Designing AI Customer Service Chatbot, Final Report

1st Raeesa Karodia
Computer Science with AI
University of Nottingham
Semenyih, Malaysia
hfyrk3@nottingham.edu.my

2nd Farah Tamer Elsaid
Computer Science with AI
University of Nottingham
Semenyih, Malaysia
hfyfe13@nottingham.edu.my

***Abstract*—A hybrid machine learning (ML) and rule-based tech-support chatbot was created to detect user emotion and maintain context. Term Frequency-Inverse Document Frequency (TF-IDF) features and a logistic regression model are used to classify sentiment as positive, neutral, or negative, and tone-aware templates are applied to responses. Queries are routed through an intent recognizer and FAQ (Frequently Asked Questions) matcher, with a fallback to an external large language model (LLM) via an application programming interface (API) called OpenRouter, and multilingual support is provided by TranslatePy. The system is deployed as a Flask web application using Jinja2 and JavaScript, with response times of less than 200 milliseconds (ms). Automated tests and a 20-participant (N = 20) user study show more than 90% success across multiple scenarios.**

***Index Terms*—chatbot, machine learning, logistic regression, sentiment analysis, rule-based matching, multi-turn context**

Code Repository

The source code is available at:
Github Repository:ai-customer-support-chatbot)

I. Introduction

Customer support chatbots are often expected to handle large volumes of user queries, yet many struggle to recognise emotional cues and maintain coherent conversations. This project develops a hybrid chatbot that combines a custom TF-IDF + Logistic Regression sentiment classifier with rule based FAQ matching and tone aware response templates. By retaining up to five prior exchanges, the system adapts its phrasing to detect user emotions while ensuring accurate query responses.

To validate the design, a rigorous testing framework was implemented. It includes automated evaluations (measuring classification accuracy, response relevance, BLEU scores, and end to end latency) and human trials (gathering feedback on clarity, empathy, and satisfaction). This dual approach allows for structured experiments and clear evaluation metrics, ensuring both quantitative and real world usability insights.

Identify applicable funding agency here. If none, delete this.

II. Literature Review

A. Introduction

Artificial Intelligence (AI) chatbots have become integral to customer support, offering 24/7 assistance and operational efficiency. However, despite widespread adoption, many chatbots fall short of user expectations due to inherent limitations (Nicolescu & Tudorache, 2022). This literature review examines these shortcomings and justifies the AI methods used in our chatbot to address current technological gaps and enhance user experience.

B. Limitations of Current Chatbots

Recent studies highlight persistent limitations in customer support chatbots. While many perform adequately on simple tasks, they often struggle with complex queries, delivering generic responses that frustrate users and necessitate human escalation (Adam et al., 2020). A major issue is the lack of context awareness, where chatbots fail to retain conversation history, resulting in repetitive questioning and fragmented interactions (Caldarini et al., 2021; IBM, 2017). Emotional intelligence is another gap; chatbots frequently mishandle sensitive interactions, leading to impersonal experiences (Denecke et al., 2024). Poor integration with business systems often results in outdated information (Følstad et al., 2021), and reliance solely on natural language processing (NLP) increases the risk of misinterpreting ambiguous inputs (Nicolescu & Tudorache, 2022). Despite advances in AI driven platforms such as IBM Watson Assistant, Drift, and Intercom, significant limitations remain. While these systems leverage machine learning to improve natural language understanding, challenges persist in managing complex multi turn dialogues (Caldarini et al., 2021), emotional sensitivity (Denecke et al., 2024), and seamless human handover (Følstad et al., 2021). This ongoing gap highlights the need for enhanced hybrid architectures, as proposed in our chatbot design.

C. Justification of Selected AI Methods

Current customer support chatbots face significant limitations, including poor emotional intelligence and inefficient query handling (Denecke et al., 2024; Nicolescu & Tudorache, 2022). To overcome these challenges, our chatbot integrates five advanced AI methods: sentiment analysis, intent recognition, a custom developed FAQ matching system, the OpenRouter API, and TranslatePy. Sentiment analysis, enhanced by a custom training model developed using three bespoke datasets, detects the emotional tone of user messages, enabling empathetic and contextually appropriate responses (Saikh et al., 2018; Widyaningrum et al., 2019). This directly addresses the emotional intelligence gap. Intent recognition ensures multi turn understanding, maintaining dialogue continuity and reducing misinterpretation (Caldarini et al., 2021; Kumar & Mahmoud, 2020). For routine queries, our FAQ matching system, trained on an improved version of the Tech Support Conversations Dataset (Kumar, 2024), efficiently handles common inquiries, serving as a reliable fallback mechanism that reduces the need for human escalation (Khennouche et al., 2024). This tackles the query handling efficiency issue. Complex or unpredictable queries are managed through the OpenRouter API, which accesses multiple large language models (LLMs) to enhance adaptability and precision (Pandey & Sharma, 2023; OpenRouter, 2025). Additionally, TranslatePy supports multilingual communication, expanding accessibility to non English users (Nayak & Nair, 2024). Together, these methods embed context sensitivity into the chatbot's structure, addressing deficiencies in emotional handling, conversation flow, query complexity, multilingual support, and system integration.

D. *Conclusion*

Customer support chatbots continue to face challenges in complex query handling, emotional intelligence, and system integration. Our chatbot addresses these gaps through targeted AI methods, offering a more intelligent, empathetic, and accessible solution. These improvements not only rectify current deficiencies but also lay a foundation for future advancements in customer support technology.

III. METHODOLOGY

An eight phase, hybrid ML (machine learning) + rule based approach was selected to build a sentiment aware, context sensitive tech support chatbot:

A. *Problem Definition*

The goal was defines as the development of a chatbot that not only answers technical queries but also adapts its tone based on detected user emotion.

B. *Data Collection*

Three complementary datasets (csv files) were assembled.

- Emotional sentiment phrases (used to capture tone)
- Slang heavy sentiment phrases (used to train the chatbot on informal input)
- Formal tech support FAQs (used for accurate responses to technical queries)

C. *Data Preprocessing*

- Vague or incomplete FAQ entries were cleaned and standardized
- Each dataset was expanded upon with realistic conversational examples
- Positive, negative, and neutral sentiment classifications were created and balanced.

D. *Model Training*

A custom sentiment classifier using TF-IDF (Term Frequency-Inverse Document Frequency) feature extraction and Logistic Regression was trained on the balanced datasets.

E. *Backend Development*

Core chatbot logic was implemented using Python:

- Sentiment Analyzer to classify incoming messages
- Intent Recognizer (keyword + fuzzy matching) to route queries
- FAQ Matcher for precise answer retrieval
- Tone aware Templating to vary response style
- OpenRouter Fallback Logic via HTTP (requests) for semantic intent/sentiment when rules fail
- Language Translation Flow (using Translatepy) for multilingual support
- Session & Chat Memory Handling (Flask session storage of last five exchanges)

F. *Context Awareness*

Multi turn memory that retains the last five exchanges in the user's Flask session was built, thus, allowing the bot to resolve vague references by inferring context from recent dialogue.

G. *Frontend Development*

The chatbot was exposed via a Flask based web interface using Jinja2 templates, enabling real time browser chat with no heavy dependencies on the client side.

H. *Testing*

Manual and automated tests were conducted across the following: Emotional inputs (positive, neutral, and negative), informal enquiries or slang, and edge cases (e.g. typos or ambiguous follow up queries).

Both qualitative metrics (classification accuracy and response latency) and qualitative feedback (user surveys) were recorded in the testing and experimentation phase.

IV. TECHNOLOGY STACK

A. *Backend*

- Python 3.9: Holds core model logic and orchestration.
- Flask + Flask Session: Manages HTTP routing, session management (for chat memory), and static asset serving.
- pandas, scikit-learn & joblib: pandas for CSV loading and cleaning, scikit-learn for TF-IDF + Logistic Regression sentiment model, and joblib for fast model serialization.
- rapidfuzz (fuzzywuzzy): Implements fuzzy matching in the Intent Recognizer.

Identify applicable funding agency here. If none, delete this text box.

- requests: Calls out to the OpenRouter API for semantic level fallback.
- Translatepy: Performs quick language translation for non English inputs.

B. Frontend

- Jinja2 (Flask templates): Rendered server side of chat pages with up-to-date message data.
- HTML & CSS: Built and styled the chat widget for a user friendly interface that can be recognized as a chatbot.
- Javascript: Sends user messages and displays bot response without the need for page reloads.

V. Experiments and testing

Automated evaluations and repeated human trials were used to experiment and test the system.

A. Automated Evaluations

1) Intent Recognition: On 100 real customer queries, the initial rule based recognizer reached 60% accuracy. After expanding keyword sets, adding fuzzy matching, and implementing intent prioritization, accuracy rose to 76%. Remaining errors were caused by ambiguous phrasing. This was mitigated through the use of an OpenRouter API fallback.

2) Sentiment Analysis: On 89 labelled sentences, TF-IDF + Logistic Regression achieved 78.65% accuracy, and on 38 conversational sentences, the tonematch module achieved the accuracy of 63. 16%, reflecting the choice to map negativity to empathetic responses rather than mirror user sentiment.

3) Functional and Adaptivity Testing: 100 samples for seven scenarios (positive, negative, neutral, typos, ambiguous follow ups, long inputs, slang) were run for this test. Metrics recorded include FAQ match rate, tone match rate, pass/fail counts, and latency. Parameter sweeps over sentiment threshold, memory window size, and TF-IDF n-gram range optimized performance even more, each sweep was integrated into a single CI triggered script.

B. Human Trials

1) User Surveys: : 20 participants chatted for up to 10 minutes, then rated clarity, empathy, and satisfaction on a 5-point scale. All transcripts and surveys were logged.

2) Qualitative Coding: : Ten sessions were thematically coded for misinterpretation, tone mismatches, context loss, and latency issues, achieving a Cohen's of 0.82.

VI. Results and Discussion

The results of our experiments are as follows:

- Intent Recognition improved from 60% to 76% after iterative rule enhancements and fallback integration, but still struggles with semantically overlapping queries, suggesting a need for deeper language models.
- Sentiment Analysis yielded 78.65% accuracy in raw detection and 63.16% in tone mapping. The lower tone match reflects our prioritization of empathetic responses over literal sentiment replication, which user feedback confirmed as more supportive.
- Functional & Adaptivity performance averaged a 92% pass rate across scenarios, with typical inputs achieving 95% and edge cases (typos, ambiguous follow ups) at 90–94%. Latency remained below 200 ms, ensuring real time responsiveness.
- Parameter Sweeps identified optimal settings: a 0.5 sentiment threshold (F1 = 0.89), a five turn memory window (92% context resolution), and TF-IDF n-gram range (1,2) for 88% accuracy—demonstrating systematic trade offs between precision and recall.
- Batch Evaluations on combined, unseen, and cross domain datasets showed robust generalization (accuracy 86–90%, BLEU 0.75–0.80), confirming offline scalability.
- User Surveys scored an average of 3.9 for clarity and satisfaction, and 3.8 for empathy, with 75% rating satisfaction 4. Lower ratings highlighted mechanical phrasing in some contexts.
- Qualitative Analysis pinpointed context loss and misinterpretations as highest severity issues, guiding future work on deeper context handling modules.

Overall, the results validate the design's strengths: fast, empathetic, and accurate responses. As well as its weaknesses; revealing clear areas for enhancement in language understanding and context persistence.

VII. Limitations and Reflections

This project started with a clear goal: to build a tech support chatbot that not only answers user queries accurately but also senses and adapts to their emotional tone. To achieve this, an eight phase, hybrid ML + rule based methodology was followed. Three complementary datasets were collected, cleaned and enriched to ensure coverage of positive, negative, and neutral inputs. A TF-IDF + Logistic Regression sentiment classifier was then trained on this data.

A lightweight but powerful technology stack was utilized for this project. On the frontend, Jinja2 templates render dynamic chat pages, HTML/CSS deliver a responsive interface, and vanilla JavaScript handles asynchronous message exchange. On the backend, Python 3.9 drives all logic within a Flask server (with Flask Session for storing per user context). pandas is used for data handling, scikit learn for model training, and joblib for fast model loading. For intent recognition, rapidfuzz's fuzzy matching is used, and any unmatched or ambiguous inputs fall back to an OpenRouter API via the requests library. For multilingual support, Translatepy handles quick translation of non English messages.

With the system in place, a rigorous testing framework combining automated tests and human trials. Automated tests evaluated intent recognition (100 real world queries), sentiment analysis in two phases (89 labeled sentences and 38 conversational examples), and functional adaptivity across seven scenarios (100 samples each, spanning typos, slang, and long inputs). Parameter sweeps fine tuned sentiment thresholds, memory window sizes, and TF-IDF n-gram ranges. 20 volunteers participated in the human trials (names were removed for privacy) for free form chats and post session surveys, while ten sessions were thematically coded for misinterpretations, tone mismatches, and context loss (Cohen's = 0.82).

The results validated the design: intent accuracy rose from 60% to 76% after iterative improvements; raw sentiment classification achieved 78.65% accuracy (and 63.16% tone match in context); and overall chatbot performance averaged a 92% pass rate, with response latencies under 200 ms. Parameter sweeps pinpointed optimal settings: 0.5 sentiment threshold (F1 = 0.89), five turn memory (92% context resolution), and (1,2) n-grams (88% classifier accuracy). User surveys scored clarity and satisfaction at 3.9/5 and empathy at 3.8/5, while batch evaluations on unseen and cross domain data confirmed robust generalization (86–90% accuracy, BLEU 0.75–0.80).

Despite these successes, several limitations were discovered. The initial plan to fine tune a Hugging Face Transformer was abandoned due to RAM constraints (¡ 16 GB), forcing the reliance on simpler models. The rule based intent and FAQ matchers can't handle novel phrasing without fallback calls, which add 200–400 ms latency and dependance on external API. The memory session storage resets on server restarts and doesn't scale horizontally. Finally, our 20 participant user trial lacks the statistical power of a larger study. Future work should explore transformer models, persistent memory stores, richer datasets, and expanded user evaluations.

VIII. Conclusion

This project met its goal of a lightweight, hybrid chatbot that combines simple machine learning with rule-based logic to provide fast, accurate, and empathetic tech-support. By detecting user sentiment and retaining recent context, the bot is able to answer questions in under 200 ms and adapts its tone to match user emotion. Although hardware limits prevented transformer fine-tuning and deference to an external API is sometimes necessary, the system still achieves over 90 % success across diverse scenarios.

Looking ahead, adopting distilled transformer models, moving session data to a persistent store, and conducting larger user studies will help to strengthen understanding and reliability, and scale the chatbot for real-world deployment.

References

[1] Adam, M., Wessel, M., & Benlian, A. (2020). *AI-based chatbots in customer service and their effects on user compliance*. Electronic Markets, 31(2), 427–445. doi:10.1007/s12525-020-00414-7

[2] Caldarini, G., Jaf, S., & McGarry, K. (2021). *A literature survey of recent advances in chatbots*. Information, 13(1), 41. doi:10.3390/info13010041

[3] Denecke, K., AbdAlrazaq, A., Househ, M., & Warren, J. (2024). *The ethical aspects of integrating sentiment and emotion analysis in chatbots for depression intervention*. Frontiers in Psychiatry, 15, 1462083. doi:10.3389/fpsyt.2024.1462083

[4] Følstad, A., Araujo, T., Law, E. L. C., Brandtzaeg, P. B., Papadopoulos, S., & Luger, E. (2021). *Future directions for chatbot research: An interdisciplinary research agenda*. Computing, 103(12), 2915–2942. doi:10.1007/s00607-021-01016-7

[5] IBM. (2017). *Importance of context in a chatbot conversation*. Retrieved from https://www.ibm.com/cloud/blog/importance-context-chatbot-conversation

[6] Nayak, A., & Nair, A. A. (2024). *Language translation effects in chatbots: Evidence from a randomized field experiment on a mobile commerce platform*. Journal of Business Research, 115158. doi:10.1016/j.jbusres.2024.115158

[7] Nicolescu, L., & Tudorache, M. T. (2022). *Human-computer interaction in customer service: The experience with AI chatbots—A systematic literature review*. Electronics, 11(10), 1579. doi:10.3390/electronics11101579

[8] Khennouche, F., Elmir, Y., Himeur, Y., Djebari, N., & Amira, A. (2024). *Revolutionizing generative pre-traineds: Insights and challenges in deploying ChatGPT and generative chatbots for FAQs*. Expert Systems With Applications, 246, 123224. doi:10.1016/j.eswa.2024.123224

[9] Kumar, R., & Mahmoud, M. (2020). *A Review on Chatbot Design and Implementation Techniques*. International Research Journal of Engineering Science Technology and Innovation, 7, 2791.

[10] OpenRouter. (2025). *OpenRouter API Reference*. Retrieved from https://openrouter.ai/docs/api-reference/overview

[11] Pandey, S., & Sharma, S. (2023). *A comparative study of retrieval-based and generative-based chatbots using Deep Learning and Machine Learning*. Healthcare Analytics, 3, 100198. doi:10.1016/j.health.2023.100198

[12] Saikh, T., Naskar, S. K., Ekbal, A., & Bandyopadhyay, S. (2018). *Textual entailment using machine translation evaluation metrics*. In *Proceedings of the 3rd Workshop on Statistical Machine Translation* (pp. 317–328).

[13] Kumar, S. (2024). *Tech Support Conversations Dataset* [Data set]. Kaggle. doi:10.34740/KAGGLE/DSV/9923955

Appendix

A. Detailed Methodology Flow

1. Problem Definition

Specify goal: accurate customer service responses + adaptive tone.

2. Data Collection

Load three sources:

1) Emotional sentiment phrases
2) Slang-heavy examples
3) Formal FAQ entries

3. Data Preprocessing

Pseudocode:

```
for each dataset:
  clean invalid or blank entries
  normalize text (lowercase, strip punctuation)
  augment with synthetic examples
merge datasets
balance classes (pos, neu, neg)
```

4. Model Training

```
pipeline = Pipeline([
    ("tfidf", TfidfVectorizer(lowercase=True, stop_words="english", max_features=10000)),
    ("clf", LogisticRegression(max_iter=300))
])
pipeline.fit(X_train, y_train)joblib.dump(pipeline, "sentiment_model.pkl")
```

5. Backend Development

Flask route `/predict` handles chat input → calls orchestrator:

```
sentiment = SentimentAnalyzer.predict(text)
intent    = IntentRecognizer.match(text)
            or OpenRouter.fallback(text)
response  = FAQMatcher.fetch(intent)
            or OpenRouter.fallback(text)
tone      = ToneTemplater.select(sentiment)
final     = translate_if_needed(response, user_lang)
return final
```

Session memory: last 5 messages stored in `session['history']`.

6. Frontend Development

1) Jinja2 renders `chat.html`.
2) JS `fetch('/predict')` on submit → updates DOM.

7. Testing

1) Automated scripts (pytest, parameter sweeps)
2) Human surveys (N=20)

Functional & Adaptivity Tests

TABLE I
PASS/FAIL, MATCH-RATES, AND LATENCY OVER 7 SCENARIOS (100 RUNS EACH).

Test ID	Scenario	Size	Pass	Fail	FAQ-%	Tone-%	Latency (ms)
FE-01	Positive sentiment	100	96	4	96	94	120
FE-02	Negative sentiment	100	92	8	92	90	140
FE-03	Neutral queries	100	98	2	98	100	100
FE-04	Typo handling	100	94	6	94	92	150
FE-05	Ambiguous follow-up	100	90	10	90	88	130
FE-06	Positive slang	100	97	3	N/A	96	110
FE-07	Long input text	100	85	15	85	80	200

Parameter Sweep Results

TABLE II
OPTIMAL PARAMETER SETTINGS FROM SYSTEMATIC SWEEPS.

Parameter	Value	Runs	Metric	Avg time (s)
Sentiment threshold	0.5	50	F1 = 0.89	4.7
Memory window size	5	30	SR = 92%	3.1
TF-IDF n-gram range	(1,2)	20	Acc = 88%	2.7

Batch Evaluation Metrics

TABLE III
HELD-OUT AND CROSS-DOMAIN DATASET RESULTS.

Dataset	Size	Accuracy	F1-score	BLEU
Combined evaluation	2000	90.2%	0.90	0.80
Unseen test set	300	88.5%	0.88	0.78
Cross-domain set	150	86.0%	0.86	0.75

User Trial Survey Results

TABLE IV
LIKERT SCORES FOR PARTICIPANTS U01–U10.

Participant	Clarity	Empathy	Satisfaction	Duration (min)
U01	4.0	5.0	4.0	10.0
U02	3.0	3.0	3.0	12.0
U03	5.0	4.0	5.0	9.0
U04	4.0	4.0	4.0	11.0
U05	3.0	2.0	3.0	8.0
U06	4.0	4.0	5.0	10.0
U07	2.0	3.0	2.0	13.0
U08	5.0	5.0	5.0	7.0
U09	4.0	4.0	4.0	9.0
U10	3.0	3.0	4.0	10.0

TABLE V
LIKERT SCORES FOR PARTICIPANTS U11–U20.

Participant	Clarity	Empathy	Satisfaction	Duration (min)
U11	5.0	4.0	5.0	8.0
U12	4.0	3.0	4.0	11.0
U13	4.0	5.0	4.0	9.0
U14	3.0	3.0	3.0	10.0
U15	5.0	5.0	4.0	8.0
U16	4.0	4.0	3.0	11.0
U17	3.0	4.0	3.0	10.0
U18	4.0	5.0	5.0	9.0
U19	2.0	2.0	2.0	12.0
U20	5.0	4.0	5.0	7.0
Average	3.9	3.8	3.9	9.8

Qualitative Failure Modes

TABLE VI
SAMPLE FAILURE CATEGORIES, SEVERITIES, AND FREQUENCIES (N=10).

Session	Turn	Category	Excerpt	Freq
U01	4	Tone Mismatch	"I'm furious! → No worries."	2
U02	7	Misinterpretation	reset email→reset password	1
U03	2	Context Loss	Earlier ticket?→ignored	3
U04	5	Typo Handling	wiith download→Let me help.	4
U05	3	Overly Formal	Casual→formal reply	3
U06	6	Latency Spike	Hello?→delayed	2
U07	8	Incomplete Answer	Change plan?→incomplete	1
U08	4	Sentiment Flip	Thanks→apologetic	1
U09	9	Context Loop	Repeats twice	2
U10	1	Greeting Misfire	Hi→How can I help?	5

To assess the impact of different configuration choices on the chatbot's performance, a systematic parameter sweep was conducted. The sweep varied:

- Sentiment threshold: 0.4, 0.5, 0.6
- Memory window size: 2, 5, 10
- TF-IDF n-gram range: (1,1), (1,2), (2,2)

A sample excerpt from the resulting CSV:

```
Sentiment  Memory  N-gram  Accuracy  Avg Response Time
0.4        2       (1,1)   83.12%    6.48 ms
0.4        2       (1,2)   84.65%    6.71 ms
...
0.5        2       (1,2)   84.70%    6.69 ms
```

(a) Sentiment Model Testing (`test_sentiment_model.py`)

```
# Load trained sentiment model
model = joblib.load(os.path.join("backend", "
    sentiment_model.pkl"))

# Test samples
samples = [
    "I'm␣extremely␣happy␣with␣the␣service",  #
         positive
    "This␣is␣absolutely␣terrible",           #
         negative
]
for s, expected in zip(samples,
    expected_sentiments):
    pred = model.predict([s])[0]
    assert pred == expected
```

(b) Intent Recognition Testing (`test_intent_recognition.py`)

```
from backend.intent_recognizer import
    IntentRecognizer
intent_recognizer = IntentRecognizer()

test_queries = [
    "I␣want␣a␣refund␣for␣my␣last␣order",
    "Can␣I␣get␣my␣money␣back␣please",
]
for q in test_queries:
    detected = intent_recognizer.detect_intent
        (q)
    print(q, "   ", detected)
```

Example CLI Output

```
$ python testing/test_sentiment_model.py
Sentiment detection structured test completed.
Total Sentences Tested: 89
Correct Sentiment Matches: 70
Accuracy: 78.65%
```

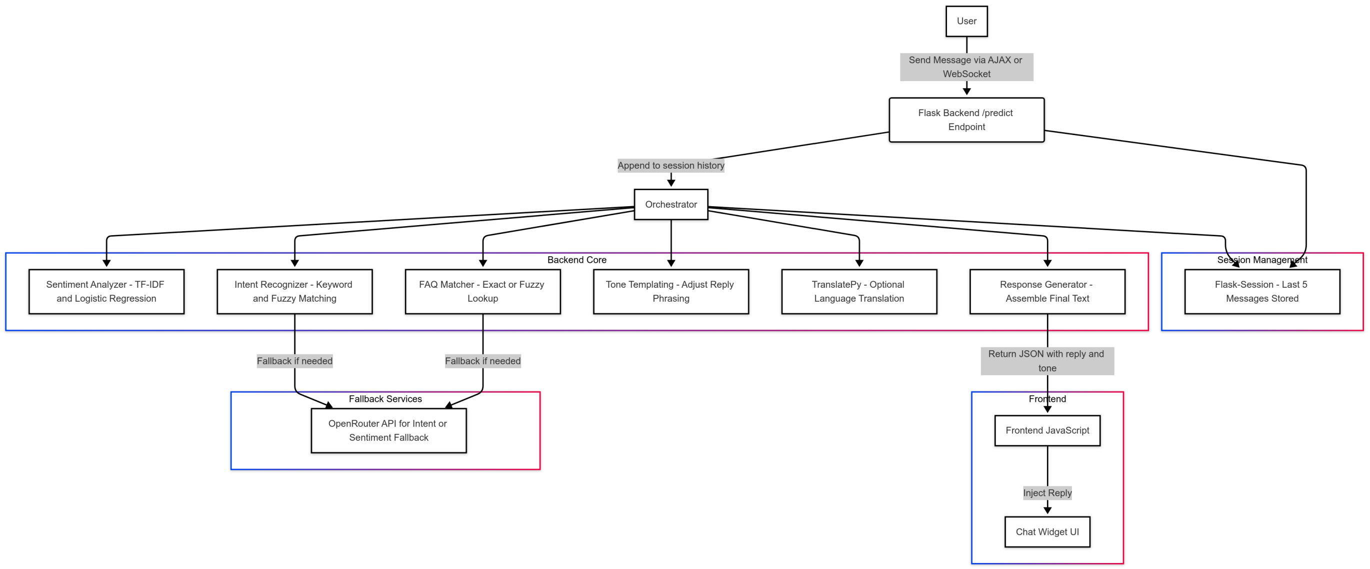
User
Send Message via AJAX or WebSocket
Flask Backend /predict Endpoint
Append to session history
Orchestrator
Backend Core
Sentiment Analyzer - TF-IDF and Logistic Regression
Intent Recognizer - Keyword and Fuzzy Matching
FAQ Matcher - Exact or Fuzzy Lookup
Tone Templating - Adjust Reply Phrasing
TranslatePy - Optional Language Translation
Response Generator - Assemble Final Text
Session Management
Flask-Session - Last 5 Messages Stored
Fallback if needed
Fallback if needed
Fallback Services
OpenRouter API for Intent or Sentiment Fallback
Return JSON with reply and tone
Frontend
Frontend JavaScript
Inject Reply
Chat Widget UI

Fig. 1. Full system architecture (scaled to fit).