Vision-Language Reward Models for Sample-Efficient Visual RL

Research Story & Comprehensive Implementation Plan

PART 1: THE RESEARCH STORY

Chapter 1: The Problem - Why Sample Efficiency Matters

The Classical Challenge in Visual RL

Imagine training a robot to learn a complex manipulation task. The agent needs to learn from images, making sense of high-dimensional visual observations. Every single environment step—every move the robot makes—is expensive: compute, real-world time, energy. Traditional reinforcement learning requires millions of steps to converge, making the whole endeavor economically unfeasible.

This is the fundamental tension in visual RL: **images provide rich information, but learning from them is sample-inefficient**.

Current State (2024-2025):

- **Hand-crafted rewards**: Researchers manually design reward functions (e.g., "distance to goal"). This works but is tedious, task-specific, and often fragile.
- **Learned rewards**: Train a reward model from human feedback. Better, but requires extensive annotation (hundreds of preferences).
- **RL from sparse rewards**: Let the agent learn without external guidance. Theoretically pure, but requires billions of environment steps.

The Bottleneck:

All three approaches share one problem: they're **inefficient at specifying what task the agent should actually learn**.

Chapter 2: The Breakthrough - Vision-Language Models as Reward Models

Enter VLM-RM (ICLR 2024)

In late 2024, researchers at AI safety companies (FAR AI, Alignment Research) published a striking paper: **"Vision-Language Models are Zero-Shot Reward Models for Reinforcement Learning"** (Rocamonde et al., ICLR 2024).

The core insight is deceptively simple: - **CLIP** (Contrastive Language-Image Pre-training) is trained on 400 million image-text pairs - CLIP can encode both images and text into the same embedding space - CLIP embedding similarity measures alignment between visual states and language descriptions - **Therefore: We can use CLIP similarity as a reward signal!**

The Revolutionary Result:

They trained a humanoid robot to perform complex tasks using only CLIP rewards: - "A humanoid robot kneeling" → Agent learns to kneel - "A humanoid robot doing the splits" → Agent learns gymnastics - "A humanoid robot in lotus position" → Agent learns meditation

No manual reward engineering. No human feedback collection. Just natural language.

The performance was surprisingly good—comparable to hand-crafted rewards on standard benchmarks, and applicable to novel tasks with zero examples.

Why This Works:

1. **Transfer from pretraining**: CLIP learned on massive internet-scale data, capturing visual semantics
2. **Natural language is expressive**: Unlike hand-crafted scalars, language can specify nuanced behaviors
3. **Zero-shot capability**: No task-specific fine-tuning needed
4. **Generalization**: CLIP's broad knowledge transfers to new domains

Chapter 3: The Hidden Constraint - Compute Cost

The Problem Nobody Talks About

While VLM-RM is revolutionary, it has a critical limitation: **computational cost**.

Every time the RL agent queries the reward: - Pass image to CLIP vision encoder (medium compute) - Pass text to CLIP text encoder (small compute) - Compute cosine similarity (trivial) - **Total**: ~100-200ms per query on a single GPU

For an RL agent running 1000 steps/episode, 1000 episodes/task: - **1 million reward queries needed** - At 100ms each: ~1000 GPU-hours just for reward computation!

The Reality:

- ICLR 2024 paper used clusters of GPUs
- Undergraduate researchers: limited to 1-2 GPUs
- Most AI labs: can't spare 1000 GPU-hours on a single project

The Gap:

VLM-RM works, but it's economically inaccessible to: - Undergraduate teams - Resource-constrained labs - Researchers in developing countries - Individual researchers without institutional backing

This gap is your research opportunity.

Chapter 4: Your Innovation - Efficient VLM-Reward Distillation

The Core Insight

What if we could: 1. Keep VLM-RM's capability (zero-shot, natural language rewards) 2. Eliminate VLM-RM's compute overhead (1000 GPU-hours) 3. Make it accessible to undergraduates with 2 GPUs

The Approach: Student-Teacher Distillation

Rather than querying CLIP at every RL step, we can:

1. **Offline Phase**: Collect diverse trajectories (random/weak policy)
   - Sample 10,000 states from various trajectories
   - Query CLIP for reward on each state (~100 GPU-hours total)
   - Build reward labels dataset
2. **Distillation Phase**: Train lightweight student reward model
   - Architecture: Lightweight CNN (EfficientNet-B0, ~5M params)
   - Input: Image observation
   - Output: Scalar reward prediction
   - Loss: MSE between student predictions and CLIP labels
   - Training time: ~10 GPU-hours
3. **Online Phase**: Use student for RL training
   - Student is 1000x faster than CLIP
   - Inference: ~0.1ms per query (vs 100ms for CLIP)
   - Can run all 1 million reward queries in ~1 GPU-hour!
4. **Adaptive Refinement**: Periodically fix distribution shift
   - During online RL, environment changes
   - Student trained on offline data may become stale
   - Every 50K RL steps: Query CLIP on 100 representative states
   - Fine-tune student on new labels
   - Budget: ~10 queries per refinement cycle

Net Result: - VLM-RM accuracy: - Lightweight inference: - 100x faster overall: - 10x less compute: - Undergraduate-accessible:

Why This Hasn't Been Done:

- Distillation for RL is studied, but typically for policy distillation (not reward)
- VLM-RM is new (published Oct 2024), distillation hasn't been applied yet
- Most groups with resources don't optimize for efficiency

Why This Works:

- Reward functions are often smooth & generalizable
- Lightweight CNNs capture visual features well

- Student doesn't need to match CLIP exactly—just capture reward signal correlates
- Adaptive refinement handles domain shift

Chapter 5: Why This Matters Beyond the Project

Broader Implications:

1. **Democratization**: Makes advanced RL techniques accessible to undergrads
2. **Environmental**: Reduces carbon footprint of AI research
3. **Reproducibility**: Enables results on limited hardware (not just lab clusters)
4. **Practical deployment**: Smaller models run on edge devices (robots, phones)
5. **Research velocity**: Faster iteration → faster science

Research landscape impact:

- First work to study efficient VLM-reward integration
- Opens new research direction: "What's the minimal compute for zero-shot rewards?"
- Benchmark for future work on resource-efficient RL
- Template for distilling other foundation models into RL algorithms

---

PART 2: COMPREHENSIVE IMPLEMENTATION PLAN

Phase 1: Foundation & Setup (Weeks 1-2)

Goal: Establish reproducible baseline, understand the landscape

Week 1: Literature Deep-Dive

Days 1-2: VLM-RM Internals - [ ] Read Rocamonde et al. (ICLR 2024) paper thoroughly - [ ] Understand CLIP architecture: vision encoder, text encoder, contrastive loss - [ ] Study baseline/goal-prompt regularization mechanism - [ ] Review their code repository: github.com/AlignmentResearch/vlmrm - [ ] Document: "VLM-RM Implementation Notes" (2-3 pages)

Key takeaways to document: - Exact reward computation: `reward = cos_sim(CLIP_vision(image), CLIP_text(text_prompt))` - Optional regularization: project onto goal-baseline direction - Text prompt engineering: single sentence descriptions work best - Task examples: MuJoCo humanoid (kneeling, splits, lotus)

Days 3-4: Sample Efficiency Landscape - [ ] Survey visual RL efficiency methods: - RAD (Random Augmentation for Data-efficiency) - DrQ-v2 (efficient off-policy learning) - CURL (Contrastive learning) - Lightweight architectures for vision - [ ] Understand metrics: sample efficiency curves, "area under curve" - [ ] Read RLiable paper on proper RL evaluation

Days 5: Knowledge Distillation Foundations - [ ] Study knowledge distillation basics: - Teacher-student loss: MSE between teacher and student outputs - Softmax temperature scaling - Why it works: student learns compressed representation - [ ] Review policy distillation literature (though different from our application) - [ ] Understand when distillation fails: distribution shift

Week 2: Environment Setup & Infrastructure

Days 1-2: Hardware Configuration - [ ] Configure 2x RTX 4090 environment - Check CUDA/cuDNN versions - Verify PyTorch GPU detection (should see 2x cards) - Monitor GPU memory (24GB each, ~45GB total usable) - [ ] Set up code organization: `vlm_reward_rl/ experiments/ # Results, logs, checkpoints src/ reward_models/ # Student/teacher reward architectures rl_agents/ # SAC/PPO implementations utils/ # Common utilities config/ # Hyperparameters scripts/ # Training scripts notebooks/ # Analysis notebooks`

Days 3-4: Dependency Installation - [ ] Create requirements.txt: `torch==2.1.0 torchvision==0.16.0 clip @ git+https://github.com/openai/CLIP.git dm-control stable-baselines3[extra] numpy matplotlib tensorboard wandb` - [ ] Verify installations: - [ ] PyTorch can access both GPUs - [ ] CLIP loads successfully (test: `clip.load("ViT-B/32")`) - [ ] DMC Suite runs (test: `python -c "from dm_control import suite; env = suite.load(domain_name='walker', task_name='walk');"`) - [ ] Stable-Baselines3 works (test SAC on CartPole)

Day 5: Experiment Tracking Setup - [ ] Create Weights & Biases account (free tier for research) - [ ] Configure WandB sweep templates for hyperparameter search - [ ] Document: baseline runs will log: - Reward per episode - Sample efficiency (steps to threshold) - Wall-clock time - GPU memory usage

Deliverable: Project skeleton, working environment, ready to run baseline code

---

Phase 2: Baseline Reproduction (Weeks 3-4)

Goal: Reproduce VLM-RM paper results, verify our setup matches published numbers

Week 3: CLIP Reward Query Infrastructure

Days 1-2: CLIP Setup & Benchmarking - [ ] Load CLIP model: `python` `import clip device = "cuda:0" model, preprocess = clip.load("ViT-B/32", device=device)` - [ ] Benchmark CLIP performance: - Single image encoding: measure latency (should be ~10-20ms) - Single text encoding: measure latency (should be ~5-10ms) - Full reward query (encode image, text, similarity): ~20-30ms - Batch processing (32 images): ~200-300ms total (~6-10ms/image)

Days 3-4: VLM-RM Reward Function Implementation - [ ] Implement CLIP reward computation: "'python class CLIPRewardModel: def **init**(self, prompt_text, baseline_text=None, alpha=0.5): self.model, self.preprocess = clip.load("ViT-B/32", "cuda:0") self.prompt_text = prompt_text self.baseline_text = baseline_text self.alpha = alpha

```
  def compute_reward(self, image):
      # Preprocess image, encode with CLIP, compute similarity
      image_features = self.model.encode_image(image)
      prompt_features = self.model.encode_text(clip.tokenize(self.prompt_text))
      reward = torch.nn.functional.cosine_similarity(image_features, prompt_features)

      if self.baseline_text:
          # Optional: project onto goal-baseline direction
          baseline_features = self.model.encode_text(clip.tokenize(self.baseline_text))
          # Implementation details in paper

      return reward.item()


- [ ] Test on standard benchmarks:
- CartPole: "a cart-pole system in balanced state"
- MountainCar: "a car at the top of the mountain"
- Walker (DMC): "a biped walking forward"

*Day 5: Verify Reward Signal Quality*
- [ ] Collect 1000 random trajectories from CartPole
- [ ] Plot: reward vs true environment return
- [ ] Measure correlation between CLIP reward and environment reward
- [ ] Document: "CLIP Reward is High Quality" (correlation > 0.8 expected)

**Week 4: Full RL Training with VLM-RM**

*Days 1-3: SAC Agent with CLIP Rewards*
- [ ] Implement SAC (or use Stable-Baselines3):
- Initialize with default hyperparameters
- Integrate custom CLIP reward function
- Log: episode reward, actual steps, time per step

- [ ] Reproduce on 3 core tasks:
- `Walker-walk`: Agent learns to walk forward
```

- `Cheetah-run`: Quadruped learns to run
- `Reacher-easy`: Arm learns to touch target

- [ ] Training protocol:
- 100K environment steps per task
- 5 random seeds per task
- Log every 1K steps
- Run on GPU 1 (leave GPU 2 free for other work)

Days 4-5: Results Analysis & Documentation
- [ ] Create plots:
- Learning curves (steps vs episode reward) for all 3 tasks
- Success rate at different thresholds
- Wall-clock time to convergence

- [ ] Compare with paper:
- Published results show ~X1000 steps to threshold on Walker-walk
- Does our reproduction match?
- Document discrepancies and reasons

- [ ] Profile GPU usage:
- Peak memory usage during training
- Percentage of time spent on reward queries vs RL updates
- Reward query latency histogram

Deliverable: Verified VLM-RM baseline matching paper results. Document: "Baseline Reproc

---

Phase 3: Reward Model Distillation (Weeks 5-7)

Goal: Train lightweight student reward models that mimic CLIP

Week 5: Offline Data Collection & Processing

Days 1-2: Trajectory Collection
- [ ] Create trajectory collection pipeline:
- Initialize weak policy (random exploration)
- Run 100 episodes per task (3 tasks = 300 episodes)
- Collect: state images, actions, rewards, next states
- Total: ~30,000 distinct states per task (~90K total)

- [ ] Dataset organization:
```python
# Store as: state_images.npz (90K x 84 x 84 x 3)
#           reward_labels.npz (90K,)  <- CLIP rewards
```

Days 3-4: CLIP Reward Labeling - [ ] Query CLIP on collected images: - Batch processing: 32 images at a time - Total queries: ~3000 batches = ~100 GPU-hours at 200ms/batch

GPU Time Distribution: Spread over week using GPU 2 - Run overnight: 8 hours → ~1440 queries - Do 3-4 overnight runs: cover most queries

- ☐ Parallel processing strategy:
  - Script 1: Collect trajectories on CPU (GPU 1 doing other work)
  - Script 2: Batch query CLIP on GPU 2
  - Both scripts read/write to shared disk

Day 5: Data Analysis - [ ] Statistics on collected dataset: - Reward distribution: histogram, mean, std - State diversity: visualize sample states from different episodes - Reward statistics per task

Week 6: Student Architecture & Training

Days 1-2: Architecture Design

Student reward model architecture:

```
class StudentRewardModel(nn.Module):
    def __init__(self, action_dim=None):
        super().__init__()
        # Lightweight CNN backbone (EfficientNet-B0)
        self.backbone = torchvision.models.efficientnet_b0(
            weights=torchvision.models.EfficientNet_B0_Weights.IMAGENET1K_V1
        )
        # Remove classification head
        self.backbone.classifier = nn.Identity()

        # Projection head for reward
        self.reward_head = nn.Sequential(
            nn.Linear(1280, 256),
            nn.ReLU(),
            nn.Linear(256, 1)
        )

    def forward(self, x):
        # x: (B, 3, 84, 84)
        features = self.backbone(x)  # (B, 1280)
        reward = self.reward_head(features)  # (B, 1)
        return reward
```

Architecture choices: - **EfficientNet-B0**: Light, pretrained on ImageNet, ~5M parameters - **Frozen backbone** (optional): Keep ImageNet features fixed, only train head - **Single scalar output**: Direct reward prediction - **Initialization**: Pretrained features give head start

Days 3-4: Training Pipeline

```
class RewardDistillationTrainer:
    def __init__(self, student_model, train_loader, val_loader):
        self.student = student_model
        self.optimizer = torch.optim.Adam(student.parameters(), lr=1e-4)
        self.criterion = nn.MSELoss()

    def train_epoch(self):
        self.student.train()
        total_loss = 0
        for images, labels in self.train_loader:
            predictions = self.student(images)
            loss = self.criterion(predictions, labels)
            self.optimizer.zero_grad()
            loss.backward()
            self.optimizer.step()
            total_loss += loss.item()
        return total_loss / len(self.train_loader)

    def validate(self):
        self.student.eval()
        total_loss = 0
        with torch.no_grad():
            for images, labels in self.val_loader:
                predictions = self.student(images)
                loss = self.criterion(predictions, labels)
                total_loss += loss.item()
        return total_loss / len(self.val_loader)
```

Training hyperparameters: - Batch size: 128 (fits in GPU memory) - Learning rate: 1e-4 (conservative, helps stability) - Epochs: 50 - Validation split: 80/20 train/val - Early stopping: patience 10 epochs - Expected training time: ~10 GPU-hours total on 1 GPU

Day 5: Preliminary Distillation Results

- ☐ Train student on one task (e.g., Walker-walk)
- ☐ Evaluate:
  - Train loss decay (should be smooth)
  - Val loss plateau (indicates convergence)
  - Visual analysis: plot predictions vs CLIP labels

Week 7: Multi-Task Distillation & Analysis

Days 1-3: Train all tasks

- ☐ Repeat distillation for all 3 tasks
- ☐ Results table:

```
Task          | CLIP Loss | Student Train Loss | Student Val Loss | Time
Walker-walk   | 0.0       | 0.052              | 0.068            | 8h
Cheetah-run   | 0.0       | 0.048              | 0.064            | 8h
Reacher-easy  | 0.0       | 0.055              | 0.071            | 8h
```

Days 4-5: Analyze Generalization

- ☐ Questions to answer:
  1. Do students generalize to held-out states? (yes = good)
  2. How does student perform on out-of-distribution states? (test with different random seed trajectories)
  3. Are certain types of states harder? (failure analysis)
- ☐ Visualization:
  - Scatter plot: student vs CLIP reward (should be ~linear)
  - Residual analysis: where are errors largest?

Deliverable: Trained student models for 3 tasks. Document: "Distillation Analysis Report"

Phase 4: Integration & Online RL (Weeks 8-10)

Goal: Use student rewards to train RL agents, compare with CLIP baseline

Week 8: Student-Based RL Training

Days 1-3: Modify RL Environment

Replace CLIP reward with student reward:

```python
class StudentRewardWrapper:
    def __init__(self, env, student_model, student_path):
        self.env = env
        self.student = student_model
        self.student.load_state_dict(torch.load(student_path))
        self.student.eval()

    def step(self, action):
        obs, _, done, info = self.env.step(action)
        image = obs['image']  # Assuming image observations

        with torch.no_grad():
            reward = self.student(image.unsqueeze(0)).item()

        return obs, reward, done, info
```

Days 4-5: Training Loop

```python
# For each task (Walker-walk, Cheetah-run, Reacher-easy):
#   For each seed (1, 2, 3, 4, 5):
```

```
#      1. Initialize SAC agent
#      2. Create StudentRewardWrapper(env, student)
#      3. Train for 100K steps
#      4. Log: episode reward, time, convergence plot
```

Week 9: Comparative Analysis

Days 1-3: Side-by-side comparison

Run parallel experiments: - GPU 1: SAC with CLIP rewards (3 tasks x 5 seeds) - GPU 2: SAC with Student rewards (3 tasks x 5 seeds)

Results table:

Task	CLIP Mean	Student Mean	Std(CLIP)	Std(Student)	Time(CLIP)	Time(Stu
Walker-walk	850	810	45	52	4.2h	0.8h
Cheetah-run	920	880	38	41	4.1h	0.8h
Reacher-easy	780	750	55	60	4.0h	0.7h

Days 4-5: Learning Curve Analysis

Plot for each task: - X-axis: environment steps (0 to 100K) - Y-axis: episode reward - Shaded region: mean $\pm$ 1 std across seeds

Key questions: 1. Does student match CLIP within statistical margin? (target: <5% gap) 2. Is convergence speed similar? (both should saturate around 50K steps) 3. Any tasks where student significantly underperforms? (red flags)

Week 10: Failure Analysis & Debugging

Days 1-3: Understand Gaps

If student underperforms: - Hypothesis 1: Student doesn't capture full reward complexity - Solution: Check train/val loss (high loss = poor approximation) - Try larger student architecture

- Hypothesis 2: Distribution shift during online RL
  - Solution: Analyze states encountered during RL vs offline data
  - Visualize: offline state embeddings vs online state embeddings
- Hypothesis 3: Exploration issues
  - Solution: Check if student reward is too smooth/noisy
  - Compare reward signal statistics (CLIP vs student)

Days 4-5: Results Documentation

Create comprehensive report: - Summary statistics table - Learning curve plots (one per task) - Failure case analysis - Recommendations for Phase 5

Deliverable: Benchmark comparing student vs CLIP rewards. Document: "Online RL Results Report"

Phase 5: Adaptive Refinement (Weeks 11-12)

Goal: Improve student performance via online adaptation

Week 11: Refinement Strategy Design

Days 1-2: Identify Distribution Shift

Analyze states encountered during online RL: - Collect 500 random states during student-RL training - Compute embedding distance from CLIP: - Extract CLIP image embeddings - PCA visualization: project to 2D - Measure: average distance between offline/online state distributions

Days 3-4: Refinement Algorithm

Implement adaptive refinement:

```python
class AdaptiveRewardRefiner:
    def __init__(self, student, clip_model, query_budget=10):
        self.student = student
        self.clip_model = clip_model
        self.query_budget = query_budget

    def should_refine(self, step, refine_interval=50000):
        return step % refine_interval == 0

    def select_states_to_query(self, online_states, query_budget):
        # Strategy: Select uncertain states
        with torch.no_grad():
            student_preds = self.student(online_states)

        # Heuristic: Select states where student predicts rewards near median
        # (high uncertainty in terms of training distribution)
        median_reward = torch.median(student_preds)
        uncertainties = torch.abs(student_preds - median_reward)

        top_indices = torch.topk(uncertainties, query_budget).indices
        return online_states[top_indices]

    def refine(self, selected_states):
        # Query CLIP on selected states
        clip_rewards = self.clip_model.compute_reward_batch(selected_states)

        # Fine-tune student on new data
        optimizer = torch.optim.Adam(self.student.parameters(), lr=1e-5)
        for _ in range(10):  # 10 gradient steps
            preds = self.student(selected_states)
            loss = F.mse_loss(preds, clip_rewards)
            optimizer.zero_grad()
```

```
        loss.backward()
        optimizer.step()
```

Day 5: Protocol Definition

Define when/how to refine: - **Refine interval**: Every 50K RL steps - **Query budget**: 10 CLIP queries per refinement - **Fine-tuning**: 10 gradient steps on new data - **Total overhead**: ~50 GPU-seconds per 50K RL steps (<0.1%)

Week 12: Online Refinement Experiments

Days 1-3: Refinement vs Non-Refinement

Run two conditions: 1. **Baseline-student**: No refinement, just student rewards 2. **Refined-student**: Student + adaptive refinement

Compare performance: - Does refinement improve convergence? - What's the "bang for buck"? (performance gain per CLIP query) - Is refinement necessary, or does student already generalize?

Days 4-5: Analysis & Writeup

Create comparison table:

```
Method              | Final Reward | Convergence Speed | Total CLIP Queries
Baseline-CLIP       | 850          | 50K steps         | 1,000,000
Baseline-Student    | 810 (95%)    | 50K steps         | 0
Refined-Student     | 835 (98%)    | 45K steps         | ~250
```

Deliverable: Adaptive refinement strategy with online results. Document: "Adaptive Refinement Report"

Phase 6: Comprehensive Ablations & Analysis (Weeks 13-14)

Goal: Understand design choices, ablate components

Week 13: Ablation Studies

Days 1-2: Vision Encoder Architecture

Train students with different backbones: - EfficientNet-B0 (current, ~5M params) - MobileNetV3 (lightweight, ~3M params) - ResNet18 (standard, ~11M params) - ViT-Tiny (transformer, ~6M params)

Measure: accuracy, training time, inference speed

Days 3-4: Distillation Data Size

Train students on varying amounts of CLIP labels: - 25% of data (~22K labels) - 50% of data (~45K labels) - 75% of data (~67K labels) - 100% of data (~90K labels)

Plot: x = data percentage, y = final RL performance

Key insight: How much data do you actually need?

Day 5: Text Prompt Engineering

Test prompt variations on same tasks: - Simple: "a robot walking" - Detailed: "a humanoid bipedal robot walking forward on flat ground" - Baseline included: Yes/No

Effect: Does prompt complexity change student performance?

Week 14: Multi-Task Transfer & Error Analysis

Days 1-2: Zero-Shot Transfer

Train student on Task A (e.g., Walker-walk), test on Task B (Walker-run): - Same environment family (different task) - Measure: Does student generalize without retraining?

Results table:

```
Train Task    | Test Task     | Transfer Success | Notes
Walker-walk   | Walker-run    | 85%              | Similar actions
Walker-walk   | Cheetah-run   | 45%              | Different morphology
Cheetah-run   | Reacher-easy  | 30%              | Very different task
```

Days 3-4: Failure Case Analysis

For tasks where student underperforms: - Visualization: Show image states where student fails - Hypothesis: What visual features is student missing? - Suggestion: Can we augment architecture?

Example analysis: - If student fails on "raised arms" in humanoid task: - Plot: Student predictions on raised-arms states - Compare with: CLIP predictions on same states - Diagnose: Is student architecture unlearning arm-related features?

Day 5: Comprehensive Summary

Create ablation summary table showing: - All variants tested - Performance comparison - Training cost - Recommendations for practitioners

Deliverable: Complete ablation study with analysis. Document: "Ablation Study Report"

Phase 7: Writing & Documentation (Weeks 15-16)

Week 15: Draft Paper Writing

Structure:

```
1. Introduction
   - Problem: VLM-RM is powerful but compute-heavy
   - Solution: Efficient distillation
```

```
   - Contribution: First study of efficient VLM-reward integration

2. Related Work
   - VLM-RM (Rocamonde et al., ICLR 2024)
   - Knowledge distillation
   - Efficient RL
   - Sample efficiency improvements

3. Method
   - Approach overview
   - Student architecture design
   - Distillation training protocol
   - Adaptive online refinement

4. Experiments
   - Baselines
   - Environments & metrics
   - Computational setup

5. Results
   - Baseline reproduction (Section 5.1)
   - Student vs CLIP comparison (Section 5.2)
   - Ablation studies (Section 5.3)
   - Failure analysis (Section 5.4)

6. Discussion
   - When does distillation work?
   - When does it fail?
   - Implications for practitioners
   - Future work

7. Conclusion
   - Summary
   - Impact
```

Days 1-3: Core sections (Methods + Results) Days 4-5: Introduction, Related Work, Discussion

Week 16: Finalization & Code Release

Days 1-2: Paper Polishing - [ ] Remove placeholder figures - [ ] Verify all citations - [ ] Check for consistency (notation, terminology) - [ ] Grammar/style pass

Days 3-4: Code Repository - [ ] Clean up codebase: - Remove debug prints - Add docstrings to all functions - Create README with installation & usage - Example notebooks: reproduce key experiments

- ☐ Directory structure:

```
vlm_reward_efficient_rl/
  README.md              # How to use
  requirements.txt       # Dependencies
  setup.py               # Install as package
  src/
    models/
      student_reward.py
      clip_reward.py
    agents/
      sac_agent.py
    utils/
      helpers.py
    config/
      defaults.yaml
  experiments/
    reproduce_baseline.py
    train_student.py
    evaluate_student_rl.py
  notebooks/
    01_baseline_analysis.ipynb
    02_distillation_visualization.ipynb
    03_rl_comparison.ipynb
  results/               # Figures, tables, logs
```

Day 5: Submission Preparation - [ ] Target conference: CoRL 2025 (June deadline) or AAAI 2026 - [ ] Format paper according to guidelines - [ ] Prepare supplementary materials: - Full hyperparameter tables - Additional experimental results - Code appendix - [ ] Create 2-minute video demo (if required)

Deliverable: Complete research paper + open-source code release

---

PART 3: KEY SUCCESS FACTORS & RISK MITIGATION

How to Ensure Success

1. Regular Checkpoints (Weekly) - End of every week, run unit tests on code - Verify training curves look reasonable - Check that GPU usage is as expected - Update shared experiment tracking (WandB)

2. Milestones with Go/No-Go Decisions

End of Week 4 (Month 1): - Baseline matches paper results? → YES: Continue | NO: Debug/pivot - CLIP query cost understood? → YES: Proceed | NO: Profile more

End of Week 8 (Month 2): - Student-RL achieves >90% of CLIP performance? → YES: Refine | NO: Rethink approach - Compute savings verified (>10x faster)? → YES: Publish | NO: Negative result (still publishable)

3. Parallel Work Stream - While GPU is training models (days 1-4 of week), work on writing/analysis (days 5-7) - Never idle—always making progress on some component

Risk Mitigation Strategies

Risk	Likelihood	Impact	Mitigation
Student doesn't generalize to online states	Medium	High	Implement adaptive refinement early (week 10)
CLIP queries take longer than estimated	Low	Medium	Batch processing + start early in week 5
Student training diverges	Low	Medium	Use conservative learning rates, early stopping
RL training fails with student rewards	Medium	High	Have fallback: use CLIP rewards in worst case
Compute insufficient (GPU runs out of memory)	Low	Medium	Use smaller batch sizes, enable gradient checkpointing
Results don't meet conference bar	Medium	Medium	Pre-plan workshop paper as backup

If Things Go Wrong

Scenario 1: Student doesn't match CLIP (< 80% performance) - Diagnosis: Check if MSE loss was high during distillation - Action: Retrain with larger student or longer training - Fallback: Switch to **Idea 3** (policy distillation instead)

Scenario 2: Online RL doesn't converge - Diagnosis: Is reward signal noisy? Is it even positive? - Action: Debug reward statistics, visualize predicted rewards - Fallback: Use CLIP for RL (expensive but works)

Scenario 3: Timeline slips - Diagnosis: Which phase is delayed? - Action: Reduce scope (fewer tasks, fewer seeds) - Fallback: Target workshop paper (faster feedback)

Scenario 4: Insufficient Compute - Diagnosis: Too many experiments planned - Action: Prioritize: Phases 1-4 are critical, Phases 5-6 are optional - Action: Use Kaggle free compute for ablations

---

PART 4: PUBLICATION STRATEGY

Positioning Your Paper

Title Options: 1. "Efficient Vision-Language Reward Models for Sample-Efficient Visual Reinforcement Learning" 2. "Lightweight Distillation of Vision-Language Reward Models" 3. "Making VLM-Based RL Practical: Efficient Distillation for Resource-Constrained Settings"

Key Message (One Sentence): "We show how to distill large vision-language models into efficient student reward models, achieving 10x speedup while maintaining >90% performance, making zero-shot RL accessible to resource-constrained researchers."

Target Venues (Priority Order)

Tier 1 (Primary): 1. **CoRL 2025** - June deadline, December conference (Robotics focus) - Perfect fit: RL + robotics manipulation - Similar prior work on VLM-RMs presented here - Submission: April-May 2025

2. **ICLR 2026** - September deadline, May conference (ML theory/practice)
   - Broad audience interested in RL efficiency
   - Submission: June-July 2025

Tier 2 (Backup): 3. **AAAI 2026** - August deadline, February conference (General AI) - Wider audience than specialized venues - Submission: May-June 2025

4. **NeurIPS Workshop** - October deadline, December conference
   - Faster feedback if main conference not accepted
   - Can focus on novel aspects (e.g., adaptive refinement)

Paper Strengths to Emphasize

1. **Practical Impact**: Makes cutting-edge RL accessible (democratization)
2. **Rigorous Evaluation**: Comprehensive ablations, error analysis
3. **Clear Contribution**: First to study efficient VLM-reward distillation

4. **Reproducibility**: Open-source code, detailed hyperparameters
5. **Resource Consciousness**: Designed for undergrads/limited compute

Expected Reviewer Comments (& Rebuttals)

Comment: "Distillation is well-studied, not novel" **Rebuttal**: First application to VLM reward models; shows domain shift handling unique to rewards vs policies

Comment: "Only tested on simulation" **Rebuttal**: DeepMind Control Suite is standard benchmark; future work on real robots

Comment: "Why not just use CLIP directly?" **Rebuttal**: 10x slower inference; impractical for resource-constrained settings (which we support with data)

Comment: "Performance gap vs CLIP is concerning" **Rebuttal**: 5-10% gap is acceptable tradeoff for 10x speedup; adaptive refinement further reduces gap

---

PART 5: TIMELINE GANTT CHART

```
Week:     1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16
Phase 1:  |===========| Literature & Setup
Phase 2:              |===========| Baseline Reproduction
Phase 3:                        |====================| Distillation (offline)
Phase 4:                                    |====================| Integration & Online RL
Phase 5:                                              |===========| Adaptive Refinement
Phase 6:                                                    |==============| Ablations
Phase 7:                                                              |===========| Writing

GPU Usage:
- GPU 1: Mostly training (Phase 2-4)
- GPU 2: Mostly data collection & analysis (Phase 3, 5-7)
- Both: Can run parallel experiments in Weeks 8-10

Critical Path: Phase 1 → Phase 2 → Phase 3 → Phase 4 (must be sequential)
Parallel: Phase 5-6-7 can overlap with Phase 4 training
```

PART 6: PAPER OUTLINE (DETAILED)

1. Introduction (800 words)

Hook: Vision-language models (VLMs) like CLIP have revolutionized task specification in RL, enabling zero-shot reward learning from natural language. However, large VLMs are computationally expensive…

Problem: VLM-RM requires expensive CLIP queries during training, making it inaccessible to researchers with limited compute.

Solution: We show efficient student-teacher distillation of VLM rewards.

Contributions: 1. First systematic study of efficient VLM-reward integration 2. Lightweight distillation achieving 10x speedup 3. Adaptive refinement strategy for online distribution shift 4. Open-source implementation enabling undergraduates

2. Related Work (1000 words)

Vision-Language Models - CLIP architecture & pretraining - Prior uses in RL (VLM-RM, RL-VLM-F, VLAC)

Sample Efficient RL - Data augmentation (RAD, DrQ-v2) - Contrastive learning (CURL) - Representation learning

Knowledge Distillation - Standard knowledge distillation - Policy distillation in RL - Online distillation

Efficient Deep Learning - Model compression - Lightweight architectures - Edge deployment

3. Method (1200 words)

3.1 VLM-RM Baseline - CLIP architecture recap - Reward computation: cosine similarity - Baseline-goal regularization

3.2 Problem Formulation - Naïve VLM-RM: expensive - Goal: Approximate with efficient student

3.3 Approach: Distillation Pipeline

- Offline Phase: Collect diverse trajectories, query CLIP
- Distillation Phase: Train student on CLIP labels
- Online Phase: Use student for RL
- Refinement Phase: Adapt student during online RL

3.4 Student Architecture - EfficientNet-B0 backbone - Reward head (MLP) - Frozen vs. trainable backbones

3.5 Adaptive Refinement - Problem: Distribution shift during RL - Solution: Periodic CLIP queries on uncertain states - Uncertainty heuristic: distance from median prediction

4. Experiments (1500 words)

4.1 Setup - Environments: 3 DMC tasks (Walker-walk, Cheetah-run, Reacher-easy) - Metrics: episode reward, convergence speed, GPU time - Baselines: CLIP-RM, random, hand-crafted rewards - Seeds: 5 per configuration

4.2 Baselines - Baseline 1: Original VLM-RM (full CLIP queries) - Baseline 2: Student without refinement - Baseline 3: Ground-truth rewards (upper bound) - Baseline 4: Random rewards (lower bound)

4.3 Metrics - Sample efficiency: area under learning curve, steps to threshold - Compute cost: GPU-hours, wall-clock time - Reliability: success rate across seeds

5. Results (2000 words)

5.1 Baseline Reproduction - Figure: VLM-RM learning curves match paper - Table: Numbers vs. published results - Confirm: CLIP queries are indeed expensive

5.2 Student Distillation Results - Table: MSE loss on train/val sets - Analysis: Does student generalize? - Visualization: Scatter plot (student vs CLIP predictions)

5.3 Online RL Evaluation - Main result table: CLIP vs Student performance - Learning curves (mean $\pm$ std) - Success in achieving >90% of CLIP performance - Speedup achieved (10x on total compute)

5.4 Ablation Studies - Vision architectures (EfficientNet vs ResNet vs ViT) - Data efficiency (how much CLIP labels needed?) - Prompt engineering (simple vs complex) - Frozen vs trainable backbone

5.5 Adaptive Refinement - Does refinement improve performance? - Trade-off: performance gain vs CLIP queries used - Recommendation: when to use refinement

5.6 Error Analysis - Which tasks does student perform best on? - Which visual features drive student mistakes? - Failure modes analysis

6. Discussion (1200 words)

6.1 When Does Distillation Work? - Smooth reward landscapes - Diverse offline data - Distribution shift problems

6.2 Practical Implications - Makes VLM-RL accessible - Democratization of advanced methods - Environmental benefits (reduced compute)

6.3 Limitations - Only simulation experiments - Limited to visual observations - Student architecture choices not exhaustive

6.4 Future Work - Real robot experiments - Multi-modal reward models - Distillation of other foundation models

7. Conclusion (400 words)

Summarize contributions, impact, and vision for future work.

FINAL CHECKLIST

- ☐ Complete all 7 phases on schedule
- ☐ Baseline matches VLM-RM paper
- ☐ Student achieves >90% of CLIP performance
- ☐ 10x compute speedup verified
- ☐ All ablations documented
- ☐ Paper written and polished
- ☐ Code released on GitHub
- ☐ Submitted to target venue (CoRL/AAAI)

You've got this!

This plan is ambitious but achievable for a dedicated undergraduate team with 2x4090 GPUs over 4 months. The research is novel, timely, and impactful. Good luck!