Case Study: Aligning a Clinical AI Assistant Using RLHF

MedAlign Health — Deploying Safe and Empathetic Patient-Facing AI in Telehealth

---

Section 1: Industry Context and Business Problem

Industry: Digital Health / Telehealth

The telehealth market has expanded rapidly, with virtual consultations becoming standard practice for non-emergency medical inquiries. AI-powered clinical assistants handle initial patient triage, symptom assessment, and post-visit follow-up, reducing clinician workload and improving patient access.

Company Profile: MedAlign Health

MedAlign Health is a Series B telehealth startup serving 2.3 million patients across 40 U.S. states. Their platform connects patients with board-certified physicians via video consultations. In 2024, they deployed "ClinAssist," an AI assistant that handles:

- Pre-visit symptom triage and severity assessment
- Post-visit care instruction delivery
- Medication reminder scheduling
- General health education queries

ClinAssist is built on a fine-tuned 7B parameter language model that was trained on medical dialogue datasets.

Business Challenge

After six months of deployment, MedAlign's clinical review board identified critical alignment issues:

1. **Tone Insensitivity**: ClinAssist occasionally provides medically accurate but emotionally cold responses to patients experiencing anxiety about diagnoses. Patient satisfaction scores for AI interactions dropped 18% in Q3.

2. **Over-Specificity**: The model sometimes provides overly detailed differential diagnoses that alarm patients unnecessarily. A patient asking about a headache might receive information about brain tumors, causing unwarranted panic.

3. **Safety Boundary Violations**: In 0.3% of interactions, the model provides responses that could be interpreted as medical advice beyond its intended scope (e.g., suggesting medication dosage changes).

4. **Inconsistent Empathy**: The model's empathetic responses are inconsistent — sometimes warm and supportive, sometimes clinical and detached, with no clear pattern.

Stakes

- **Patient Safety**: Misaligned responses could lead to patients making harmful health decisions
- **Regulatory Risk**: FDA and FTC have increased scrutiny of AI-generated medical content. Non-compliant responses could trigger regulatory action
- **Business Impact**: Patient churn rate for AI-dissatisfied users is 3.2x higher than the baseline
- **Liability**: MedAlign carries \$50M in malpractice insurance. Misaligned AI responses could expose the company to lawsuits

Constraints

- **Latency**: Responses must be generated in under 2 seconds for acceptable user experience
- **Compute Budget**: \$15,000/month for inference and training infrastructure
- **Privacy**: All training must use de-identified data compliant with HIPAA regulations
- **Evaluation**: Board-certified physicians must validate the alignment framework before deployment
- **Model Size**: 7B parameters maximum (inference must run on a single A100 GPU)

Section 2: Technical Problem Formulation

Problem Type: Policy Optimization with Human Feedback (RLHF)

Justification: The alignment problem here is fundamentally about matching human preferences — specifically, the preferences of board-certified physicians regarding what constitutes a safe, empathetic, and appropriate patient-facing response. This cannot be solved with supervised learning alone because:

1. There is no single "correct" response — multiple valid responses exist for any patient query
2. Quality is multidimensional (safety, empathy, accuracy, appropriate scope)
3. Physicians can reliably compare two responses but struggle to score individual responses on a consistent numerical scale

RLHF is the correct technical solution because it learns a reward model from physician preference comparisons and then optimizes the language model against this learned reward.

Input/Output Specifications

Input: Patient query x, consisting of: - Patient message (text, up to 512 tokens) - Conversation history (up to 3 prior turns) - Patient demographic context (age bracket, visit type)

Output: Clinical assistant response y, consisting of: - Natural language response (up to 256 tokens) - Internal safety flag (binary: requires physician escalation or not)

Mathematical Foundation

Stage 1: Reward Model

Given a prompt x and two candidate responses y_w (preferred by physicians) and y_l (rejected), the Bradley-Terry preference model gives:

$$P(y_w \succ y_l|x) = \sigma(r_\theta(x, y_w) - r_\theta(x, y_l))$$

The reward model loss:

$$\mathcal{L}_{\text{RM}} = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma(r_\theta(x, y_w) - r_\theta(x, y_l))\right]$$

Numerical Example: A physician compares two responses to "I have a persistent headache": - Response A (preferred, $r = 2.8$): "I understand that persistent headaches can be concerning. Based on what you have described, this could be tension-related. I recommend tracking the frequency and discussing with your physician at your next visit." - Response B (rejected, $r = 0.5$): "Persistent headaches can indicate conditions ranging from tension to more serious neurological issues including tumors. You should get an MRI immediately."

$$P(A \succ B) = \sigma(2.8 - 0.5) = \sigma(2.3) = 0.909$$

The model is 90.9% confident the physician prefers Response A.

Stage 2: PPO Optimization

The policy is optimized using the clipped PPO objective with KL penalty:

$$L^{\text{CLIP}} = \mathbb{E}\left[\min\left(r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon)A_t\right)\right]$$

Where the reward is modified to include a KL penalty:

$$r_{\text{total}} = r_{\text{RM}}(x, y) - \beta \cdot \text{KL}(\pi_\theta \| \pi_{\text{ref}})$$

Loss Function Design

The total training objective has three terms:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{PPO}} + \alpha \cdot \mathcal{L}_{\text{safety}} + \gamma \cdot \mathcal{L}_{\text{KL}}$$

Term 1: PPO Loss ($\mathcal{L}_{\text{PPO}}$): The clipped surrogate objective that maximizes reward from the physician-trained reward model. This drives the model toward responses that physicians prefer.

Term 2: Safety Loss ($\mathcal{L}_{\text{safety}}$): A binary cross-entropy term on the safety classifier head. This ensures the model correctly flags responses that require physician escalation. Weight $\alpha = 2.0$ ensures safety takes priority.

Term 3: KL Divergence ($\mathcal{L}_{\text{KL}}$): Prevents the model from deviating too far from the SFT baseline, guarding against reward hacking. Weight $\gamma = \beta = 0.1$.

Evaluation Metrics

1. **Preference Win Rate**: Fraction of A/B comparisons where the RLHF model is preferred by physicians over the SFT baseline (target: > 65%)
2. **Safety Compliance Rate**: Fraction of responses that pass the safety classifier (target: > 99.5%)
3. **Empathy Score**: Average score on a 1-5 physician-rated empathy scale (target: > 4.0)
4. **Factual Accuracy**: Fraction of medical claims rated as accurate by physicians (target: > 95%)
5. **Response Latency**: P95 generation time (target: < 2 seconds)

Baseline

The current SFT-only ClinAssist model achieves: - Preference Win Rate: 50% (by definition, against itself) - Safety Compliance: 99.2% - Empathy Score: 3.1 / 5.0 - Factual Accuracy: 94.8%

Why RLHF is the Right Approach

1. **Multidimensional Quality**: Safety, empathy, accuracy, and scope are difficult to capture in a single supervised loss
2. **Physician Preferences**: The "gold standard" is what a physician considers appropriate — RLHF directly optimizes for this
3. **Subtle Alignment**: The difference between "appropriate reassurance" and "dismissive" or "alarming" is subtle and best captured through comparisons
4. **Continuous Improvement**: New preference data can be collected and the model re-aligned without rebuilding from scratch

---

Section 3: Implementation Notebook Structure

3.1 Data Loading and Preprocessing

```
def load_preference_data(data_path: str) -> dict:
    """
    Load physician preference data from de-identified clinical conversations.
```

```
    Args:
        data_path: Path to the preference dataset (parquet format)

    Returns:
        dict with keys:
            'prompts': List[str] — patient queries with context
            'preferred': List[str] — physician-preferred responses
            'rejected': List[str] — physician-rejected responses
            'safety_labels': List[int] — 1 if requires escalation, 0 otherwise

    Hint:
        - Load with pandas, filter for complete triplets
        - Tokenize using the model's tokenizer
        - Truncate to max_length=512 for prompts, 256 for responses
    """
    # TODO: Implement
    pass
```

3.2 Exploratory Data Analysis

```
def analyze_preference_patterns(data: dict) -> None:
    """
    Analyze patterns in physician preferences.

    TODO:
        1. Plot response length distributions for preferred vs rejected
        2. Compute vocabulary overlap between preferred and rejected
        3. Analyze safety label distribution across query categories
        4. Identify most common keywords in preferred vs rejected responses
        5. Create a word cloud of empathetic vs clinical language

    Hint:
        - Use matplotlib for histograms and bar charts
        - Use collections.Counter for keyword analysis
    """
    # TODO: Implement
    pass
```

3.3 Baseline Model (SFT)

```
def build_sft_baseline(model_name: str, train_data: dict) -> nn.Module:
    """
    Fine-tune the base model on physician-written demonstrations.

    Args:
        model_name: HuggingFace model identifier
        train_data: Dict with 'prompts' and 'responses' (gold standard)

    Returns:
        Fine-tuned model

    Hint:
        - Use standard causal LM loss on concatenated prompt+response
        - Train for 3 epochs with lr=2e-5
        - Use gradient accumulation if batch size is limited
    """
    # TODO: Implement
    pass
```

3.4 Reward Model

```
class ClinicalRewardModel(nn.Module):
    """
    Reward model for clinical response quality.

    Architecture:
        - Shared LLM backbone (frozen lower layers, trainable upper layers)
        - Scalar reward head: Linear(hidden_dim, 1)
        - Safety classifier head: Linear(hidden_dim, 2)
```

```
    TODO: Implement forward method and loss computation
    """

    def __init__(self, base_model, hidden_dim: int = 4096):
        super().__init__()
        # TODO: Initialize backbone, reward head, safety head
        pass

    def forward(self, input_ids, attention_mask):
        """
        Returns:
            reward: (batch,) scalar reward scores
            safety_logits: (batch, 2) safety classification logits
        """
        # TODO: Implement
        pass

    def preference_loss(self, r_preferred, r_rejected, safety_logits, safety_labels):
        """
        Combined Bradley-Terry loss + safety classification loss.

        Hint:
            - Bradley-Terry: -log(sigma(r_preferred - r_rejected))
            - Safety: F.cross_entropy(safety_logits, safety_labels)
            - Combine with alpha=2.0 weight on safety
        """
        # TODO: Implement
        pass
```

3.5 Training Loop (PPO with KL Penalty)

```
def rlhf_training_loop(
    model: nn.Module,
    ref_model: nn.Module,
    reward_model: nn.Module,
    tokenizer,
    train_prompts: list,
    num_steps: int = 1000,
    batch_size: int = 8,
    beta: float = 0.1,
    epsilon: float = 0.2,
    lr: float = 1e-5,
):
    """
    Full RLHF training loop with PPO and KL penalty.

    TODO: Implement the training loop following these steps:
        1. Sample prompts from train_prompts
        2. Generate completions from current model
        3. Score with reward_model (reward + safety)
        4. Compute per-token KL penalty against ref_model
        5. Compute advantages (reward - value baseline)
        6. PPO update with clipped objective
        7. Log metrics every 50 steps

    Hint:
        - Use torch.no_grad() for generation and reward scoring
        - Clip gradients to norm 1.0
        - Monitor KL divergence — if it exceeds 10.0, increase beta
    """
    # TODO: Implement
    pass
```

3.6 Evaluation

```
def evaluate_alignment(
    model: nn.Module,
    ref_model: nn.Module,
    reward_model: nn.Module,
    eval_prompts: list,
```

```
    tokenizer,
) -> dict:
    """
    Comprehensive evaluation of alignment quality.

    TODO: Compute and return:
        1. Preference win rate (model vs ref_model, scored by reward_model)
        2. Average reward score
        3. Safety compliance rate
        4. KL divergence from reference
        5. Response length statistics

    Hint:
        - Generate from both model and ref_model for each prompt
        - Use reward_model to score and compare
        - Flag any response with safety_logits indicating escalation needed
    """
    # TODO: Implement
    pass
```

3.7 Error Analysis

```
def error_analysis(
    model: nn.Module,
    eval_data: dict,
    reward_model: nn.Module,
    tokenizer,
) -> None:
    """
    Analyze failure modes of the aligned model.

    TODO:
        1. Identify the 20 lowest-scoring responses
        2. Categorize failures: safety violations, low empathy, inaccuracy, scope creep
        3. Analyze if failures correlate with query type, length, or topic
        4. Generate side-by-side comparisons: model output vs physician-preferred output
        5. Plot reward distribution, highlighting the failure tail

    Hint:
        - Use pandas DataFrame for structured analysis
        - Create confusion matrix for safety classification
    """
    # TODO: Implement
    pass
```

3.8 Deployment Considerations

```
def prepare_for_deployment(model: nn.Module, tokenizer) -> dict:
    """
    Prepare the aligned model for production deployment.

    TODO:
        1. Export model with torch.jit.trace or ONNX
        2. Add response post-processing (safety filter, length limits)
        3. Create a FastAPI endpoint with proper error handling
        4. Implement request/response logging for monitoring
        5. Set up A/B testing configuration (RLHF vs SFT baseline)

    Hint:
        - Use vLLM or TGI for efficient inference
        - Implement a safety override that escalates to physician if safety flag triggers
        - Log all responses for ongoing physician review
    """
    # TODO: Implement
    pass
```

3.9 Ethical Considerations

```
def ethical_audit(model: nn.Module, tokenizer, audit_prompts: list) -> dict:
    """
    Audit the model for ethical concerns.

    TODO:
        1. Test for demographic bias (do response quality metrics vary by
           patient age, gender, or described symptoms?)
        2. Test for sycophancy (does the model agree with medically incorrect
           patient self-diagnoses?)
        3. Test boundary awareness (does the model refuse to provide advice
           outside its scope, like prescribing medication?)
        4. Test for hallucination (does the model cite non-existent studies
           or medications?)
        5. Compute disparity metrics across demographic groups

    Hint:
        - Create audit prompt sets for each category
        - Use physician annotations to validate boundary awareness
        - Flag any demographic disparity > 5% as requiring investigation
    """
    # TODO: Implement
    pass
```

Section 4: Production and System Design Extension

Architecture

The production RLHF-aligned ClinAssist system consists of:

1. **Inference Service**: vLLM-based serving of the 7B aligned model on a single A100 GPU, achieving P95 latency of 1.4 seconds
2. **Safety Gateway**: Real-time safety classifier that intercepts model outputs before delivery. Responses flagged as requiring escalation are routed to an on-call physician
3. **Reward Logging Service**: Captures all prompt-response pairs with metadata for ongoing reward model improvement
4. **Physician Feedback Pipeline**: Weekly batch of 500 randomly sampled interactions sent to board-certified physicians for preference annotation

API Design

```
POST /api/v1/clinical-assist
{
    "patient_id": "anonymized_hash",
    "message": "I have been having chest pain for two days",
    "conversation_history": [...],
    "context": {"age_bracket": "40-50", "visit_type": "follow-up"}
}

Response:
{
    "response": "I understand that chest pain can be very concerning...",
    "safety_flag": true,
    "escalation_reason": "Potential cardiac symptoms require physician review",
    "confidence_score": 0.82,
    "response_id": "uuid-for-tracking"
}
```

Serving Infrastructure

- **Primary**: Single A100 GPU (80GB) running vLLM with continuous batching
- **Fallback**: CPU-based inference with quantized (GPTQ 4-bit) model for overflow traffic
- **Scaling**: Horizontal scaling with Kubernetes, auto-scaling based on request queue depth
- **Caching**: Common query patterns cached with semantic similarity matching (cosine > 0.95)

Monitoring

1. **Reward Drift Detection**: Track mean reward model score over 7-day rolling windows. Alert if score drops > 0.5 standard deviations from the 30-day baseline
2. **KL Divergence Monitoring**: Continuously measure KL between production model and the last-deployed reference. Alert if KL > 5.0 (indicates model drift during serving, which should not happen but guards against bugs)
3. **Safety Alert Rate**: Track the percentage of responses flagged for escalation. Alert if rate exceeds 2% (up from baseline 0.8%)
4. **Patient Satisfaction Proxy**: Correlate reward model scores with post-interaction patient satisfaction surveys. Retrain reward model if correlation drops below 0.6

Drift Detection

- **Data Drift**: Monitor input query distribution using embedding-space density estimation. New symptom patterns or terminology (e.g., novel disease outbreaks) trigger retraining alerts
- **Concept Drift**: Monthly physician preference re-annotation on 200 samples. If physician preferences have shifted (agreement with reward model drops below 80%), schedule reward model retraining
- **Model Drift**: Should not occur in serving, but monitor via output entropy. Sudden entropy changes indicate potential issues

A/B Testing

- **Design**: 90/10 split (RLHF model / SFT baseline) for new patients, 100% RLHF for existing patients
- **Primary Metric**: Preference win rate assessed by blinded physician review (weekly)
- **Secondary Metrics**: Patient satisfaction score, escalation rate, response latency
- **Duration**: 4 weeks minimum, statistical significance at $p < 0.01$
- **Safety Override**: Automatic rollback to SFT baseline if safety compliance drops below 99.0% in the RLHF arm

CI/CD Pipeline

1. **Data Pipeline**: Weekly ingestion of new physician preference annotations into training dataset

2. **Training Pipeline**: Monthly reward model retraining followed by PPO re-optimization (triggered by concept drift detection)
3. **Evaluation Gate**: Automated evaluation on held-out test set. Model must pass all five metrics before promotion
4. **Staged Rollout**: Canary (1%) -> Shadow (10%, log-only) -> Production (100%)
5. **Rollback**: One-click rollback to previous model version with < 5 minute recovery time

Cost Analysis

Component	Monthly Cost
A100 GPU inference (1 instance)	\$8,500
Training compute (monthly retrain)	\$2,000
Physician annotation (2000 comparisons/month)	\$3,000
Storage and logging	\$500
Monitoring infrastructure	\$1,000
Total	**\$15,000**

This fits within MedAlign's \$15,000/month compute budget. The physician annotation cost is the main variable — it scales linearly with the number of preference pairs collected, but is essential for maintaining alignment quality.