

## The Easiest Implementation: The "DPO-QLoRA" Pipeline

To avoid the complexity of training a separate Reward Model (required for PPO), the current industry standard for a "simple" RL implementation is the DPO pipeline.

### 1. Generate Synthetic Preference Pairs (The "Data")

Instead of writing 100,000 hours of dialogues like Dartmouth, use a "Teacher" model (e.g., GPT-4o or Claude 3.5 Sonnet) to generate synthetic interactions:

- **Prompt:** Use a "Dual-Agent" setup where one LLM plays a "Distressed Client" (based on DSM-5 personas) and another plays a "CBT Therapist."
- **Create Pairs:** For every therapist prompt, generate two responses:
  - **Chosen (\$y\_w\$):** A response that follows clinical guidelines (e.g., uses Socratic questioning).
  - **Rejected (\$y\_l\$):** A response that is technically coherent but clinically poor (e.g., dismissive, overly prescriptive, or "toxic positivity").

### 2. Apply QLoRA + DPO (The "Training")

Use a library like **Hugging Face TRL (Transformer Reinforcement Learning)** to fine-tune a base model (like Llama 3 or Qwen 2.5).

- **QLoRA** keeps memory usage low (allowing you to train on a single consumer GPU).<sup>1</sup>
- **DPO** optimizes the model to maximize the log-likelihood of the "Chosen" response while minimizing the "Rejected" one.<sup>2</sup>

## Part 1: Generating Synthetic Preference Pairs

In mental health, "preference" means choosing a response that is clinically safe and empathetic over one that is dismissive or harmful. You need a "Teacher" model to generate a prompt, a "good" response, and a "bad" response.

### Recommended Libraries:

- **[Distilabel](#) (by Argilla/Hugging Face):** Currently the "gold standard" for synthetic preference data. It allows you to create pipelines where a model (e.g., Claude or Llama 3) generates clinical scenarios and then acts as a "Judge" to rank multiple candidate responses.
- **[DataDreamer](#):** A high-level library designed specifically for "LLM-in-the-loop" workflows.<sup>2</sup> It handles the prompt chaining required to generate a patient persona, a therapist's response, and a clinical critique in one reproducible script.
- **[NeMo Curator](#):** If you are working with very large datasets, NVIDIA's tool is optimized for high-throughput synthetic data generation and quality filtering.

### Example Workflow:

2. Use **Distilabel** to prompt an LLM: "Simulate a patient with symptoms of MDD."
  3. Generate two responses: one using **CBT techniques** (Chosen) and one using **Toxic Positivity** (Rejected).
  4. Save these as a .jsonl dataset in the format: {"prompt": "...", "chosen": "...", "rejected": "..."}.
- 

## Part 2: Applying QLoRA + DPO

Once you have your preference pairs, you need to "align" your model so it learns to prefer the therapeutic style.

### Recommended Libraries:

- [\*\*TRL \(Transformer Reinforcement Learning\)\*\*](#): This is the core library for the **DPOTrainer**. It integrates seamlessly with peft (for LoRA) and bitsandbytes (for 4-bit quantization).
  - [\*\*Unsloth\*\*](#): This is highly recommended for individual developers. It provides specialized kernels that make DPO training **2x faster** and use **60% less memory** than standard Hugging Face implementations. It allows you to run DPO on a single 24GB GPU (like an RTX 3090/4090).
  - [\*\*Axolotl\*\*](#): A config-driven trainer. If you don't want to write Python training loops, you just create a .yaml file specifying your model, dataset, and dpo as the training method, and Axolotl handles the rest.
- 

## The "Simplest" Implementation Stack

If you want the fastest path from zero to a trained model, use this combination:

Task	Library	Why?
Data Generation	<b>Distilabel</b>	Has built-in "Self-Correction" templates for clinical data.
Quantization	<b>bitsandbytes</b>	Standard for 4-bit (QLoRA) efficiency.
Training Engine	<b>Unsloth + TRL</b>	Lowest VRAM requirements; extremely

		beginner-friendly.
<b>Tracking</b>	<b>WandB (Weights &amp; Biases)</b>	Essential for monitoring the "DPO Loss" (which can be finicky).

## Clinical Implementation Tip

When using synthetic data for mental health, your "Rejected" responses should not just be "bad grammar." They should be **clinical errors**, such as:

1. **Giving direct advice** (non-Socratic).
2. **Missing a risk signal** (failure to flag self-harm).
3. **Invalidating the user's emotion** ("Just try to be happy").

By training the model on these specific "Rejected" pairs, the resulting LLM becomes much more robust at maintaining professional boundaries than a model trained on SFT alone.

## Use of [PsychBERT](#) and [sentiment-classification-bert-mini](#)

These two models can be integrated into your mental health LLM workflow as **automated "Reward Models"** or **"Quality Filters"** during the synthetic data generation phase.

In a DPO (Direct Preference Optimization) pipeline, the hardest part is determining which response is "Chosen" and which is "Rejected." Instead of hiring human therapists to label 10,000 pairs, you use these BERT-based models to score responses automatically.

### 1. Using psychbert-cased for Clinical "Chosen" Labeling

**PsychBERT** is specifically trained on mental health literature and social media conversations. In your workflow, it acts as a **domain-specific judge**.

- **How to use it:** When your synthetic "Teacher" model (like Llama 3) generates two potential therapist responses, you pass both through PsychBERT.
- **The Logic:** You can use the model to extract embeddings and calculate which response aligns more closely with "professional clinical language" vs. "general internet chat."
- **Workflow Role:** It serves as a filter to ensure the "Chosen" response contains the semantic markers of actual clinical discourse rather than just generic polite text.

### 2. Using sentiment-classification-bert-mini for Safety Guardrails

This is a lightweight model focused on emotional tone. In mental health, a "good" response is often neutral-to-supportive, while a "rejected" response might be overly negative, dismissive, or "toxic."

5. **How to use it:** You use this model to score the **emotional valence** of the generated responses.
  6. **The Logic:**
    - **High Sentiment Score (Negative):** If the BERT model detects high frustration or negativity in the LLM's response, it is automatically labeled as **Rejected**.
    - **High Sentiment Score (Supportive/Neutral):** If the response is balanced and empathetic, it is a candidate for **Chosen**.
  7. **Workflow Role:** It acts as an automated safety layer to catch "hallucinated rudeness" or dismissive tones that a general LLM might occasionally produce during synthetic generation.
- 

## The Integrated Workflow: "The Automated Judge"

4. **Generation:** Your base LLM generates Response A and Response B for a patient prompt.
5. **Clinical Scoring:** psychbert-cased analyzes both. Response A is found to have a 0.85 clinical relevance score; Response B has 0.40.
6. **Sentiment Scoring:** sentiment-classification-bert-mini checks for "Toxic Positivity" or "Negativity." If Response A is empathetic but Response B is dismissive, Response B's score drops.
7. **Pair Construction:** Your script automatically creates a DPO pair: {"prompt": "...", "chosen": "Response A", "rejected": "Response B"}.
8. **DPO Training:** You feed thousands of these BERT-verified pairs into the **TRL DPOTrainer** to align your model.

## Why this is better than "Standard" LLM Labeling

Using smaller BERT models as judges is often **more reliable and 10x cheaper** than using a massive LLM (like GPT-4) as a judge. PsychBERT, in particular, has a "latent understanding" of clinical terminology that general-purpose models often lack, making your training data higher quality from a medical standpoint.

### [PsychBERT and Mental Health NLP](#)

This video provides a deep dive into using BERT-based models for mental health sentiment analysis, which is the foundational step for creating the "Reward Model" logic in your DPO pipeline.

This script uses the two BERT models you mentioned—**PsychBERT** to verify clinical relevance and the **Sentiment-Mini** model to ensure an empathetic tone—acting as automated "filters" for your synthetic data.

## Phase 1: Generating the DPO Dataset (Distilabel)

This script defines a pipeline that generates two therapist responses and uses the BERT

models to decide which one is "Chosen."

Python

```
from distilabel.pipeline import Pipeline
from distilabel.steps import LoadDataFromDicts
from distilabel.steps.tasks import TextGeneration
from transformers import pipeline
import torch

# 1. Initialize our BERT "Judges"
clinical_judge = pipeline("feature-extraction", model="mnaylor/psychbert-cased")
sentiment_judge = pipeline("sentiment-analysis",
model="Varnikasiva/sentiment-classification-bert-mini")

def clinical_scoring_step(inputs):
    """
    Custom logic: Scores responses based on clinical sentiment and terminology.
    In a real scenario, you'd compare embeddings to a 'gold standard' clinical text.
    """
    for item in inputs:
        # Check sentiment for 'empathy' (rejecting high negativity)
        sent_a = sentiment_judge(item["generation_model_a"])[0]
        sent_b = sentiment_judge(item["generation_model_b"])[0]

        # Scoring Logic: Prefer 'Positive/Neutral' sentiment + PsychBERT similarity
        # Here we simplify: Choose the one with better sentiment & lower 'toxicity'
        if sent_a['label'] == 'POSITIVE' and sent_b['label'] == 'NEGATIVE':
            item["chosen"] = item["generation_model_a"]
            item["rejected"] = item["generation_model_b"]
        else:
            item["chosen"] = item["generation_model_b"]
            item["rejected"] = item["generation_model_a"]

    return inputs

# 2. Define the Distilabel Pipeline
with Pipeline(name="MentalHealth-DPO") as pipeline:
    # Seed data: Common patient scenarios
    loader = LoadDataFromDicts(data=[
        {"instruction": "I've been feeling very hopeless lately and can't get out of bed."},
```

```

        {"instruction": "I feel a panic attack coming on, my heart is racing."}
    ])

# Generate two different styles using a 'Teacher' model (e.g., Llama 3)
gen_a = TextGeneration(lilm=teacher_llm, system_prompt="You are a warm CBT therapist.")
gen_b = TextGeneration(lilm=teacher_llm, system_prompt="You are a direct, clinical
psychiatrist.")

# Apply our custom BERT filtering (simplified for this example)
# [Logic to route to clinical_scoring_step would go here]

```

---

## Phase 2: Training with QLoRA + DPO (TRL & Unsloth)

Once your dataset is ready in the {"prompt", "chosen", "rejected"} format, you can run the training. Using **Unsloth** makes this much faster on a single GPU.

Python

```

from unsloth import FastLanguageModel, PatchDPOTrainer
from trl import DPOConfig, DPOTrainer
from datasets import load_dataset

# 1. Load Model & Tokenizer (4-bit for efficiency)
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/llama-3-8b-instruct-bnb-4bit",
    max_seq_length = 2048,
    load_in_4bit = True,
)

# 2. Add LoRA Adapters
model = FastLanguageModel.get_peft_model(
    model,
    r = 16,
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj"],
    lora_alpha = 16,
    lora_dropout = 0,
)

# 3. Initialize DPO Trainer

```

```

dpo_trainer = DPOTrainer(
    model = model,
    ref_model = None, # Unslot handles this internally to save memory
    args = DPOConfig(
        per_device_train_batch_size = 2,
        gradient_accumulation_steps = 4,
        beta = 0.1, # The 'strength' of the RL alignment
        learning_rate = 5e-5,
        max_steps = 1000,
        output_dir = "mental-health-model-v1",
    ),
    train_dataset = my_distilabel_dataset,
    tokenizer = tokenizer,
)
dpo_trainer.train()

```

## The Benefit of Using Your Specific Models

- **PsychBERT as a "Relevance Filter":** During the data generation phase, you can use PsychBERT to calculate **Cosine Similarity** between your generated response and a corpus of real clinical transcripts. If the LLM generates a response that is "too robotic" or "unscientific," the similarity score will be low, and you can mark that response as **Rejected**.
- **Sentiment BERT as a "Tone Guard":** This ensures the model doesn't just learn "correct facts" but also learns "gentle delivery." If the sentiment model flags a response as **0.99 Negative**, it becomes an automatic "Rejected" example, teaching the LLM that clinical accuracy *without empathy* is a failure.

## Use of GoEmotions and Hourglass of Emotions

To incorporate the GoEmotions valence mapping and the Hourglass of Emotions polarity scoring into a single workflow, you can move from a simple weighted average to a **Unified Affective Matrix**. This approach reconciles the discrete labels of GoEmotions with the multi-dimensional clinical theory of the Hourglass model.

### 1. GoEmotions Valence Mapping (Extracted Code)

Based on the provided consecutive\_emotion.ipynb and ML (1).docx, the following code implements the transformation of GoEmotions probabilities into a continuous valence scale

(\$[-1, 1]\$).

Python

```
from typing import Dict
import torch.nn.functional as F
import torch

# Clinically-tuned lookup table mapping discrete labels to valence weights
VALENCE_MAP: Dict[str, float] = {
    "admiration": 0.7, "amusement": 0.8, "anger": -0.8, "annoyance": -0.6,
    "approval": 0.5, "caring": 0.7, "confusion": -0.2, "curiosity": 0.2,
    "desire": 0.3, "disappointment": -0.7, "disapproval": -0.6, "disgust": -0.9,
    "embarrassment": -0.5, "excitement": 0.8, "fear": -0.9, "gratitude": 0.9,
    "grief": -1.0, "joy": 1.0, "love": 0.9, "nervousness": -0.7,
    "optimism": 0.6, "pride": 0.7, "realization": 0.1, "relief": 0.6,
    "remorse": -0.8, "sadness": -0.9, "surprise": 0.1, "neutral": 0.0,
}

def compute_valence(text, model, tokenizer):
    """
    Transforms GoEmotions discrete probabilities into a continuous valence score.
    """

    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True)
    with torch.no_grad():
        logits = model(**inputs).logits
        # Use sigmoid as GoEmotions is a multi-label dataset
        probs = torch.sigmoid(logits).cpu().numpy()[0]

    # Calculate weighted mean of active labels
    valence_score = 0.0
    for label, prob in zip(model.config.id2label.values(), probs):
        valence_score += prob * VALENCE_MAP.get(label, 0.0)

    return valence_score
```

## 2. Reconciliation with the Hourglass of Emotions

You do not need to keep these as independent frameworks. In fact, standardizing them into a **Unified Framework** is clinically superior because it provides context to the valence score

(e.g., distinguishing between "sadness-driven" and "anger-driven" negative valence).

## The Reconciliation Mechanism: The 4-Dimension Mapping

The Hourglass Model Revisited defines polarity through four dimensions:

- **Introspection (I):** Sadness  $\leftrightarrow$  Joy
- **Temper (T):** Anger  $\leftrightarrow$  Calmness
- **Attitude (A):** Disgust  $\leftrightarrow$  Pleasantness
- **Sensitivity (S):** Fear  $\leftrightarrow$  Eagerness

You can map the 27 GoEmotions into this  $4 \times 27$  matrix. For example, love is a bi-dimensional emotion (Joy + Pleasantness), while grief is primarily negative Introspection.

## Standardized Scoring Formula

Instead of the simple sum used in the GoEmotions code, use the Refined Hourglass Formula from the revisited model:

$$\text{Polarity} = \frac{I_{\text{total}} + T_{\text{total}} + A_{\text{total}} + S_{\text{total}}}{N}$$

Where  $N$  is the number of active dimensions (dimensions with a non-zero sum).

## Proposed Unified Code Framework

### Python

```
# Map GoEmotions labels to Hourglass Dimensions [I, T, A, S]
HG_MATRIX = {
    "joy": [1.0, 0, 0, 0],      # Pure Introspection
    "anger": [0, -1.0, 0, 0],   # Pure Temper
    "disgust": [0, 0, -1.0, 0], # Pure Attitude
    "fear": [0, 0, 0, -1.0],   # Pure Sensitivity
    "love": [0.5, 0, 0.5, 0],  # Bi-dimensional (Joy + Pleasantness)
    "remorse": [-0.5, -0.5, 0, 0] # Mixed (Sadness + Anger)
    # ... map all 27 labels similarly
}

def compute_standardized_polarity(probs, labels):
    dims = [0.0, 0.0, 0.0, 0.0] # I, T, A, S
    for prob, label in zip(probs, labels):
        if label in HG_MATRIX:
```

```

mapping = HG_MATRIX[label]
for i in range(4):
    dims[i] += prob * mapping[i]

active_dims = sum(1 for d in dims if abs(d) > 0.01)
return sum(dims) / max(active_dims, 1)

```

## Clinical Benefits of This Unified Approach

8. **Risk Categorization:** As suggested in 10-30-2025.pdf, you can set thresholds. A low Polarity score driven by the **Temper (T)** dimension indicates a risk of aggression/outbursts, while a low score driven by **Sensitivity (S)** indicates a risk of anxiety or panic attacks.
9. **Clinical Calibration:** The ML (1).docx file highlights the use of **Isotonic Regression** to calibrate these scores against human-expert ratings. This "tunable" layer ensures the AI's emotional assessment matches a therapist's judgment.
10. **Trend Visualization:** By tracking the 4 dimensions independently over time, the "Daily Chatbot" mentioned in your workflow can provide therapists with a "Mood Profile" rather than just a simple sentiment line.

## Putting it all together (part 1)

The integration of the **Unified Affective Matrix** transforms your DPO pipeline from a simple "sentiment tracker" into a **Clinical Guardrail System**. By mapping GoEmotions to the 4 Hourglass dimensions (Introspection, Temper, Attitude, Sensitivity), you can automate the selection of training pairs based on therapeutic goals rather than just "politeness."

Here is how to implement this in both phases of your workflow.

## Phase 1: Using the Matrix as a "Clinical Scorer" in Distilabel

In Phase 1, the goal is to choose the "Better" response ( $y_{w}$ ). With the Unified Matrix, "Better" is defined as the response that moves the patient's affective state toward a **target clinical profile**.

### 1. Define the Mapping (The Unified Matrix)

First, we create the mapping between the 27 GoEmotions and the 4 Hourglass Dimensions.

Python

```

# Standardized Mapping: GoEmotions -> [Introspection, Temper, Attitude, Sensitivity]
UNIFIED_MATRIX = {
    "joy": [1.0, 0, 0, 0], "sadness": [-1.0, 0, 0, 0], "grief": [-1.0, 0, 0, 0],
    "anger": [0, -1.0, 0, 0], "annoyance": [0, -0.6, 0, 0],
    "disgust": [0, 0, -1.0, 0], "admiration": [0, 0, 0.8, 0],
    "fear": [0, 0, 0, -1.0], "nervousness": [0, 0, 0, -0.7],
    "optimism": [0.5, 0, 0, 0.5], # Bi-dimensional
    "caring": [0.3, 0, 0.7, 0], # Joy + Pleasantness
    "neutral": [0, 0, 0, 0]
}

def get_clinical_score(probs, labels):
    # Calculate scores for the 4 dimensions
    dims = [0.0, 0.0, 0.0, 0.0]
    for p, label in zip(probs, labels):
        if label in UNIFIED_MATRIX:
            for i in range(4): dims[i] += p * UNIFIED_MATRIX[label][i]

    # The 'Polarity' formula from the Revisited Hourglass model
    active_dims = sum(1 for d in dims if abs(d) > 0.05)
    polarity = sum(dims) / max(active_dims, 1)
    return polarity, dims

```

## 2. Custom Distilabel Step

Use this logic to label your synthetic data. For example, if a patient expresses high **Anger (-T)**, the "Chosen" response should be the one that produces a **Calmness (+T)** shift.

Python

```

from distilabel.steps import Step
from distilabel.steps.typing import StepInput

class ClinicalPreferenceStep(Step):
    def process(self, inputs: StepInput) -> StepInput:
        for item in inputs:
            # Score both generated responses
            score_a, dims_a = get_clinical_score(item["probs_a"], item["labels"])
            score_b, dims_b = get_clinical_score(item["probs_b"], item["labels"])

```

```

# Clinical Rule: Reject 'Toxic Positivity'
# (e.g., if response A has high valence but ignores the patient's 'Fear' dimension)
if dims_a[3] < -0.5 and score_a > 0.5: # Negative Sensitivity + Positive Valence = Invalid
    item["chosen"] = item["generation_b"]
    item["rejected"] = item["generation_a"]
elif score_a > score_b:
    item["chosen"] = item["generation_a"]
    item["rejected"] = item["generation_b"]
else:
    item["chosen"] = item["generation_b"]
    item["rejected"] = item["generation_a"]
return inputs

```

## Phase 2: Influence on QLoRA + DPO Training

In Phase 2, the Unified Matrix ensures that the "Margin" between the chosen and rejected response is clinically meaningful.

- **Focusing the "Loss":** By using the Matrix to filter your dataset, you ensure the DPO algorithm (\$L\_{DPO}\$) is optimizing for **Affective Regulation**. The model learns that a "Correct" answer is one that addresses the specific dimension in deficit (e.g., increasing *Attitude* when *Disgust* is detected).
- Calibrating Beta (The KL Constraint):  
The beta parameter in DPOConfig controls how much the model can deviate from the base model.
  - **High Beta (0.5+):** Use this if your Unified Matrix scores show that the base model is "clinically unsafe" (e.g., frequently responds with negative *Temper*). This forces the model to stick strictly to the preferences derived from the Hourglass theory.
  - **Low Beta (0.1):** Use this if the base model is already empathetic but just needs "fine-tuning" on specific clinical terminology from **PsychBERT**.
- Validation via the Matrix:  
During training, you can use the Isotonic Regression (mentioned in your ML (1).docx) as an external evaluator. Every 100 steps, you generate a sample from your model, score it with the Unified Matrix, and check if the Polarity is improving over the training run.

## Summary of Benefits

- **Reconciled Framework:** You no longer just track "happy vs. sad." You track **Clinical Polarity**.
- **Safety:** It prevents the LLM from learning "Harmful Support" (where the model is polite but encourages negative *Introspection* or *Sensitivity*).

- **Interpretability:** If a model fails, the Unified Matrix tells you *which* dimension failed (e.g., "The model is too aggressive," i.e., Low Temper).

The "Emotion-Informed State Representation" defined in the RRL framework ( $s_t = [u_t, h_t, e_t]$ ) can be directly integrated into your existing DPO workflow. In this context, the **Unified Affective Matrix** acts as the engine that generates the  $e_t$  (affective embedding) component of that state.

By combining these, you move from training on isolated prompt-response pairs to training on **state-aware therapeutic trajectories**.

## Putting it all together (part 2)

### Mapping RRL Components to Your Workflow

RRL State Component	Clinical/Technical Implementation	Source
$u_t$ (User Attributes)	Static metadata in the prompt (e.g., "Diagnosis: MDD", "Severity: High").	User Profile / EHR
$h_t$ (History)	A rolling window of previous session polarity scores (mean valence of last 3 turns).	Conversation Logs
$e_t$ (Affective Embedding)	The 4-dimensional vector $[I, T, A, S]$ from your <b>Unified Affective Matrix</b> .	GoEmotions + Matrix

---

## Phase 1: State-Augmented Synthetic Generation (Distilabel)

In this phase, you use the RRL state to "prime" the Teacher LLM. Instead of asking for a generic response, you ask it to respond to a specific **emotional state and history**.

### Step 1: Constructing the "State-Aware" Prompt

Modify your Distilabel pipeline to prepend the RRL state to every instruction.

Python

```
def create_rrl_prompt(instruction, user_profile, history_scores, current_matrix_dims):
    # e_t: Current emotional dimensions from your Hourglass-GoEmotions mapping
    et_str = f"Introspection: {current_matrix_dims[0]:.2f}, Temper: {current_matrix_dims[1]:.2f}"

    # h_t: Brief history summary
    ht_str = "Improving" if history_scores[-1] > history_scores[0] else "Declining"

    state_header = (
        f"[USER STATE]\n"
        f"Diagnosis: {user_profile['diagnosis']}\n"
        f"Current Affect (e_t): {et_str}\n"
        f"Recent Trend (h_t): {ht_str}\n"
        f"---"
    )
    return f"{state_header}\nPatient: {instruction}"

# Use this in Distilabel to generate 'Chosen' vs 'Rejected' responses
# that are specifically tailored to 'correcting' the state.
```

## Step 2: Scoring the State Transition

In RRL, the reward depends on the transition  $s_t \rightarrow s_{t+1}$ . In your DPO dataset, the **Chosen (\$y\_w\$)** response is the one that minimizes clinical risk in the next state.

Python

```
def rrl_scoring_logic(generated_response, previous_state):
    # 1. Calculate new affective state ( $e_{t+1}$ ) using your Unified Matrix
    new_dims = get_clinical_score_from_matrix(generated_response)

    # 2. Check for 'Emotional Alignment' (r_emo from RRL)
    # If user is in high distress (Low Sensitivity), the response
    # must show high 'Aptitude/Support' to be 'Chosen'.
    if previous_state['sensitivity'] < -0.7 and new_dims['support'] < 0.5:
        return -1.0 # High penalty: response was insensitive to state
```

```
return new_dims['polarity'] # Return the clinically calibrated score
```

## Phase 2: Context-Conditioned Training (TRL & Unsloth)

When you reach the training phase, you don't just train the model on the patient's text; you train it to **read and respond to the state metadata**.

- **Input Formatting:** Your training samples should look like this:
  - **Prompt:** [STATE: MDD, Low-Temper] Patient: I hate everything.
  - **Chosen:** It sounds like you're feeling a lot of frustration right now. Let's look at what triggered that...
  - **Rejected:** Why are you being so negative? You should try to be happy.
- Multi-Objective Alignment:  
By including the RRL state in the prompt during DPO training, the LoRA adapters learn to associate specific "State Profiles" with specific "Therapeutic Strategies" (e.g., if state shows high Temper, use De-escalation).

### Why this reconciles the frameworks:

- **The Matrix is the Sensor:** It provides the high-resolution emotional data ( $e_t$ ) required by the RRL framework.
- **RRL is the Strategy:** It provides the mathematical "State" logic that tells the DPO algorithm *why* one response is better than another across a session, not just a single turn.

## Summary of the Combined Stack

9. **GoEmotions (BERT):** Categorizes the raw text into 27 buckets.
10. **Unified Matrix:** Maps those 27 buckets into 4 Hourglass Dimensions ( $I, T, A, S$ ).
11. **RRL State ( $s_t$ ):** Combines those 4 dimensions with user history and diagnosis.
12. **Distilabel:** Generates synthetic pairs where  $y_w$  is the response that improves  $s_t$ .
13. **TRL (DPO):** Trains the LLM to internalize this state-to-response mapping.

By removing the Efficacy Measurement layer (EMA and Bio-correlates), the stack becomes a **purely linguistic clinical feedback loop**. The model now relies entirely on the **Unified Affective Matrix** to interpret the patient's state and uses **GRPO** to explore which verbal interventions yield the best "clinical trajectory."

Here is the rebuilt stack and the specific code transitions needed to move from DPO to GRPO.

## Putting it all together (part 3)

### Using GRPO instead of DPO

- **GoEmotions (BERT)**: Categorizes raw text into 27 buckets.
  - **Unified Matrix**: Maps these buckets into the 4 Hourglass Dimensions ( $I, T, A, S$ ).
  - **RRL State ( $s_t$ )**: Fuses the  $[I, T, A, S]$  vector with the conversation history and user diagnosis to create the "Emotion-Informed State."
  - **Distilabel (Group Gen)**: Generates  $k=8$  completions for a single state.
  - **TRL (GRPO)**: Optimizes the policy by comparing the relative clinical rewards of those 8 completions.
- 

## Step 1: The Unified Matrix Reward Engine

Without bio-data, the reward is derived from the "Affective Shift." We define a reward function that looks at how well a response addresses the dimension currently in deficit (e.g., if  $s_t$  shows low **Temper**, the reward favors responses that increase **Calmness**).

Python

```
# Unified Matrix Mapping (Internal Logic)
HG_MATRIX = {
    "joy": [1.0, 0, 0, 0], "sadness": [-1.0, 0, 0, 0],
    "anger": [0, -1.0, 0, 0], "fear": [0, 0, 0, -1.0],
    # ... rest of the 27 GoEmotions mapped to [I, T, A, S]
}

def calculate_affective_reward(completion_text, target_dimension_index):
    """
    Evaluates how well the response aligns with the required
    Hourglass dimension shift.
    """

    # 1. Get GoEmotions probabilities for the LLM's response
    probs = get_goemotions_probs(completion_text)

    # 2. Map to 4 dimensions
    dims = [0.0, 0.0, 0.0, 0.0]
    for p, label in zip(probs, labels):
        for i in range(4):
            dims[i] += p * HG_MATRIX.get(label, [0,0,0,0])[i]

    # 3. Reward is the value of the 'Target' dimension (e.g., boosting Temper)
    # plus a penalty for 'Toxic Positivity' (high Joy while ignoring Sadness)
    base_reward = dims[target_dimension_index]
```

```
    return base_reward
```

---

## Step 2: Transitioning Distilabel (Pairs \$\\to\$ Groups)

In DPO, you needed a "Chosen" and "Rejected" pair. In GRPO, you just need a prompt and a group of responses.

**Change:** Set num\_generations to 8 (or more) to allow for "Group Relative" comparison.

Python

```
from distilabel.steps.tasks import TextGeneration

# Distilabel update for GRPO
generate_responses = TextGeneration(
    llm=teacher_llm,
    num_generations=8, # GRPO needs a group to calculate relative advantage
    group_generations=True,
    system_prompt="Provide a therapeutic response based on the patient's RRL State."
)
```

---

## Step 3: The GRPOTrainer Code Snippet

This is the core architectural shift. Instead of training on fixed preferences, the model learns in real-time which of its own generated responses are "relatively" better for the clinical state.

Python

```
from trl import GRPOTrainer, GRPOConfig
from unsloth import PatchGRPOTrainer

# 1. Patch for memory efficiency (important for clinical-sized models)
PatchGRPOTrainer()
```

```

# 2. Define the GRPO Reward Functions
def clinical_alignment_reward(prompts, completions, **kwargs):
    # Uses the Unified Matrix to reward clinical 'correctness'
    return [calculate_affective_reward(c, target_idx) for c in completions]

def safety_reward(prompts, completions, **kwargs):
    # Penalize responses that are too short or contain harmful phrases
    return [1.0 if is_safe(c) else -2.0 for c in completions]

# 3. Configure the Trainer
training_args = GRPOConfig(
    num_generations=8,          # Group size
    max_prompt_length=512,
    max_completion_length=512,
    beta=0.04,                  # KL Penalty: The safety guardrail from your PPTX
    learning_rate=5e-6,
    per_device_train_batch_size=1,
    gradient_accumulation_steps=4,
)

trainer = GRPOTrainer(
    model=model,
    reward_funcs=[clinical_alignment_reward, safety_reward],
    args=training_args,
    train_dataset=dataset,
    tokenizer=tokenizer,
)

```

trainer.train()

---

## Key Differences in the Rebuilt Stack

11. **From Binary to Relative:** In DPO, if both generated responses were "bad," the model still had to pick one. In GRPO, the model sees 8 responses and learns that the "least bad" one is the baseline, while the "best" one provides the positive gradient.
12. **Trajectory Learning:** The **RRL State (\$s\_t\$)** now acts as the "Context" for the group. The model learns: "When the state is  $[I=-0.8, T=0.2]$ , the group members that use Socratic questioning ( $y_k$ ) have a higher relative reward than those using direct advice."