How to Integrate Reinforcement Learning on LLM Agents

Presenter: Ibrahim Al Azher

Outline

1. Foundation of LLM Agents
2. Bottleneck of LLM Agents
3. Reinforcement learning (RL) concepts
4. Difference between RL on LLM vs RL on LLM Agents
5. How to integrate RL on LLM
6. How to integrate RL on LLM Agents
7. RL on LLM Agents in HPC systems (High performance computing)

[1] SAND boosting LLM agents with self taught action deliberation (chat: SAND for limitation generation)

Foundation of Multi-Agent Orchestation

Agent Communication & Collaboration

- **Role Specialization:** Agents are segmented into functional units, moving past single, all-purpose prompts.
- **"State" as History:** The "State" is the compilation of all previous agent turns, context, and evidence in multi-turn systems.
- **Communication Topology:**
  - **Sequential/Chain:** A -> B -> Master.
  - **Hierarchical (Leader-Worker):** Leader directs tasks, provides feedback, and requests revisions based on Uncertainty Gates.
  - **Self-Consistency:** Each agent explores various paths for self-consistency.
- **Deliberation Hooks:** Inspired by SAND [1], agents sample N candidates and only "think" (reason) when candidates conflict (using Self-Consistency as the trigger).

[1] SAND boosting LLM agents with self taught action deliberation (chat: SAND for limitation generation)

Communication Topology

Sequential/Chain

Worker A → Worker B → Master.

Hierarchical (Leader-Worker)

The Leader routes tasks, provides feedback, and calls for revisions based on Uncertainty Gates.

Shared Memory

Concept: A "Shared Workspace" for agents that persists across turns, preventing information silos and giving the Leader full visibility.

- **The Blackboard:** A structured JSON or Database (Redis/Object Store) where agents post "Limitation Units" (LUs).
- **Access:** Workers are Write/Append-Only to their sections; the Master/Leader has Read/Write access to the whole board.
- **Version Control:** Tracks changes, storing both Draft and Revised versions of limitations.
- **Importance:**
  - **Consistency:** Avoids contradictions between specialized agents (e.g., Experimental vs. Theory).
  - **Context Management:** Reduces Context Window bloat; agents only pull relevant snippets.
  - **Conflict Resolution:** Central hub for the Master Agent to de-duplicate and resolve contradictions.

HPC Implementation: Ray's Object Store functions as the physical shared memory for sharing large tensors and trajectories across GPU nodes efficiently.

The Bottlenecks of Agentic Systems

Why Multi-Agent Systems Fail

- **Master-Worker Mismatch:** Master agents often "hallucinate" consensus, omitting critical minority critiques during the final merge.
- **The Over-Commitment Trap:** Agents commit to the first generic limitation (e.g., "more data needed") without deep deliberation or structural critique.
- **Feedback Loops & Redundancy:** Agents repeat points for "coverage," resulting in verbose, low-signal outputs.
- **The Credit Assignment Problem:** Difficulty determining which agent failed (e.g., Agent A's lack of evidence vs. Agent B's poor feedback) when the final output is poor.
- **Groundedness Gaps:** Agents generate responses lack support from the specific source text ("Stochastic Parrot" effect).

How to overcome

- Integrating verifier agent: verify each agent's response, provide score in terms of groundness, specificity, coverage, non hallucinations, usefulness.
- When to stop: one approach can be if the performs doesn't increase more than 10%.
- Verify with external source: Leader agent can access tools, RAG for verification

Reinforcement Learning

1. **RLHF (Reinforcement Learning from Human Feedback):** Foundational framework for LLM alignment. Trains a separate Reward Model (RM) on human preferences (A vs. B comparisons). The LLM is optimized against this RM using an RL algorithm (e.g., PPO).
2. **RLAIF (Reinforcement Learning from AI Feedback):** Identical to RLHF, but uses a "Teacher" LLM (like GPT-4) instead of human labelers to rank outputs based on a rubric.
    - *Relevance:* This mirrors using a Verifier Agent to score worker agents.
3. **PPO (Proximal Policy Optimization):** An "actor-critic" RL algorithm. Requires maintaining four models: Actor (training), Reference (frozen), Critic, and Reward Model. Uses a "clipped" objective to limit model changes per update.
    - *Bottleneck:* Extremely memory-intensive, requiring massive HPC clusters for large models (e.g., 70B).

Reinforcement Learning

1. **DPO (Direct Preference Optimization):** Eliminates the Reward Model and Critic. Derives optimal policy directly from the log-ratio of "chosen" vs. "rejected" pairs. The LLM acts as its own reward model. Used here to create preference pairs (higher-score = chosen, lower-score = rejected).
2. **GRPO (Group Relative Policy Optimization):** Used for DeepSeek-R1; designed for Reasoning/Structured Tasks. Eliminates the Critic. Works by sampling a group of outputs (e.g., K=4) and calculating the group's average reward. Outputs better than the average are reinforced; worse outputs are penalized. Chosen for being more memory-efficient than PPO and better for structured tasks.
3. **KTO (Kahneman-Tversky Optimization):** Unlike DPO, it only requires "Good" or "Bad" labels, not ranked pairs. Based on Prospect Theory (humans weigh losses heavier than gains). Benefit: Simplifies data collection as only binary labels (Binary Reward) are needed.

RL Fundamentals (DPO & GRPO)

Core Equations and Objectives

- **Direct Preference Optimization (DPO):** Eliminates the need for a separate Reward Model by using the LLM itself. It optimizes the log-ratio of chosen (y_w) vs. rejected (y_l) completions.
    - **Loss Function:**

$$\mathcal{L}_{DPO}(\pi_\theta; \pi_{ref}) = -\mathbb{E}_{(x,y_w,y_l)\sim D}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)}\right)\right]$$

- **Group Relative Policy Optimization (GRPO):** Optimized for mathematical/structured reasoning (like DeepSeek-R1). It replaces the Critic model with a group-relative mean.
    - **Advantage Calculation:** For a group of G outputs:

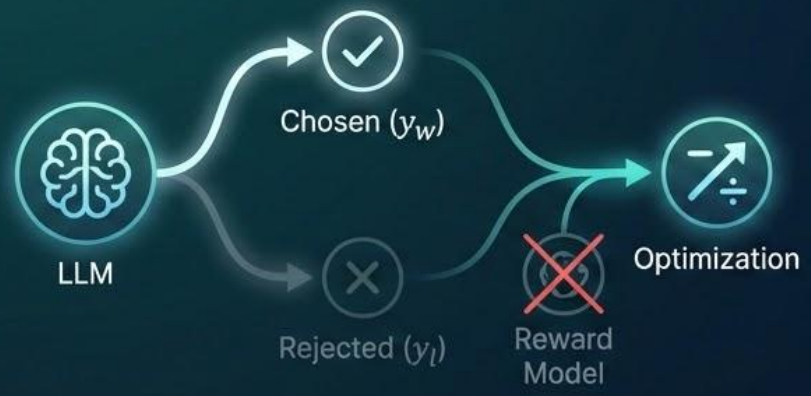
$$A_i = \frac{r_i - \text{mean}(r_1, \ldots, r_G)}{\text{std}(r_1, \ldots, r_G)}$$

    - **Objective:**

$$\mathcal{L}_{GRPO}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\left(\min\left(\frac{\pi_\theta(a_i|s)}{\pi_{old}(a_i|s)}A_i, \text{clip}\left(\frac{\pi_\theta(a_i|s)}{\pi_{old}(a_i|s)}, 1-\epsilon, 1+\epsilon\right)A_i\right)\right.\right.$$

RL Fundamentals (DPO & GRPO)

Core Equations and Objectives



Direct Preference Optimization (DPO)

Eliminates the need for a separate Reward Model by using the LLM itself. It optimizes the log-ratio of chosen (y_w) vs. rejected (y_l) completions.

Group Relative Policy Optimization (GRPO)

Optimized for mathematical/structured reasoning (like DeepSeek-R1). It replaces the Critic model with a group-relative mean.

RL on LLM vs. RL on LLM Agents

The "Brain" vs. The "Body and Hands"

- **RL on LLM (Traditional Alignment):**
  - **Goal:** Optimize single-turn response quality (Helpfulness, Honesty, Harmlessness).
  - **Horizon:** Single-step Markov Decision Process (MDP). One prompt → One completion.
  - **Focus:** Linguistic style, instruction following, and safety.
- **RL on LLM Agents (Agentic RL):**
  - **Goal:** Optimize multi-turn goal achievement (Tool-use, Feedback loops, Strategy).
  - **Horizon:** Temporally extended POMDP. Trained on Trajectories.
  - Prompt → [Action → Observation ×N] → Goal.
  - **Focus:** Communication efficiency, recovery from errors, and cross-agent coordination

SFT for Agents: Bootstrapping the "Protocol"

Learning How to Talk Before Learning How to Win

- **Why SFT is Necessary for Agents:** RL is a "sparse reward" problem. If the model doesn't know *how* to call a tool or format a feedback message, it will never find the reward.
- **The SFT Dataset for Communication:**
  - **Protocol Alignment:** Training on structured JSON or specific tags (e.g., <thought>, <critique>, <action>).
  - **Communication Style:** SFT on high-quality teacher trajectories (e.g., GPT-4o acting as a Leader).
  - **Imitation Learning:** The model learns the "language" of the multi-agent system (how to accept/reject worker output).
- **The Ceiling:** SFT only makes the model as good as the training data. It doesn't teach **strategy** or **robustness** to environment errors

RL for Agents: Optimizing the "Tactics"

Beyond Imitation: Learning through Trial and Error

- **Strategic Communication:** Using RL to punish "lazy" feedback. If a Leader gives generic feedback and the Final Score stays low, the Leader is penalized.
- **Trajectory Optimization:**
  - **Recovery:** RL teaches agents to "pivot" when a worker agent hallucinates.
  - **Efficiency:** Reward models that reach high-quality results in 2 turns instead of 5 (Verbosity/Token penalties).
- **Group Dynamics (GRPO):**
  - By sampling K different "communication paths" for the same paper, the model learns which **mode of communication** consistently yields a higher verifier score.

RL on LLM vs LLM Agents

Differences

Feature	RL on LLM (Chat)	RL on LLM Agents (Scientific System)
Input State	Current User Prompt	Prompt + Tool History + Cross-Agent Chat
Action Space	Tokens in a single block	API Calls, Search queries, Revisions
Reward Signal	Subjective (Human Rank)	Objective (Verifier Score, Evidence Match)
Credit Assignment	Immediate (on the text)	Delayed (broadcast back to earlier turns)
Risk	Repetitive/Boring text	"Reward Hacking" (e.g., making lists longer to hide errors)

RL on LLM Agents

Workflow of RL on LLM based agents

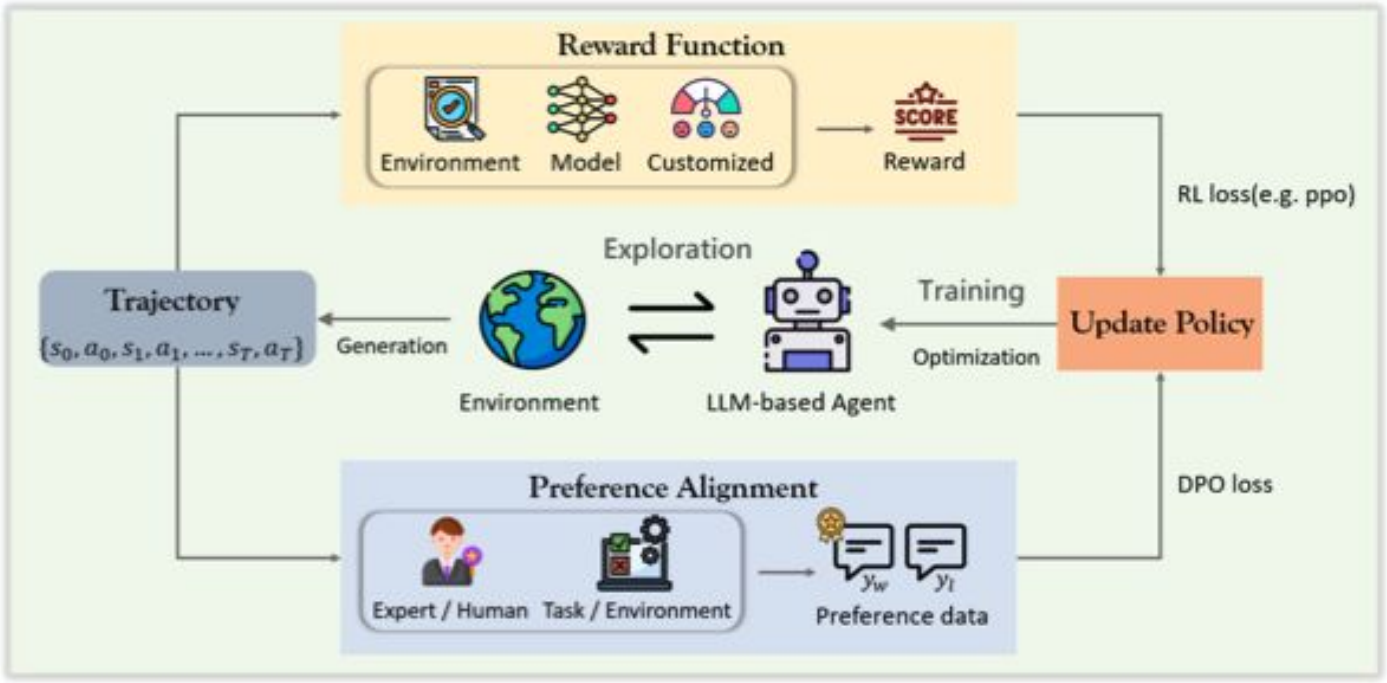


Fig. 3. Workflow of reinforcement learning-based optimization for LLM-based agents.

LLM Agents (Single Rollout) (Example)

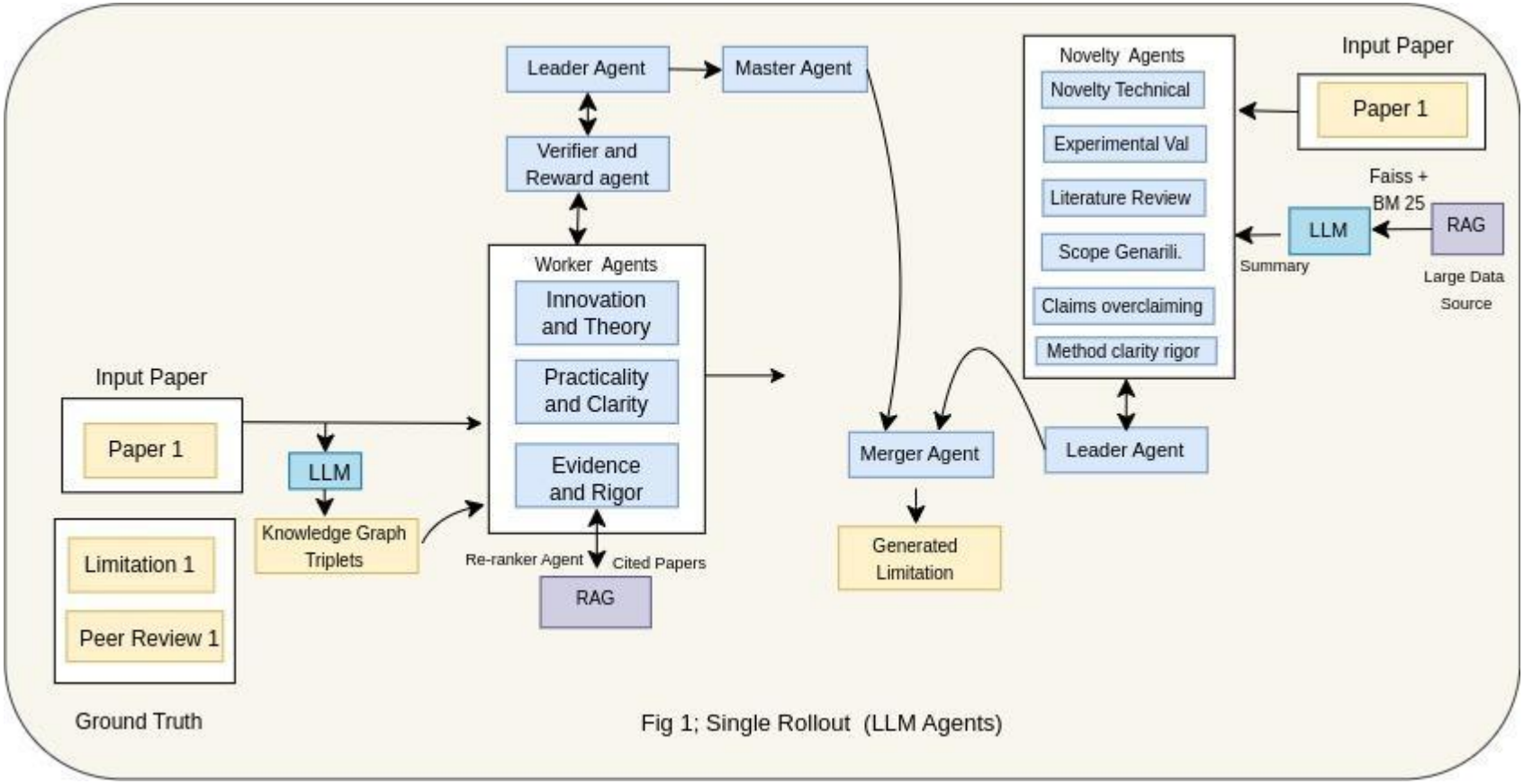


Fig 1; Single Rollout (LLM Agents)

Reinforcement learning and Inference

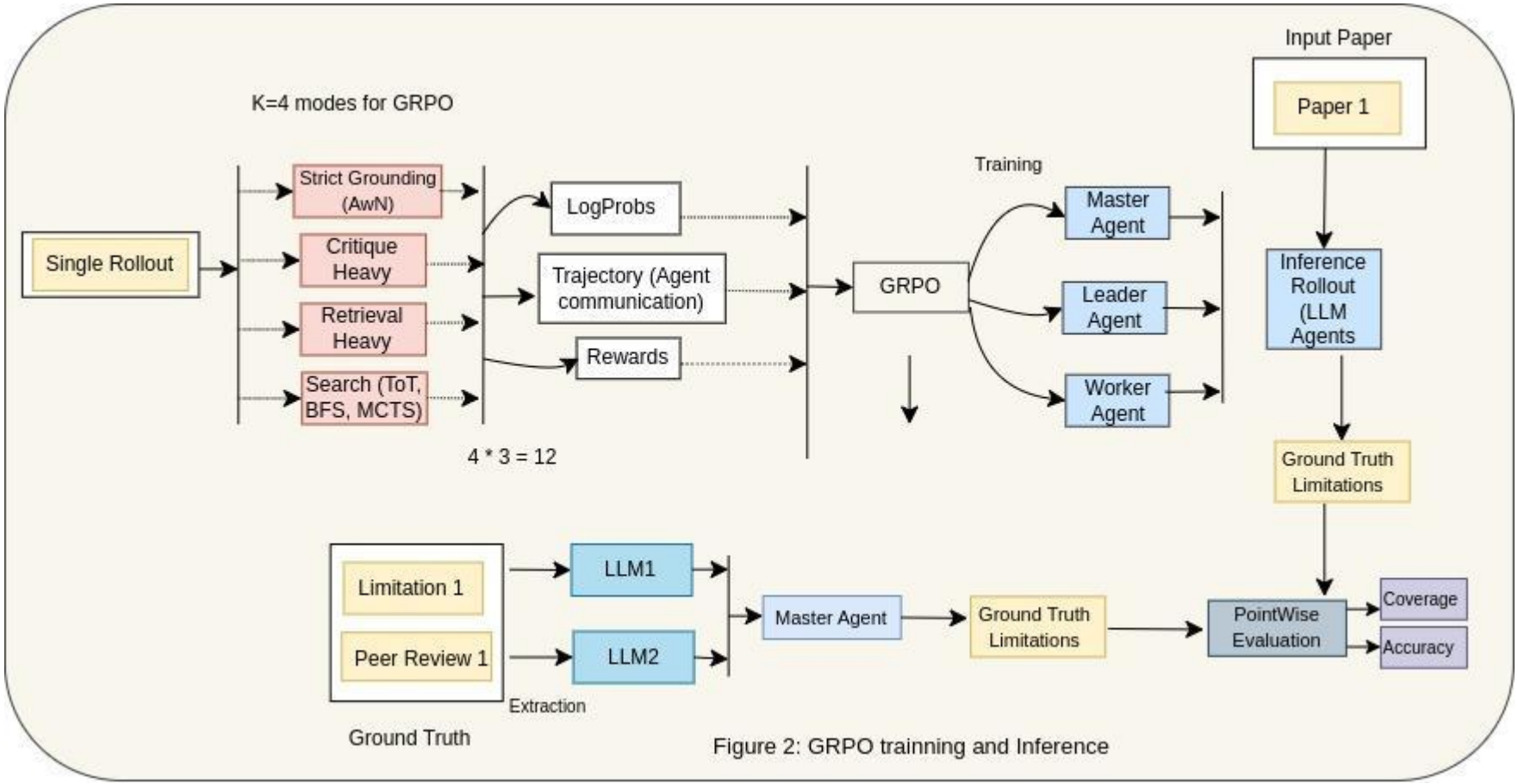


Figure 2: GRPO trainning and Inference

RL in Multi-Agent Trajectories

Credit Assignment & Influence-Aware Rewards

RL can be trained on Two ways:

- **On policy: Reference log probs captured from current model before gradient step**
- **Offline policy: first apply rollout, collect trajectories, logprobs, then apply RL, save LoRA adapter for each types of agents for inference.**
- **Trajectory Definition:** A sequence of
  - Agent_ID: unique ID
  - Prompt
  - Output: Agent's responses
  - Logprob: Token probabilities

Verification Reward Model

- **Influence-Aware Verification (MAPoRL2 style [2]):** An agent's reward Rt is not just its current score, but its "Future Influence."
  - Rt = Immediate_Score + γ(Improvement in Final Master Output).
- **Revision Trajectories (Agent-R style [3]):** Learning from failure.
  - Find the **First Error Step** (where the Verifier flags a hallucination).
  - Splicing: Combine the "Bad Prefix" + "Revision Signal" ("I realize I was wrong...") + "Good Suffix."
  - This teaches the Leader agent **Recovery**, not just perfection.

MAPoRL2 multi agent post co-training for collaborative large language models with reinforcement learning [2]
Agent R training language model agents to reflect via iterative self-training [3]

Practical Engineering with LLM

Memory, QLoRA, and the GRPO Loop

- **Memory Efficiency:**
  1. Base Llama-3 70B (4-bit NF4 via bitsandbytes) takes ~35GB.
  2. **Bfloat16 Adapters:** Keep gradients in higher precision to avoid training instability.
- **The Logprob Trap (The "On-Policy" Requirement):**
  1. **Step 1 (Rollout):** Collect 4 candidates from π_old. Save the tokens and their **frozen logprobs** as scalars. Compute logp_old = log p_theta_old (response | prompt) under a frozen old policy (from rollout)
  2. **Step 2 (Update):** Re-compute logprobs with the live model πθ (gradients enabled). Ratio = exp (logp_new - log_old), here 'logp_new' comes from newer model.
  3. The gradient comes from the ratio growing as πθ moves away from the frozen πold. Model is trained on gradients from above average and updates weights.

Verifier Rubric

- **Verifier Rubric:**
  - **Groundedness:** NLI-based check against paper spans.
  - **Specificity:** Penalty for generic phrases (e.g., "further research").
  - **Adversarial Penalty:** Deduct points for agents that "agree" too easily without checking citations.

The "Agent-Specific Adapter" Strategy

Optimization Goals for Specialized Heads

- **Worker Adapter:** Optimized for **Token-Level Factuality**. High penalty on hallucination and redundant descriptions.
- **Leader Adapter:** Optimized for **Instruction Following & Routing**. Its reward is tied to the *efficiency* of the loop (reaching stability in fewer turns).
- **Master Adapter:** Optimized for **Information Density**. Learned ability to de-duplicate while maintaining the "harshest" valid critiques from workers.
- **Stopping Criterion:** $\Delta\text{Score}<10\%$ across rounds. This prevents "Reward Hacking" where agents keep talking to artificially inflate verbosity.

High-Throughput Rollout Layer

vLLM + Ray Orchestration

- **The Orchestration Layer (Ray):**
  - Manages distributed scheduling of the 7-agent pipeline.
  - **Parallel Execution:** Instead of sequential agent calls, Ray fires the multiple specialist agents + Master in parallel.
  - **Resource Management:** Assigns fractional GPUs (e.g., 0.5 GPU) per agent instance to maximize A100 utilization.
- **The Generation Engine (vLLM):**
  - Used for the **Rollout Phase** only (fast sampling).
  - Two types: a) AWQ Quantized Llama 3 (inference only); b) Bfloat16 with Qlora.
  - With AWQ Quantized: use offline policy (efficient in terms of vRAM). Only used for frozen old policy. Have to save trajectories, rewards, logbrobs in jsonl
  - With Bfloat16 with Qlora (PEFT): use online policy. Memory intensive. Better approach (true on policy)
  - **Model Mixing: For example** "Behavior Policy" to generate K=48 candidates (3 agents × 4 modes × 4 samples) per paper (each samples comes with self consistency or different temperatures.

The GRPO Training Loop

From Rollouts to Weight Updates

- **Logprob Extraction:** Calls /v1/chat/completions with logprobs=True. Fallback: Uses tokenizer.apply_chat_template() for sum logprob calculation on the fly.
- **The Dataset (rollouts.jsonl):**
  - Stores the trace of every agent turn: [Prompt, Trajectory, Logprobs_old, Reward_breakdown].
  - Includes metadata like node_id and parent_id to reconstruct the multi-agent graph.
- **Advantage Calculation (The "Group" in GRPO):**
  - **Group Baseline:** Grouped by (Agent, Paper, Mode).
  - Advantage= Std(Group_Rewards) + ϵReward − Mean(Group_Rewards)
  - **Optimization Signal:** High rewards + low P_old generate the strongest push for probability updates.
- **KL Divergence Control:** GRPO integrates KL directly into the loss to ensure the model doesn't "collapse" into a single repetitive response style.

Implementing the Clipped Objective

Math and the "New vs. Old" Ratio

- **The Policy Ratio ($r_t(\theta)$):**
  - **log(P_old):** Retrieved from the frozen rollouts.jsonl (generated by the AWQ/vLLM behavior policy). Or 'on policy' at first it calculates the 'P_old' as a 'rollout' phase.
  - **log(P_new):** Computed "on-the-fly" using the Hugging Face **HF + QLoRA** model currently being updated.
  - $r_t(\theta) = \exp(\log(P_{new}(y|x)) - \log(P_{old}(y|x)))$
  - Model will updated the weight and biases from above average.
- **PPO-Clipped Loss:**
  - $L(GRPO) = -E[\min(r_t(\theta) \cdot \hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot \hat{A}_t)]$
- **Update Strategy:**
  - If $Adv > 0$: The loss encourages $r_t(\theta)$ to increase (making the good behavior more likely).
  - If $Adv < 0$: The loss encourages $r_t(\theta)$ to decrease.

Inference

Targeted Fine-Tuning Goals

Stored multiple LoRA based adapter after training with GRPO for different LLM Agents.

- **Worker Adapter:** Focuses on **Synthesized Extraction**.
  - *Reward Signal:* High score on Groundedness and Evidence Rigor.
- **Leader Adapter:** Focuses on **Strategic Feedback**.
  - *Reward Signal:* Improvement in worker scores after one round of feedback.
  - *Constraint:* High penalty for vague feedback like "make it better."
- **Master Adapter:** Focuses on **Consolidation**.
  - *Reward Signal:* Non-redundancy and high Precision/Recall relative to the Ground Truth (author limitations).
- **Inference Strategy:** At test time, the base model loads the specific adapter (Worker/Leader/Master) dynamically depending on which role is active in the pipeline.

HPC Workflow & Infrastructure

Scaling with PBS and Multi-GPU Clusters

- **Stage 1: Distributed Rollout (Offline Data Collection):**
  - HPC job spins up a Ray cluster + vLLM server.
  - Processes 1000s of papers to generate rollouts.jsonl.
- **Stage 2: QLoRA Training (Update Phase):**
  - Have to use multi node clusters (with Ray)
  - Uses **Transformers + PEFT** on BF16 precision.
  - **Gradient Accumulation:** Used to simulate larger batch sizes on 40GB A100s.
  - **Memory Tip:** Gradient checkpointing is essential when training the 70B model with a 1M context window (gpt-4.1-mini level expectations).
- **Stage 3: Evaluation & Convergence:**
  - Monitors the **10% Improvement Floor**. If the Leader-Worker interaction score plateaus, the job auto-terminates to save compute credits.