Code Explanation: Misinformation Detection System

1. Overview

This system detects misinformation by analyzing contradictions between claims, temporal patterns, and contextual integrity using transformer-based models (DeBERTa). It consists of three main components:

-ContradictionMatrix: Tracks contradictions between claims

- IntegrityAnalyzer: Performs deep analysis of claim integrity

- EnhancedMisinfoDetector: Combines multiple detection methods

2. ContradictionMatrix Class

Purpose

Tracks relationships between claims and identifies contradictions within specific time windows.

Key Components

| Component | Description |

`matrix` Nested dictionary storing contradiction scores between claim pairs

`claim_db` Database storing claim metadata (text, source, timestamp)

Key Methods

Method | Functionality

`add_claim()` Stores new claims in the database

`update_contradictions()` Updates contradiction scores bidirectionally

`get_temporal_contradictions()` Finds contradictions within a 30-day window

3. IntegrityAnalyzer Class

Purpose

Performs NLP-based integrity analysis using DeBERTa models.

Initialization

Loads two DeBERTa models:

1. Contradiction Detection (`deberta-large-mnli`)

2. Text Embeddings (`deberta-base`)

Key Methods**

| Method | Functionality |

|--------|--------------|

| `encode_claim()` | Generates embeddings for semantic analysis |

| `detect_contradiction()` | Computes contradiction probability between two claims |

| `analyze_claim()` | Main analysis pipeline calculating integrity scores |

Analysis Process

1. Temporal Analysis: Checks for recent contradictions

2. Cross-Claim Comparison: Scores contradictions against all known claims

3. Integrity Scoring: Combines base score with temporal decay factor

4. EnhancedMisinfoDetector Class

Purpose

Orchestrates multi-faceted misinformation detection.

Key Features

1. Integrity Analysis (60% weight)

2. Deception Pattern Detection (40% weight)

3. Temporal Consistency Checks

Composite Scoring

python

```
composite_score = 0.6 * integrity_score + 0.4 * (1 - deception_score)
```

5. Example Usage

python

```
sample_claims = [
  {
    "id": "claim1",
    "text": "COVID vaccines are 95% effective",
    "source": "WHO",
```

```
    "timestamp": datetime(2023, 1, 1)
  },
  # ... (additional claims)
]

detector = EnhancedMisinfoDetector()
for claim in sample_claims:
  results = detector.analyze_claim(claim)
```

Output Includes

- Integrity score (0-1)
- Top contradictions
- Temporal patterns
- Composite risk score

6. Technical Dependencies

Library	Purpose
PyTorch	Deep Learning backend
Transformers	NLP model access
NumPy	Numerical operations
NetworkX	Context graph management

7. Key Parameters

Parameter	Value	Purpose
`contradiction_threshold`	0.8	Minimum score for contradiction flag
`temporal_decay`	0.1/month	Reduces impact of older contradictions

After running app.py

open http://localhost:5000/ on browser to input the test contradictions

Appendix:

analyzer.py

```
import torch
import numpy as np
from datetime import datetime, timedelta
from collections import defaultdict
from transformers import AutoModelForSequenceClassification, AutoTokenizer

class ContradictionMatrix:
    def __init__(self):
        self.matrix = defaultdict(dict)  # claim_id -> {other_claim_id: contradiction_score}
        self.claim_db = {}  # claim_id -> claim_data

    def add_claim(self, claim_id, claim_text, source, timestamp):
        self.claim_db[claim_id] = {
            "text": claim_text,
            "source": source,
            "timestamp": timestamp,
            "embeddings": None
        }

    def update_contradictions(self, claim_id, other_claim_id, score):
        self.matrix[claim_id][other_claim_id] = score
        self.matrix[other_claim_id][claim_id] = score
```

```
    def get_temporal_contradictions(self, claim_id, time_window=30):
        target_claim = self.claim_db[claim_id]
        contradictions = []

        for other_id, other_claim in self.claim_db.items():
            if other_id == claim_id:
                continue

            time_diff = (target_claim["timestamp"] - other_claim["timestamp"]).days
            if abs(time_diff) <= time_window:
                score = self.matrix[claim_id].get(other_id, 0)
                if score > 0.7:  # High contradiction threshold
                    contradictions.append({
                        "claim_id": other_id,
                        "text": other_claim["text"],
                        "source": other_claim["source"],
                        "time_diff_days": time_diff,
                        "score": score
                    })

        return sorted(contradictions, key=lambda x: x["score"], reverse=True)

class IntegrityAnalyzer:
    def __init__(self):
        # Load DeBERTa models
        self.contradiction_model_name = "microsoft/deberta-large-mnli"
        self.contradiction_tokenizer =
AutoTokenizer.from_pretrained(self.contradiction_model_name)
        self.contradiction_model =
AutoModelForSequenceClassification.from_pretrained(self.contradiction_model_name)
```

```
        self.embedding_model_name = "microsoft/deberta-base"
        self.embedding_tokenizer = AutoTokenizer.from_pretrained(self.embedding_model_name)

        # Initialize matrices
        self.contradiction_matrix = ContradictionMatrix()
        self.context_graph = nx.Graph()

        # Thresholds
        self.contradiction_threshold = 0.8
        self.temporal_decay = 0.1  # Per month

    def encode_claim(self, text):
        inputs = self.embedding_tokenizer(text, return_tensors="pt", padding=True, truncation=True,
max_length=512)
        with torch.no_grad():
            outputs = self.contradiction_model(**inputs, output_hidden_states=True)
        return outputs.hidden_states[-1][:,0,:].cpu().numpy()

    def detect_contradiction(self, claim1, claim2):
        inputs = self.contradiction_tokenizer(claim1, claim2, return_tensors="pt", padding=True,
truncation=True)
        with torch.no_grad():
            outputs = self.contradiction_model(**inputs)
        probs = torch.softmax(outputs.logits, dim=-1)
        return probs[0][2].item()  # Contradiction probability

    def analyze_claim(self, claim_data):
        claim_id = claim_data["id"]
        self.contradiction_matrix.add_claim(
            claim_id,
            claim_data["text"],
            claim_data["source"],
```

```
        claim_data["timestamp"]
    )

    # Temporal-contextual analysis
    temporal_contradictions = self.contradiction_matrix.get_temporal_contradictions(claim_id)

    # Cross-claim contradiction analysis
    contradiction_scores = []
    for other_id, other_claim in self.contradiction_matrix.claim_db.items():
        if other_id == claim_id:
            continue

        score = self.detect_contradiction(claim_data["text"], other_claim["text"])
        self.contradiction_matrix.update_contradictions(claim_id, other_id, score)
        contradiction_scores.append(score)

    # Integrity score calculation
    base_integrity = 1.0 - max(contradiction_scores) if contradiction_scores else 1.0
    temporal_factor = np.exp(-self.temporal_decay *
                             len([c for c in temporal_contradictions if c["score"] > 0.7]))
    integrity_score = base_integrity * temporal_factor

    return {
        "claim_id": claim_id,
        "integrity_score": float(integrity_score),
        "contradiction_score": float(max(contradiction_scores)) if contradiction_scores else 0.0,
        "temporal_contradictions": temporal_contradictions,
        "contextual_similarity": self._get_contextual_similarity(claim_data["text"])
    }

def _get_contextual_similarity(self, text):
```

```
        # Implement contextual similarity using DeBERTa embeddings
        pass

class EnhancedMisinfoDetector:
    def __init__(self):
        self.integrity_analyzer = IntegrityAnalyzer()
        self.claim_history = []

    def analyze_claim(self, claim_data):
        # Integrity analysis
        integrity_results = self.integrity_analyzer.analyze_claim(claim_data)

        # Deception pattern detection
        deception_results = self._detect_deception(claim_data["text"])

        # Temporal consistency
        temporal_results = self._check_temporal_consistency(claim_data)

        return {
            **integrity_results,
            **deception_results,
            **temporal_results,
            "composite_score": self._calculate_composite_score(integrity_results, deception_results)
        }

    def _detect_deception(self, text):
        # Existing deception detection logic
        pass

    def _check_temporal_consistency(self, claim_data):
        # Check against historical claims
```

```
        pass

    def _calculate_composite_score(self, integrity, deception):
        # Weighted combination of scores
        return 0.6 * integrity["integrity_score"] + 0.4 * (1 - deception["deception_score"])

# Example Usage
if __name__ == "__main__":
    detector = EnhancedMisinfoDetector()

    sample_claims = [
        {
            "id": "claim1",
            "text": "COVID vaccines are 95% effective",
            "source": "WHO",
            "timestamp": datetime(2023, 1, 1)
        },
        {
            "id": "claim2",
            "text": "Vaccines cause severe side effects in most people",
            "source": "AntiVaxBlog",
            "timestamp": datetime(2023, 1, 15)
        }
    ]

    for claim in sample_claims:
        results = detector.analyze_claim(claim)
        print(f"Analysis for claim {claim['id']}:")
        print(json.dumps(results, indent=2))
        print("\n" + "="*80 + "\n")
```

app.py

```
from flask import Flask, render_template, request, jsonify
from analyzer import MisDisInfoDetector
from datetime import datetime, timedelta
import json

app = Flask(__name__)
detector = MisDisInfoDetector()

KNOWN_FACTS = [
  "Clinical trials show the vaccine has mild side effects in less than 10% of recipients",
  "The vaccine has been approved by major health organizations"
]

@app.route('/')
def index():
  return render_template('index.html')

@app.route('/analyze', methods=['POST'])
def analyze():
  data = request.json
  claim_data = {
    "text": data['claim'],
    "source": data.get('source', 'unknown'),
    "timestamp": datetime.now(),
    "context": data.get('context', '')
  }

  results = detector.analyze_claim(claim_data, known_facts=KNOWN_FACTS)
```

```
    return jsonify(results)

if __name__ == '__main__':
    app.run(debug=True)
```

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Misinformation Detector</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <h1>Information Integrity Analyzer</h1>
    <form id="claimForm">
        <div>
            <label for="claim">Enter your claim:</label>
            <textarea id="claim" required></textarea>
        </div>
        <div>
            <label for="source">Source (optional):</label>
            <input type="text" id="source">
        </div>
        <button type="submit">Analyze</button>
    </form>

    <div id="results" style="display: none;">
        <h2>Analysis Results</h2>
        <div id="classification"></div>
```

```
    <div id="explanation"></div>

    <div class="score">
      <div>Factuality Score: <span id="factualityScore"></span></div>
      <div class="bar"><div class="fill" id="factualityBar"></div></div>
    </div>

    <div class="score">
      <div>Intent Score: <span id="intentScore"></span></div>
      <div class="bar"><div class="fill" id="intentBar"></div></div>
    </div>

    <div class="score">
      <div>Composite Score: <span id="compositeScore"></span></div>
      <div class="bar"><div class="fill" id="compositeBar"></div></div>
    </div>
  </div>

  <script>
    document.getElementById('claimForm').addEventListener('submit', async function(e) {
      e.preventDefault();

      const claim = document.getElementById('claim').value;
      const source = document.getElementById('source').value;

      const response = await fetch('/analyze', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ claim, source })
      });
```

```
        const results = await response.json();
        displayResults(results);
    });

    function displayResults(data) {
        document.getElementById('results').style.display = 'block';
        document.getElementById('classification').innerHTML =
            `<strong>Classification:</strong> ${data.classification} (confidence:
${(data.confidence * 100).toFixed(1)}%)`;

        document.getElementById('factualityScore').textContent =
            data.factuality_score ? data.factuality_score.toFixed(2) : 'N/A';
        document.getElementById('intentScore').textContent = data.intent_score.toFixed(2);
        document.getElementById('compositeScore').textContent =
data.composite_score.toFixed(2);

        if (data.factuality_score) {
            document.getElementById('factualityBar').style.width = `${data.factuality_score *
100}%`;
        }
        document.getElementById('intentBar').style.width = `${data.intent_score * 100}%`;
        document.getElementById('compositeBar').style.width = `${data.composite_score *
100}%`;

        // Simple explanation based on classification
        let explanation = "";
        if (data.classification === "ACCURATE_INFORMATION") {
            explanation = "This information appears to be accurate and reliable.";
        } else if (data.classification === "MISINFORMATION") {
            explanation = "This appears to be false information, but may not be intentionally
deceptive.";
        } else if (data.classification === "DISINFORMATION") {
            explanation = "This appears to be intentionally deceptive information.";
```

```
        } else {
            explanation = "The system cannot determine the accuracy of this information with confidence.";
        }

        document.getElementById('explanation').innerHTML = `<p>${explanation}</p>`;
    }
  </script>
</body>
</html>
```

Styles.css

```
/* styles.css - Futuristic Theme */
:root {
  --primary: #00ffcc;  /* Cyber teal */
  --secondary: #0066ff; /* Neon blue */
  --dark: #0a0a1a;    /* Deep space blue */
  --light: #e0e0e0;   /* Bright gray */
  --accent: #ff00aa;  /* Electric pink */
}

body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background-color: var(--dark);
  color: var(--light);
  margin: 0;
  padding: 0;
  line-height: 1.6;
```

```
}

h1, h2, h3 {
  color: var(--primary);
  text-shadow: 0 0 5px rgba(0, 255, 204, 0.3);
  letter-spacing: 1px;
}

h1 {
  border-bottom: 2px solid var(--primary);
  padding-bottom: 10px;
  font-size: 2.5rem;
}

#container {
  max-width: 800px;
  margin: 0 auto;
  padding: 20px;
}

/* Form Elements */
#claimForm {
  background: rgba(10, 10, 26, 0.7);
  padding: 25px;
  border-radius: 8px;
  border: 1px solid var(--secondary);
  box-shadow: 0 0 15px rgba(0, 102, 255, 0.2);
}

label {
  display: block;
```

```
  margin-bottom: 8px;
  color: var(--primary);
  font-weight: bold;
}

input, textarea {
  width: 100%;
  padding: 12px;
  margin-bottom: 20px;
  background: rgba(255, 255, 255, 0.1);
  border: 1px solid var(--secondary);
  border-radius: 4px;
  color: var(--light);
  font-size: 16px;
}

input:focus, textarea:focus {
  outline: none;
  border-color: var(--primary);
  box-shadow: 0 0 8px rgba(0, 255, 204, 0.4);
}

button {
  background: linear-gradient(135deg, var(--primary), var(--secondary));
  color: var(--dark);
  padding: 12px 25px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  font-size: 16px;
  font-weight: bold;
```

```
    text-transform: uppercase;
    letter-spacing: 1px;
    transition: all 0.3s ease;
}

button:hover {
    transform: translateY(-2px);
    box-shadow: 0 5px 15px rgba(0, 255, 204, 0.4);
}

/* Results Section */
#results {
    background: rgba(10, 10, 26, 0.7);
    padding: 25px;
    border-radius: 8px;
    border: 1px solid var(--secondary);
    box-shadow: 0 0 15px rgba(0, 102, 255, 0.2);
    margin-top: 30px;
}

.score {
    margin: 20px 0;
}

.bar {
    height: 20px;
    background: rgba(255, 255, 255, 0.1);
    border-radius: 10px;
    overflow: hidden;
    margin: 10px 0;
}
```

```
.fill {
  height: 100%;
  background: linear-gradient(90deg, var(--primary), var(--accent));
  transition: width 0.5s ease;
}

/* Glow Effects */
.glow {
  animation: glow 2s infinite alternate;
}

@keyframes glow {
  from {
    box-shadow: 0 0 5px rgba(0, 255, 204, 0.5);
  }
  to {
    box-shadow: 0 0 20px rgba(0, 255, 204, 0.8);
  }
}

/* Responsive Design */
@media (max-width: 768px) {
  #container {
    padding: 15px;
  }

  h1 {
    font-size: 2rem;
  }
}
```