Knowledge Graph-Based Therapeutic Chatbot: Detailed Analysis Process

A sophisticated system that maps user messages to clinical entities through pattern detection and provides therapeutic responses based on a comprehensive knowledge graph architecture.

by Amr Shaarawy

Pattern Detection & Mapping Architecture

The system employs a sophisticated pipeline to map user messages to clinical entities:

Natural Language Input Processing

- User messages arrive as unstructured text
- Context aggregation combines recent conversation history (up to 5 previous messages)
- The complete context is provided to the Gemma 3-27B-IT model for analysis

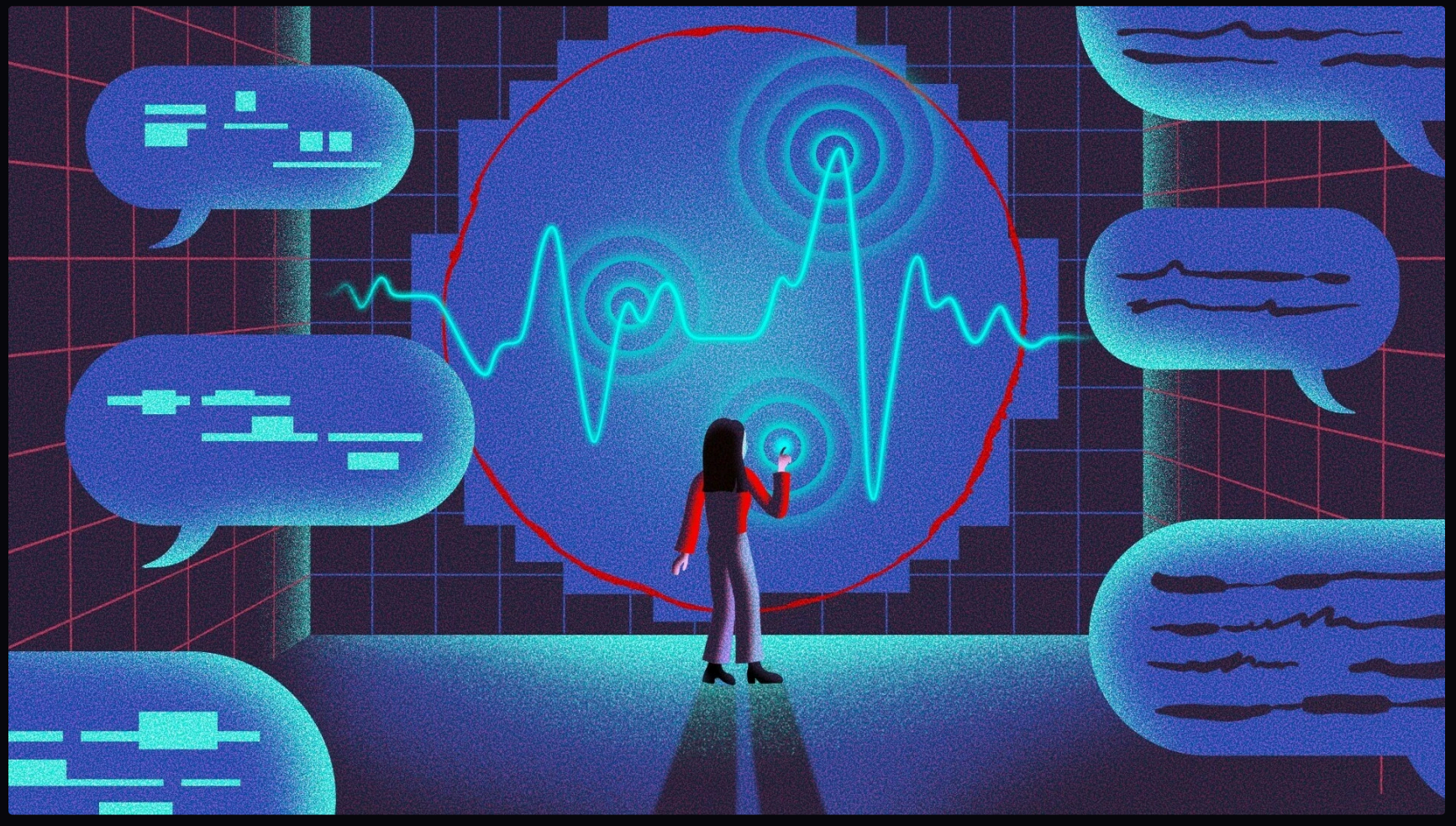
Clinical Entity Recognition

The system sends a specialized prompt to Gemma with:

- Clear instructions for mental health pattern detection
- Differentiation criteria for similar conditions
- Examples of typical symptom presentations
- The full conversation context and latest message

Gemma analyzes text for indicators of:

- **8 predefined symptoms**: Depression, Anxiety, Excessive Sleepiness, Shortness of breath, Panic attacks, Dizziness, Chest pain, Fear of losing control
- **14 predefined behaviors**: Dissociative reaction, Hypervigilance, Exaggerated startle response, More talkative than usual, Delusions, Fatigue, Inflated self-esteem, Disorganized thinking or speech, Angry outburst, Recklessness, Concentration issues, Excessive worry or fear, Irritability, Diminished interest
- Weight conditions and substance use patterns



Symptom & Behavior Mapping Pipeline

Pattern Extraction & JSON Structure

The LLM returns a structured JSON with four categories:

```
{ "symptoms": ["Depression", "Anxiety"], "behaviors": ["Fatigue", "Diminished interest"], "weight_conditions": ["Overweight"], "substance_use": ["Alcohol"] }
```

JSON parsing logic handles potential formatting issues:

- Removes markdown code blocks if present
- Processes the clean JSON string
- Handles exceptions with empty default values

Contextual Pattern Intelligence

Special detection rules distinguish similar presentations:

- Depression vs. Bipolar (looks for mania indicators)
- Anxiety vs. Panic (differentiates general anxiety from acute episodes)
- PTSD vs. Anxiety (identifies trauma-specific responses)

Example differentiation (from code):

> # Depression requires depressive symptoms and no manic symptoms WHEN d.name = 'Major Depressive Disorder' AND ('Depression' IN matching_symptoms OR 'Fatigue' IN matching_behaviors OR 'Diminished interest' IN matching_behaviors) AND NOT ('More talkative than usual' IN matching_behaviors OR 'Inflated self-esteem' IN matching_behaviors OR 'Recklessness' IN matching_behaviors) THEN 10 // Boost depression score if typical depression symptoms without manic symptoms

Advanced Pattern Detection Architecture

Symptom Recognition Strategies

- **Direct mention detection**: Identifies explicit mentions of symptoms ("I feel depressed")
- **Behavioral inference**: Maps behaviors to underlying symptoms ("I can't get out of bed" → Fatigue)
- **Contextual pattern recognition**: Identifies patterns across multiple messages
- **Negation handling**: Distinguishes between presence and absence of symptoms

Dynamic Pattern Collection

- Patterns are accumulated across the conversation, not just from single messages
- New symptoms and behaviors are added to the user's evolving clinical profile
- Conflicting information is handled with recency prioritization
- Boolean flag new_info_detected tracks when new clinical information appears

Pattern Analysis Examples

Example 1: Depression Recognition

User input: "I've been feeling so low lately. I just can't seem to enjoy anything and I'm always tired."

System maps to:

- Symptoms: ["Depression"]
- Behaviors: ["Fatigue", "Diminished interest"]

The combination triggers clinical rule for depression pattern

Example 2: Anxiety with Panic Features

User input: "My heart races and I feel like I can't breathe. I'm constantly worried something terrible will happen."

System maps to:

- Symptoms: ["Anxiety", "Shortness of breath", "Chest pain"]
- Behaviors: ["Excessive worry or fear"]

Pattern suggests possible panic disorder or generalized anxiety

Example 3: Bipolar Pattern Detection

User input: "Last week I was up all night with so many ideas and spent all my savings. Now I can barely function."

System maps to:

- Symptoms: ["Depression"]
- Behaviors: ["More talkative than usual", "Recklessness", "Fatigue"]

Pattern suggests manic episode followed by depressive crash

Neo4j Pattern Matching Algorithm

Once patterns are identified, the system employs a sophisticated Neo4j graph querying algorithm:

Clinical Relevance Scoring

- **Base matching**: Symptom matches (×2 points) + behavior matches
- **Diagnostic coverage**: What percentage of the disorder's typical symptoms are present
- **User coverage**: What percentage of the user's symptoms are explained by this disorder
- **Clinical adjustments**: Special rules for differential diagnosis challenges

Query Construction

```
WITH d, matching_symptoms, matching_behaviors, match_points * 0.4 + // Base symptom matching diagnostic_coverage * 30 + // How much of the disorder criteria are met user_coverage * 20 + // How much of the user's presentation is explained clinical_adjustment // Clinical rule-based adjustments AS clinical_relevance_score
```

Multi-layer Matching Strategy

1. **Step 1**: Direct disorder matching with clinical scoring
2. **Step 2**: If unsuccessful, find similar patients based on symptoms
3. **Step 3**: If still unsuccessful, find disorders through symptom categories
4. **Step 4**: Fall back to general patient matching as last resort

Conversation State Management

The system maintains a comprehensive conversation state for each user session:

State Object Structure

```
class ConversationState:
    def __init__(self, session_id):
        self.session_id = session_id
        self.conversation_history = []
        self.detected_symptoms = []
        self.detected_behaviors = []
        self.weight_conditions = []
        self.substance_use = []
        self.recommended_disorder = None
        self.recommended_therapy = None
        self.recommended_coping = []
        self.last_database_query_count = 0
        self.similar_patients = []
```

Dynamic Pattern Collection Logic

```
if "symptoms" in patterns and patterns["symptoms"]:
    for symptom in patterns["symptoms"]:
        if symptom not in state.detected_symptoms:
            state.detected_symptoms.append(symptom)
            new_info_detected = True
```

Database Query Triggering

- Only queries when sufficient information is available
- Requires at least one symptom or behavior to begin matching
- Returns top 3 matching disorders with confidence scores

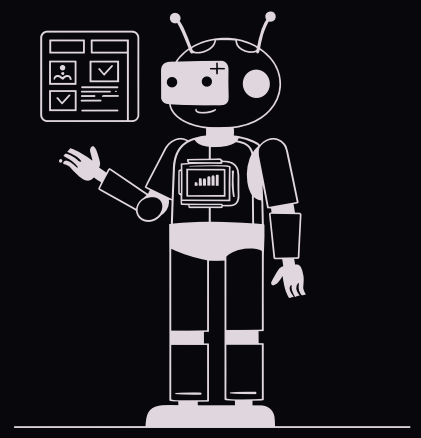
Therapeutic Response Generation

Context Preparation

- Recent conversation history (last 5 messages)
- Identified disorder (without explicitly mentioning diagnosis)
- Recommended therapy approach principles
- Suggested coping mechanisms

Therapeutic Guidance Injection

```
therapeutic_guidance = ""
if state.recommended_disorder:
    therapeutic_guidance += f"\nThe user may be experiencing symptoms consistent with {state.recommended_disorder}."
if state.recommended_therapy:
    therapeutic_guidance += f"\nUse principles from {state.recommended_therapy} in your response without explicitly mentioning the approach."
```



Response Variation Requirements

- Explicitly instructs the model to vary responses
- Never repeat the same phrases or sentence structures
- Reference specific details from user messages
- Natural variation in response length and style
- Occasionally (but not always) ask follow-up questions

Final Response Formulation

- The Gemma model processes the complete guidance
- Generates a natural, conversational response
- Incorporates therapeutic principles without clinical language
- Response is added to conversation history for context

Diagnostic Decision Override System

The system implements special clinical logic for challenging differential diagnoses:

```
# Clinical override for depression vs bipolar
if has_depression_symptoms and not has_manic_symptoms:
    # Check if we incorrectly recommended bipolar
    if disorders and "Bipolar" in disorders[0]["name"]:
        # Look for Major Depressive Disorder in our results
        depression_found = False
        for disorder in disorders:
            if disorder["name"] == "Major Depressive Disorder":
                depression_found = True
                # Move it to the top
                disorders.remove(disorder)
                disorders.insert(0, disorder)
                break

    # If no depression in results but strong indicators, override
    if not depression_found and (
        "Depression" in state.detected_symptoms or
        ("Fatigue" in state.detected_behaviors and "Diminished interest" in state.detected_behaviors)
    ):
        # Add Major Depressive Disorder as top result
        disorders.insert(0, {
            "name": "Major Depressive Disorder",
            "relevance": max(d["relevance"] for d in disorders) + 1 if disorders else 5,
            "confidence": 0.8
        })
```

Demo Time!