

Project Report: Mental Health Support Chatbot for Students

Submitted By:

Vureti Praveen

12309813

Btech(CSE)

Lovely Professional University

[Date: October 02, 2025]

Abstract

This project presents an AI-powered mental health support chatbot designed specifically for students facing stress, anxiety, and other emotional challenges. Built using Python with Streamlit for the interface, NLTK for natural language processing, and SQLite for data storage, the chatbot provides context-aware responses, emotion detection, crisis intervention, and mood tracking. The system emphasizes privacy, empathy, and actionable coping strategies. Key features include sentiment analysis, targeted remedies for stressors like exams and work, and visual mood trends. This report covers the architecture, implementation, setup, troubleshooting, and deployment, demonstrating practical application of NLP, web development, and database management.

Table of Contents

1. Introduction
2. Overall Architecture
3. Key Components and Code Explanation
4. Setup Guide
5. Troubleshooting
6. Deployment Options
7. Features and Usage
8. Technology Stack
9. Limitations and Future Extensions
10. Conclusion
11. References
12. Appendices (Code Snippets, Requirements)

1. Introduction

Mental health issues among students are increasingly prevalent, with stressors like exams, work, and relationships contributing significantly. This chatbot serves as a supportive tool, offering immediate, empathetic responses without replacing professional care. It uses AI to detect emotions, analyze sentiment, and provide tailored advice, while storing interactions locally for privacy.

The project draws from real-world needs, incorporating crisis detection (e.g., suicide keywords) to refer users to hotlines like 988. It's a learning-focused implementation, showcasing Python's versatility in AI and web apps.

2. Overall Architecture

The chatbot follows a modular data flow:

...

User Input → NLP Processing → Context Analysis → Response Generation → Database Storage → UI Update

...

Data Flow Steps:

1. **User Input**: Captured via Streamlit text input.
2. **NLP Processing**: Sentiment analysis (VADER) and keyword-based emotion/stressor detection.
3. **Context Analysis**: Reviews conversation history to avoid repetition and build continuity.
4. **Response Generation**: Selects from templates based on priorities (crisis first, then stressors, emotions).
5. **Database Storage**: Saves interactions in SQLite for mood tracking.
6. **UI Update**: Displays responses and updates charts in real-time.

This architecture ensures natural, progressive conversations while prioritizing user safety.

3. Key Components and Code Explanation

3.1 Natural Language Processing (NLP)

- **VADER Sentiment Analysis**: Uses NLTK's SentimentIntensityAnalyzer to score text from -1 (negative) to +1 (positive). Handles slang and emoticons.

```
```python
sia = SentimentIntensityAnalyzer()
scores = sia.polarity_scores(message)
```
```

Example: "I'm really happy!" → +0.6; "I'm not feeling good" → -0.3.

- **Emotion Detection**: Keyword matching for emotions like stress, anxiety.

```
```python
emotions = {
 'stress': ['stress', 'stressed', 'overwhelmed', 'pressure'],
 'anxiety': ['anxious', 'worried', 'nervous', 'panic']
}
```
```

Converts message to lowercase and checks for matches.

3.2 Context Awareness

Analyzes history to track topics, emotions, and questions asked.

```
```python
```

```
def analyze_conversation_context(conversation_history):
```

```
 # Tracks discussed topics, prevents repetition
```

```
```
```

Remembers stressors (e.g., exams) for follow-up.

3.3 Response Generation

Prioritizes: Crisis → Stressors → Emotions → General.

- **Crisis Detection**:

```
```python
```

```
CRISIS_KEYWORDS = ["suicide", "kill myself", "end it all"]
```

```
```
```

Overrides other logic to provide resources.

- **Stressor-Specific Responses**:

```
```python
```

```
if primary_stressor == 'exam_anxiety':
```

```
 # Provide study strategies
```

```
```
```

- **Random Selection**: Uses `random.choice()` for variety.

3.4 Database Design

SQLite schema:

```
```sql
```

```
CREATE TABLE mood_tracking (
 id INTEGER PRIMARY KEY AUTOINCREMENT,
 timestamp TEXT,
 user_message TEXT,
 bot_response TEXT,
 sentiment REAL,
 conversation_context TEXT
);
```
```

Lightweight and local; stores history for trends.

3.5 User Interface (Streamlit)

Manages session state for persistence:

```
```python  
if 'chat_history' not in st.session_state:
 st.session_state.chat_history = []
```
```

Updates UI reactively with charts via Matplotlib.

3.6 Technical Concepts

- **Keyword Matching**: Simple and efficient, but limited (e.g., misses sarcasm).
- **Data Visualization**: Plots sentiment over time.

```
```python
```

```
fig, ax = plt.subplots(figsize=(8, 4))
```

```
ax.plot(df['timestamp'], df['sentiment'], marker='o')
```

```
```
```

4. Setup Guide

System Requirements:

- Python 3.8+
- Git
- Code editor (e.g., VS Code)

Steps:

1. Clone/Download project.
2. Create virtual environment: ``python3 -m venv venv`` and activate.
3. Install dependencies: ``pip install -r requirements.txt`` (includes streamlit, nltk, etc.).
4. Run: ``streamlit run app.py``.
5. Access at <http://localhost:8501>.

Test with messages like "I'm stressed about exams."

5. Troubleshooting

Common issues:

- ****pip not found****: Install via brew/apt/python.org.
- ****ModuleNotFoundError****: Activate venv and reinstall requirements.
- ****Database errors****: Delete `chatbot.db` and restart.
- ****App slow****: Clear history or restart server.
- ****NLTK errors****: Download VADER lexicon manually.

Performance tips: Use 2GB+ RAM; update dependencies.

6. Deployment Options

Streamlit Cloud (Recommended)

Push to GitHub, deploy via share.streamlit.io.

Heroku

Create Procfile: ``web: streamlit run app.py --server.port=$PORT --server.address=0.0.0.0``.

Deploy with Heroku CLI.

Docker

Dockerfile example:

```
``dockerfile
```

```
FROM python:3.9-slim
```

```
COPY requirements.txt .
```

```
RUN pip install -r requirements.txt
```

```
COPY . .
```

```
CMD ["streamlit", "run", "app.py", "--server.address=0.0.0.0"]
```

```
```
```

Other options: GCP, AWS, Azure, or self-hosted with nginx.

Security: Use env variables; enable HTTPS.

## 7. Features and Usage

- **Intelligent Conversation**: Context-aware, emotion-based responses.
- **Mood Tracking**: Charts sentiment trends.
- **Targeted Support**: Remedies for exam anxiety, work stress, etc.
- **Crisis Resources**: Hotlines like 741741.

Usage: Type concerns; view sidebar for trends and resources.

## 8. Technology Stack

- Frontend: Streamlit
- Backend: Python 3.8+
- NLP: NLTK (VADER)
- Database: SQLite
- Visualization: Matplotlib, Pandas
- Other: re, random, datetime

Requirements:

...

streamlit

nltk

scikit-learn

matplotlib

pandas

...

## 9. Limitations and Future Extensions

Limitations: Basic keyword matching (no sarcasm detection); local-only storage.

Extensions:

- Easy: Add emotions/strategies.
- Medium: Voice input, exports.
- Advanced: ML models, multi-language, APIs.

Learning: Python basics to advanced NLP.

## 10. Conclusion

This chatbot demonstrates AI's potential in mental health support, blending empathy with technology. It's a proof-of-concept for student wellness tools, emphasizing ethical use and professional referrals.

# 11. References

- NLTK Documentation: <https://www.nltk.org>
- Streamlit Docs: <https://docs.streamlit.io>
- Crisis Resources: [NAMI.org](https://www.nami.org), [MHANational.org](https://www.mhanational.org)
- Project Files: `app.py`, `README.md`, etc.

# 12. Appendices

## Appendix A: Key Code Snippets

(Insert relevant snippets from `app.py`, e.g., imports, functions.)

## Appendix B: Project Structure

...

`mental_health_chatbot/`

├─ `app.py`

├─ `requirements.txt`

├─ `README.md`

├─ `SETUP_GUIDE.md`

├─ `TROUBLESHOOTING.md`

├─ `DEPLOYMENT.md`

├─ `CODE_EXPLANATION.md`

└─ `chatbot.db` (generated)

...