for message formatting and handlers, along with logging, regular expressions, difflib, and randomization modules. The implementation optionally uses an OpenAI-compatible client through OpenRouter for answer refinement; this feature can be disabled through the USE_LLM configuration flag. The static inspection identified 14 principal functions, 273 default FAQ pairs after parsing, and 100 quiz questions.

The default knowledge base is stored as free-form text and converted into structured pairs by extract_faq_pairs(). Each line ending in a question mark is treated as a new question, and the following lines are accumulated as its answer until another question is encountered. When this pattern is absent, a fallback mechanism attempts to interpret tab-separated or multi-space-separated question-and-answer columns. This dual strategy makes the parser tolerant of moderately inconsistent source formatting while retaining an easily editable text representation.

```
import re
def extract_faq_pairs(raw: str) -> list[tuple[str, str]]:
    if not raw:
        return []
    lines = [line.rstrip() for line in raw.splitlines()]
    pairs: list[tuple[str, str]] = []
    current_question: str | None = None
```

```
    answer_lines: list[str] = []
    def commit() -> None:
        question = (
            _normalize(current_question)
            if current_question
            else None
        )
        answer = _normalize("\n".join(answer_lines))
        if question and answer:
            pairs.append((question, answer))
    for line in lines:
        if re.search(r"\?\s*$", line):
            if current_question is not None:

                commit()
                answer_lines = []
            current_question = line
        elif current_question is not None:
            answer_lines.append(line)
    if current_question is not None:
```

```
            commit()
        if not pairs:
            for line in lines:
                columns = re.split(r"\t+|\s{2,}", line)
                if len(columns) < 2 or len(columns[0]) <= 4:
                    continue
                question = _normalize(columns[0])
                answer = _normalize(" ".join(columns[1:]))
                if question and answer:
                    pairs.append((question, answer))
        return pairs
```

Listing 1. Core function for extracting FAQ pairs from free-form text.

The best_answer() function performs lightweight retrieval. It creates a list of stored questions and calls difflib.get_close_matches() with n = 1 and a cutoff of 0.55. If no candidate reaches the threshold, the function attempts a simple substring search. The threshold balances tolerance for minor wording differences against the risk of returning an unrelated entry. Because the method compares the input with complete stored questions rather than semantic embeddings, performance depends on lexical overlap and the consistency of FAQ phrasing.

```
def best_answer(user_text: str) -> str | None:
    if not FAQ_PAIRS:
        return None
    questions = [
        question
        for question, _ in FAQ_PAIRS
    ]
    matches = difflib.get_close_matches(
        user_text,
        questions,
        n=1,
        cutoff=0.55,
    )
    if not matches:
        normalized_user_text = user_text.lower()
        for question, answer in FAQ_PAIRS:
            if normalized_user_text in question.lower():
                return answer
        return None
    selected_question = matches[0]
    selected_index = questions.index(selected_question)
    return FAQ_PAIRS[selected_index][1]
```

Listing 2. Retrieval of the closest FAQ answer using text similarity

Telegram integration is established through ApplicationBuilder. Three primary commands are registered: /start initializes the bot and displays the menu, /ingatfaq activates the FAQ-update state, and /resetfaq restores the built-in collection. CallbackQueryHandler processes inline-button actions, whereas MessageHandler processes ordinary text that is not a command. This separation reduces ambiguity among command, callback, and free-text interactions and improves maintainability.

DOI: https://doi.org/10.33650/coreai.v6i2

```
if __name__ == "__main__":
    _init_default_faq()
    app = (
        ApplicationBuilder()
        .token(TELEGRAM_TOKEN)
        .build()
    )
    app.add_handler(CommandHandler("start", start))
    app.add_handler(CommandHandler("ingatfaq", ingatfaq))
    app.add_handler(CommandHandler("resetfaq", resetfaq))
    app.add_handler(CallbackQueryHandler(handle_callbacks))
    app.add_handler(
        MessageHandler(
            filters.TEXT & ~filters.COMMAND,
            handle_text,
        )
    )
    logger.info("FAQ Bot is active")
    app.run_polling()
```

Listing 3. Registration of Telegram commands, callbacks, and text handlers.

The main_menu() and aibot() functions construct inline buttons for service links, the chatbot mode, procurement quizzes, consultation appointments, Standard Operating Procedures, and training information. The optional refinement mode stores an ai_refine flag in context.user_data. The quiz module uses the same user-state mechanism to retain the current question index, score, and total number of questions. Consequently, one text handler can route messages according to the active conversational context without requiring a separate bot instance for each feature.

Figure 2. The chatbot retrieves an FAQ response and presents follow-up navigation buttons.

The response example in Figure 2 demonstrates the intended controlled-answer behavior. The bot identifies a question concerning a profile change, returns the corresponding procedural explanation from the FAQ collection, and provides buttons for returning to the menu or asking another question. The screenshot contains Indonesian interface text because the prototype targets Indonesian public-service users; all analytical descriptions in this manuscript are presented in English.

Figure 3. Main menu and Ask the Bot interface of the Telegram service.

Figure 4. Interactive procurement quiz with reply-button answer choices.

Figures 3 and 4 show that the prototype extends beyond ordinary question answering. The menu exposes multiple service functions through buttons, reducing dependence on manually typed commands. The quiz feature presents one question at a time, displays selectable answer options, checks the selected response, and maintains the score in the user session. These features demonstrate the reuse of Telegram interaction components for both information delivery and guided learning.

Table 4. Scenario-based functional inspection results

Scenario	Implementation evidence	Expected behavior	Assessment
/start	CommandHandler registration and start() menu construction.	Initializes FAQ and shows the menu.	Covered
FAQ retrieval	best_answer() similarity and fallback logic.	Returns a matched answer or None.	Covered
FAQ update	Update state and extract_faq_pairs().	Parses and stores new pairs.	Covered
FAQ reset	/resetfaq handler and default initializer.	Restores built-in FAQ data.	Covered
AI refinement	ai_refine flag and llm_refine() call path.	Rephrases only the retrieved context.	Covered
Quiz interaction	start_quiz() and state-aware handle_text().	Scores answers and advances questions.	Covered
Production security	Token and external API configuration.	Protects credentials and restricts updates.	Needs strengthening
Persistence	In-memory FAQ and user state.	Survives restart and supports audit history.	Not implemented

The functional inspection indicates that the implementation contains the control paths required for its principal features. Nevertheless, the results are limited to static and logic-level analysis rather than a controlled user experiment or complete production integration test. The prototype stores FAQ data and conversational states in memory, so a process restart can remove updates and active sessions. Credentials for Telegram and external language-model services must also be managed outside the source code, following established secret-management practices (OWASP Foundation, n.d.). In addition, unrestricted FAQ-update commands could allow unauthorized modification unless administrator checks are introduced.

From a retrieval perspective, difflib provides a transparent baseline but does not represent meaning beyond textual similarity. Questions with different wording but equivalent intent may fail to reach the threshold, while short generic phrases may match incorrectly. Future evaluations should therefore use a labeled set of real user questions to measure retrieval accuracy, precision, recall, response time, and failure categories. Embedding-based retrieval or a hybrid lexical-semantic approach can then be compared with the current baseline while retaining the principle that generated responses must remain grounded in the retrieved FAQ content.

CONCLUSIONS

This study documented a Python-based Telegram FAQ chatbot that combines free-form FAQ parsing, lightweight text retrieval, inline menus, dynamic FAQ updates, FAQ reset, optional context-constrained language-model refinement, and an interactive quiz. Static analysis identified 14 principal functions, 273 built-in FAQ pairs, and 100 quiz questions. The architecture demonstrates that difflib-based retrieval can support a practical, transparent, and resource-efficient prototype for recurring public-service questions, particularly when responses must remain tied to an existing knowledge base. The research contribution is a reproducible implementation structure that integrates retrieval, Telegram handlers, and user-state management within one controlled conversational workflow. The findings should be interpreted as an implementation and logic-level evaluation rather than evidence of production accuracy or user satisfaction. The prototype has not yet been tested with a representative set of real user questions, complete Telegram deployment conditions, or concurrent users. It also requires persistent storage, role-based authorization for knowledge-base updates, secure secret management, structured logging, backup and recovery procedures, and systematic monitoring. Future research should conduct empirical retrieval and usability testing, compare lexical retrieval with semantic or hybrid methods, evaluate latency and concurrency, and assess whether constrained answer refinement improves readability without introducing information outside the retrieved source.

REFERENCES

Adamopoulou, E., & Moussiades, L. (2020). Chatbots: History, technology, and applications. *Machine Learning with Applications, 2*, Article 100006. https://doi.org/10.1016/j.mlwa.2020.100006

Agustini, D., Farida, M., Rosadi, M. E., Firdaus, M. I., & Muttaqin, R. (2025). Platform kuis interaktif pembelajaran anak sekolah dengan integrasi bot Telegram berbasis generatif AI. *Jurnal Nasional Komputasi dan Teknologi Informasi, 8*(1), 532–541.

Alfaiz, F. Z., & Maryam. (2021). Implementation Telegram chat bot on student orientation period registration system for efficiency of data management. *Jurnal Teknik Informatika, 2*(2), 85–93. https://doi.org/10.20884/1.jutif.2021.2.2.56

Astari, M. R., Mardlian, M. S. A. K., Bahri, S., & Siregar, M. U. (2023). Rancangan aplikasi chatbot Telegram "Tanya Zaid" sebagai media pembelajaran nahwu. Prosiding Konferensi Integrasi Interkoneksi Islam dan Sains, 5, 313–323.

Ferelestian, V. J., Susanto, B., & Senapartha, I. K. D. (2023). Pengembangan Telegram chatbot informasi mahasiswa menggunakan Wit.ai. JUTEI, 7(2), 89–97. https://doi.org/10.21460/jutei.72.257

Firdaus, F., & Najiyah, I. (2024). Implementasi bot Telegram untuk otomatisasi pelaporan harian: Studi kasus PT Dirgantara Indonesia. *E-Proceeding Teknik Informatika, 5*(1), 67–79.

Furqan, M., Sriani, & Shidqi, M. N. (2023). Chatbot Telegram menggunakan natural language processing. *Walisongo Journal of Information Technology, 5*(1), 15–26. https://doi.org/10.21580/wjit.2023.5.1.14793

Hidayat, R., Susanto, F., Rahayu, E. M., Hertiki, Arbani, A. N., & Soelistijowati, J. O. (2021). Pemanfaatan Quiz Bot Telegram dalam pembelajaran menyimak bahasa Inggris. *Jurnal Penamas Adi Buana, 4*(2), 76–86.

Maharani, A. A., Khotimah, K., & Dewi, U. (2026). Pemanfaatan multimedia interaktif bot Telegram untuk meningkatkan hasil belajar fisika pada materi gerak harmonik sederhana. *SCIENCE: Jurnal Inovasi Pendidikan Matematika dan IPA, 6*(3), 1916–1928. https://doi.org/10.51878/science.v6i3.13384

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press.

Mulyana, D. I., Lestari, D., Ramdhani, F., Ruliansyah, M. J., & Beay, R. (2023). Implementasi chatbot Telegram dalam meningkatkan partisipasi kegiatan warga. *Jurnal Pengabdian kepada Masyarakat Nusantara, 4*(2), 866–874.

Python Software Foundation. (n.d.). *difflib—Helpers for computing deltas*. Python documentation. https://docs.python.org/3/library/difflib.html

python-telegram-bot Authors. (n.d.). *python-telegram-bot documentation*. https://docs.python-telegram-bot.org/

Riyanto, N. D., & Asmunin. (n.d.). *Perancangan dan pembuatan aplikasi chatbot Telegram untuk layanan informasi akademik Fakultas Vokasi Universitas Negeri Surabaya metode natural* language *processing menggunakan Wit.ai* [Unpublished manuscript]. Fakultas Vokasi, Universitas Negeri Surabaya.

Setiyawan, M. A., & Kenoya, E. D. (2025). Pengembangan chatbot Telegram FAQ layanan ICT menggunakan algoritma Random Forest dan metode Word2Vec. *INTI Nusa Mandiri, 20*(1), 150–162. https://doi.org/10.33480/inti.v20i1.5766

Shawar, B. A., & Atwell, E. (2007). Chatbots: Are they really useful? *LDV Forum, 22*(1), 29–49.

Sudiatmika, I. P. G. A., & Dewi, K. H. S. (2020). E-learning berbasis Telegram Bot. *Jurnal Riset Inovasi* Bidang *Informatika dan Pendidikan Informatika, 1*(2), 49–60.

Sutiyo, F. R. A., Harahap, N. S., Agustian, S., & Candra, R. M. (2024). Implementasi question answering berbasis chatbot Telegram pada Tafsir Al-Jalalain menggunakan LangChain dan LLM. *KLIK: Kajian Ilmiah Informatika dan Komputer, 4*(5), 2464–2472. https://doi.org/10.30865/klik.v4i5.1784

Talenggoran, R. H. H., Krisnawati, L. D., & Virginia, G. (2025). Penerapan framework RASA untuk membangun sistem FAQ Bot sebagai layanan informasi BIRO 3. *JUTEI, 9*(2), 81–92. https://doi.org/10.21460/jutei.2025.92.431

Umam, A. K., Wijayanti, E., & Chamid, A. A. (2025). Pengembangan chatbot pada platform Telegram sebagai media informasi seputar handphone. *Bit-Tech, 8*(1), 33–40. https://doi.org/10.32877/bt.v8i1.2150