🛡️ Technical Report: AI Safety Models POC

1. Introduction

This technical report summarizes the design and implementation of a Proof of Concept (POC) for a suite of AI Safety Models aimed at enhancing user safety in a conversational AI environment. The system focuses on detecting harmful, dangerous, or inappropriate messages in real-time, with a foundation that supports ethical and scalable ML development.

---

2. High-Level Design Decisions

2.1 Multi-label Classification

We opted for a **multi-label classification model** because messages can contain multiple safety risks simultaneously (e.g., a message may be both abusive and age-inappropriate). A `MultiOutputClassifier` wrapper around a logistic regression model was chosen for simplicity and interpretability in the POC phase.

2.2 FastAPI for Integration

FastAPI was chosen as the RESTful API framework for:

- Rapid prototyping
- Strong typing with Pydantic
- Native interactive docs (Swagger UI)
- High performance for real-time detection

2.3 Modular Architecture

The project was structured with separate modules for:

- Data and training (`train_safety_model.py`)
- Inference (`main.py`)
- Evaluation (`evaluate_model.py`)
  This separation supports team collaboration, CI/CD workflows, and easy scaling.

---

3. Data Sources & Preprocessing

3.1 Simulated Dataset

Since publicly labeled multi-label safety datasets are rare, we created a **simulated dataset** (`safety_data.csv`) with ~10 rows for demonstration. The dataset includes:

- `text`: user input messages
- `abuse`: flag for harmful language
- `escalation`: signs of increasing aggression
- `crisis`: signs of emotional distress
- `age_inappropriate`: content unsuitable for minors

Text Example	***abuse***	***escalation***	***crisis***	***age_inappropriate***
"You're a loser!"	1	1	0	0
"I'm thinking about ending it all"	0	0	1	0

3.2 Preprocessing

- Text is vectorized using `TfidfVectorizer` with English stopword removal.
- Lowercasing is applied.
- No stemming or lemmatization was performed at this stage for simplicity.

---

4. Model Architecture & Training

4.1 Pipeline

The model uses the following scikit-learn pipeline:

```
TfidfVectorizer → MultiOutputClassifier(LogisticRegression)
```

4.2 Training

- **Train/test split:** 80/20
- **Classifier:** Logistic Regression with default hyperparameters
- **Labels:** `['abuse', 'escalation', 'crisis', 'age_inappropriate']`
- **Training time:** ~1 second on CPU

4.3 Why Logistic Regression?

- Fast to train and infer
- Well-suited for interpretable prototypes
- Easily swappable with more advanced models (e.g., BERT) later

---

5. Evaluation & Metrics

5.1 Script

The evaluation is handled by `evaluate_model.py`, which computes:

- Per-label **precision**, **recall**, **F1-score**
- **Label-wise accuracy**
- **Exact match ratio** (all labels correct per message)

5.2 Sample Output

```
Classification Report (per label):

                  precision    recall  f1-score
           abuse       1.00      1.00      1.00
      escalation       1.00      0.50      0.67
          crisis       1.00      1.00      1.00
age_inappropriate       1.00      1.00      1.00

Exact Match Ratio: 0.80
```

---

6. Leadership Considerations & Iteration Plan

As a technical leader guiding a team through this POC and toward a production-ready system, the approach would involve:

6.1 Roadmapping & Milestones

- **MVP Phase**: Simple interpretable models (e.g., logistic regression)
- **Phase 2**: Advanced NLP (transformers), real-world datasets, human-in-the-loop
- **Phase 3**: Deployment (Docker, cloud hosting, CI/CD, alert systems)

6.2 Team Roles

- **ML Engineer**: Focus on model performance and evaluation
- **Data Annotator/Analyst**: Help build and label real datasets
- **Backend Developer**: Integrate with chat systems or moderation dashboards
- **Product/UX**: Collaborate on user experience for flagged content

6.3 Iteration Strategy

- Weekly sprints with demos
- Ethical review checkpoints (e.g., for false positives in crisis detection)
- Transparent metric reporting

6.4 Collaboration

- Use GitHub or GitLab for version control
- Use issues/PRs for structured code reviews
- Encourage unit testing and modular code to support scaling and reuse

---

7. Conclusion

This POC demonstrates a working prototype for real-time AI safety moderation, addressing four key safety categories. The system is lightweight, interpretable, and serves as a foundation for expanding into more complex and production-grade solutions.

With proper data, ethical oversight, and iteration, it can be evolved into a robust real-world safety net for conversational AI platforms.

---

8. Appendix: Files in the Repository

File	Purpose
`train_safety_model.py`	Train multi-label safety model
`main.py`	FastAPI server for real-time inference
`evaluate_model.py`	Model performance evaluation
`data/safety_data.csv`	Simulated dataset
`requirements.txt`	Python dependencies

`README.md`	Project documentation