

Neuro-Symbolic Agentic Reinforcement Learning for Long-Term Original Character Companionship and Interaction

Zhenhan Huang

Neurobo AI

zhenhan.email.jp@gmail.com

Abstract

As human-agent interaction (HAI) evolves toward long-term social companionship, users expect *Original Character (OC)* agents to maintain a consistent persona, manage shared memories, and adapt to ever-changing preferences. However, LLM-based agents optimized by prompting or SFT exhibit a generalization gap: they behave as myopic instruction followers, leading to cascading errors in multi-turn interactions. For the agents to learn trajectory-level value functions that enable farsighted decision-making, we propose the NSARL framework, which formalizes OC companion agents' interactions as a POMDP and decomposes the agent into three sub-policies (Router, Memory, and Persona), optimized via closed-loop RL from AI feedback (RLAIF) with verifiable rewards in a graph-constrained action space. Our preliminary experiments indicate a trade-off: SFT yields stronger persona generation, while NSARL improves structural logic, through conservative strategies (e.g., over-routing) that increase workflow completeness, advocating for a hybrid deployment strategy.

1 Introduction

As human-agent interaction (HAI) evolves from transactional assistants to long-term social companions, users demand agents that maintain a personalized, consistent *Original Character (OC)* persona over time. Such companions must proactively manage shared memories, track evolving preferences, and sustain suspension of disbelief over long-horizon interactions. Yet, current large language model (LLM)-based agents, primarily optimized by prompt engineering or supervised fine-tuning (SFT), face a critical generalization gap in these settings (Zhang et al., 2025a; Kim et al., 2024).

While the existing methods perform well on multi-turn dialogue benchmarks (Duan et al., 2023),

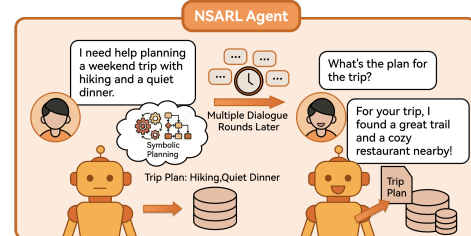
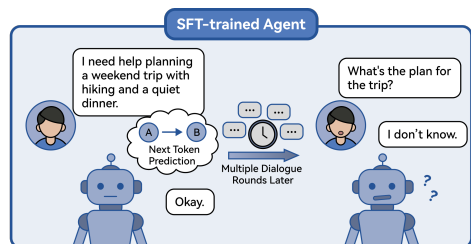


Figure 1: Comparison between traditional and our agent paradigms.

they still struggle in long-horizon, socially rich interactions (Mahowald et al., 2024). Surveys note that SFT largely reduces to single-step optimization (i.e., RL with horizon=1, $\gamma = 0$): it trains trajectory imitation without learning dynamics for robust multi-step control (Zhang et al., 2025a). Consequently, as illustrated in Figure 1, three gaps follow:

(1) **Myopic Decision-Making:** Agents optimize immediate likelihood over long-horizon quality, failing to recognize that “lazy” decisions cause long-term amnesia (Luo et al., 2025); (2) **Cascading Errors:** Trained on optimal trajectories, SFT agents lack error recovery mechanisms; and (3) **Static Persona vs. Dynamic Context:** Static prompts fail to capture evolving user preferences or proactive memory management.

In this work, we bridge the above-mentioned gaps with a Neuro-Symbolic Agentic Reinforcement Learning framework for OC companion agents (NSARL). We formalize the OC agent's operation as a partially observable Markov deci-

estimation through retrieval into working memory.

2.2 Graph-Constrained Policy Architecture

We represent OC agent’s decision-making as a directed cyclic graph \mathcal{G} of functional nodes with *LangGraph*, shown in Figure 2.

2.2.1 Sub-Policy Nodes

The **trainable** nodes include: **Router** ($\pi_{\theta}^{\text{router}}$), which maps observation o_t to an execution plan p_t ; **Memory Agent** (π_{ϕ}^{mem}), which selects a memory operation (action, target, query, content); and **Persona Agent** (π_{ψ}^{text}), which synthesizes user input, vision output, and memory results into the final response.

2.2.2 Deterministic Nodes

Complementing the trainable modules, three deterministic nodes serve as the stable environment interface: the **Orchestrator** enforces valid execution through rule-based plan rectification (e.g., repairing missing memory ops); the **Memorizer** manages database I/O and links location metadata; and the **Vision Agent** employs VLMs (e.g., GPT-4o-mini) to extract grounded scene semantics.

2.3 Policy Optimization

We employ Group Relative Policy Optimization (GRPO) with Agent Lightning (Luo et al., 2025) for credit assignment. Instead of end-to-end training, we perform modular optimization with RLVR:

2.3.1 Targeted Rollouts

To ensure the target module is executed during training (even if an untrained Router would skip it), we use *forced plans*; e.g., for Memory Agent training we inject `force_plan = [memory_agent, memorizer, text_agent]`.

2.3.2 Reward Functions

We compute rewards with a Judge LLM using objective and verifiable criteria:

Router Reward (R_{router}): Optimized for plan completeness and efficiency.

$$R_{\text{router}} = \max(0, \mathbb{I}(\mathcal{P} \supseteq \mathcal{P}^*) \cdot 1.0 - \beta \cdot |\mathcal{P} \setminus \mathcal{P}^*| - \delta \cdot \mathbb{I}(\text{OrderErr})) \quad (1)$$

where \mathcal{P} is the predicted plan and \mathcal{P}^* is the ground truth. We set $\beta = 0.2$ (extra-agent penalty) and $\delta = 0.3$ (order penalty); incomplete plans receive zero reward to enforce compliance.

Memory Reward (R_{mem}): A two-stage verifiable metric consisting of a binary gate and a semantic score function $\mathcal{S}(\cdot, \cdot)$:

$$R_{\text{mem}} = \mathbb{I}(a_{\text{type}} = a_{\text{type}}^*) \cdot \mathcal{S}(c_{\text{pred}}, c^*) \quad (2)$$

Here, a_{type} and c_{pred} are predicted action type and content, and a_{type}^* and c^* are ground truth. \mathcal{S} is approximated by a Judge LLM with discrete scores based on semantic overlap (1.0: identity, 0.8: paraphrase, 0.5: missing entities, 0.2: irrelevant).

Persona Reward (R_{text}): A weighted combination of information accuracy ($\mathcal{Q}_{\text{info}}$) and persona alignment ($\mathcal{Q}_{\text{persona}}$):

$$R_{\text{text}} = w_1 \cdot \mathcal{Q}_{\text{info}}(r, \mathcal{C}) + w_2 \cdot \mathcal{Q}_{\text{persona}}(r, \pi_{\text{char}}) \quad (3)$$

where r is the agent response, \mathcal{C} is retrieved context (memory/vision), and π_{char} is the character definition. We set $w_1 = 0.4$ and $w_2 = 0.6$ to prioritize fidelity while maintaining accuracy; both \mathcal{Q} terms are computed by the Judge LLM.

2.4 Execution Loop

To handle runtime failures, we implement a self-correcting retry mechanism with feedback-driven conditional edges:

$$\text{memorizer} \xrightarrow{\text{check_success}} \begin{cases} \text{memory_agent} & \text{if } \neg \text{success} \wedge \text{retry} < 3 \\ \text{orchestrator} & \text{otherwise} \end{cases} \quad (4)$$

In case the Memorizer fails, control routes back to the Memory Agent with an incremented retry counter and feedback, enabling *query refinement*. Algorithm 1 in Appendix B summarizes training and inference.

3 Preliminary Results

In this study, we implement Qwen2.5-1.5B-Instruct (Yang et al., 2024a,b) as the base agent model on which both the SFT and NSARL agent models are trained. We prepared 600 real-world user interactions (500 train / 100 test from non-overlapping scenarios), each represented as a 5-tuple (schema in Appendix I). Ground-truth labels and reference responses were expert-curated with LLM-assisted augmentation. Due to privacy and proprietary constraints, the raw data cannot be publicly released. Table 1 compares the task performance of the three sub-policies (modules) in each agent setting.

Figure 3 shows mean scores with 95% bootstrap CIs. NSARL achieves the best structural performance on Router and Memory (see distributions

Table 1: Performance comparison across three modules (mean score). NSARL improves structural modules (Router, Memory), while SFT yields the best text/persona quality but shows reduced robustness on structured Memory outputs.

Method	Router (Plan)	Memory (Acc.)	Text (Persona)	Avg
Base	0.261	0.259	0.228	0.249
SFT	0.584	0.082	0.368	0.345
NSARL	0.619	0.378	0.251	0.416

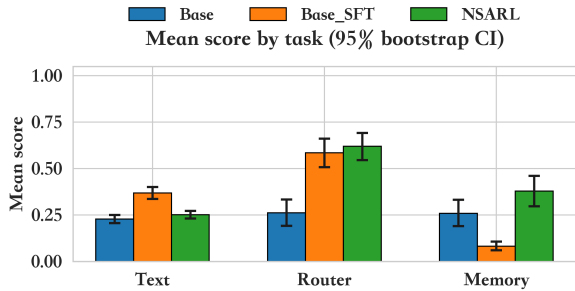


Figure 3: Mean score by task with 95% bootstrap confidence intervals.

in Figure 6 and Figure 5), but does so through conservative policies: a *save-leaning* memory action bias (Figure 5 and Table 3) and *over-routing* (calling extra agents to avoid omissions, Figure 7 and Table 2). In contrast, SFT yields the best persona/text quality but collapses Memory accuracy from 0.259 to 0.082 (68% degradation), revealing a sharp behavioral dichotomy between structural reliability and social fluency. A human evaluation (Appendix J) confirms the same inter-paradigm pattern, with human-Judge alignment ($\rho = 0.52$) comparable to inter-evaluator agreement ($\rho = 0.46$).

Table 2: Router planning inspection under normalized plans.

Model	Jaccard \uparrow	Exact \uparrow	Over \downarrow	Under \downarrow	AvgExtra \downarrow	AvgMissing \downarrow
Base	0.697	0.343	0.333	0.172	0.535	0.333
SFT	0.839	0.573	0.396	0.021	0.406	0.031
NSARL	0.733	0.300	0.700	0.000	0.800	0.000

Table 3: Memory action-level inspection.

Model	ActionAcc \uparrow	SavePrec \uparrow	SaveRec \uparrow	GTDist (s/r/n)	PredDist (s/r/n)
Base	0.380	0.694	0.510	49/9/42	36/42/22
SFT	0.350	0.818	0.551	49/9/42	33/66/1
NSARL	0.490	0.490	1.000	49/9/42	100/0/0

4 Discussion: The Conservatism of RL

The emergence of conservative strategies (Over-route and a save-leaning Memory policy) in our RL

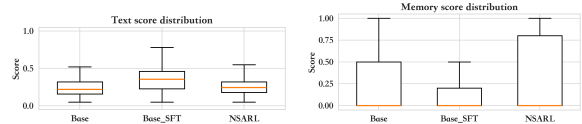


Figure 4: Text score dist. Figure 5: Mem score dist.

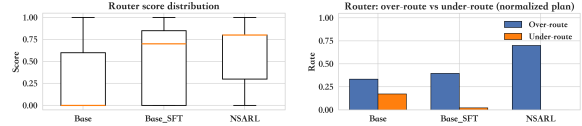


Figure 6: Router score Figure 7: Router rates

agents reveals a “Better Safe Than Sorry” policy that aligns with findings in safety-constrained RL. By prioritizing *completeness* (recall) in the reward signal, the agent learns to “hedge its bets.” While this behavior might seem inefficient (calling Vision Agent on text-only input), in a social companion context, it serves as a robust error-prevention mechanism. This reflects a cost asymmetry: the cost of a missed memory (user feels ignored) is socially higher than the cost of a redundant retrieval (latent computation). RL naturally converges to this asymmetry, whereas SFT, mimicking average human data, may not reliably satisfy the system schema under distribution shifts.

Future work will focus on **Reward Shaping** to penalize redundancy without sacrificing safety, effectively tuning the agent’s risk tolerance. The modular architecture ensures that such calibration can be performed at the sub-policy level with limited cascading effects on other modules.

5 Conclusion

We presented NSARL, a neuro-symbolic framework for OC agents. Our analysis uncovers a dichotomy: RLVR-trained agents act as rigorous “system engineers,” learning conservative strategies to improve structural reliability, whereas SFT agents act as creative “role-players,” maintaining higher persona consistency but exhibiting weaker robustness on structured Memory operations.

This challenges the notion that a single training paradigm can solve all agentic dimensions. We advocate for a **hybrid deployment strategy**: using NSARL modules to govern logic and state transitions, and SFT modules to generate persona-rich content. This combination addresses the generalization gap by enforcing safety boundaries without sacrificing social fluency.

6 Limitations

Our study has three primary limitations. First, the reward functions for Memory and Router currently prioritize high recall, which incentivizes conservative behaviors (e.g., over-routing and a save-leaning bias in Memory action selection) rather than precise discrimination. While this can improve safety, it introduces computational inefficiency and may reduce action precision. Second, our evaluation relies on a Judge LLM and structured evaluation scenarios derived from user interactions; while we use verifiable metrics (RLVR), subjective attributes like “character voice” are difficult to quantify. Third, the “hybrid deployment” strategy, while promising, introduces system complexity that requires further engineering to balance the latency overhead of switching between SFT and RL modules. Future work will address these by incorporating precision penalties into the reward model and conducting longitudinal human user studies.

References

- Hongzhan Chen and 1 others. 2024. Socialbench: Sociality evaluation of role-playing conversational agents. In *Findings of the Association for Computational Linguistics: ACL 2024*.
- Yanqi Dai and 1 others. 2025. Mmrole: A comprehensive framework for developing and evaluating multimodal role-playing agents. In *Proceedings of the Thirteenth International Conference on Learning Representations (ICLR 2025)*.
- DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Shihan Dou and 1 others. 2024. StepCoder: Improving code generation with reinforcement learning from compiler feedback. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*.
- Haodong Duan, Jueqi Wei, Chonghua Wang, Hongwei Liu, Yixiao Fang, Songyang Zhang, Dahua Lin, and Kai Chen. 2023. Botchat: Evaluating llms’ capabilities of having multi-turn dialogues. *Preprint*, arXiv:2310.13650.
- Feiteng Fang and 1 others. 2025. Charm: Character-based act-adaptive reward modeling for advanced role-playing language agents. *arXiv preprint arXiv:2505.23923*.
- B. Jin and 1 others. 2025. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.
- Callie Y. Kim, Christine P. Lee, and Bilge Mutlu. 2024. Understanding large-language model (llm)-powered human-robot interaction. In *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction (HRI '24)*.
- Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. 2025. Agent lightning: Train any ai agents with reinforcement learning. *Preprint*, arXiv:2508.03680.
- K. Mahowald and 1 others. 2024. Dissociating language and thought in large language models. *Trends in Cognitive Sciences*.
- J. S. Park and 1 others. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of UIST*.
- Z. Shao and 1 others. 2025. Deepseekmath-v2: Towards self-verifiable mathematical reasoning. *arXiv preprint arXiv:2511.22570*.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv:2409.19256*.
- Z. Wang and 1 others. 2025. Raiden-r1: Improving role-awareness of llms via grpo with verifiable reward. *arXiv preprint arXiv:2505.10218*.
- X. Wen and 1 others. 2025. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base LLMs. *arXiv preprint arXiv:2506.14245*.
- J. Wu and 1 others. 2025. RLVR-World: Training world models with reinforcement learning. *arXiv preprint arXiv:2505.13934*.
- Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanxia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. 2024. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *Preprint*, arXiv:2408.08152.
- A. Yang and 1 others. 2024a. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.
- An Yang and 1 others. 2024b. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.
- Tao Yang and 1 others. 2025. PsyPlay: Personality-infused role-playing conversational agents. *arXiv preprint arXiv:2502.03821*.
- Huaye Zeng, Dongfu Jiang, Haozhe Wang, Ping Nie, Xiaotong Chen, and Wenhua Chen. 2025. ACECODER: Acing coder rl via automated test-case synthesis. *ArXiv*, 2502.01718.

G. Zhang and 1 others. 2025a. The landscape of agentic reinforcement learning for LLMs: A survey. *arXiv preprint arXiv:2509.02547*.

Weizhi Zhang and 1 others. 2025b. PersonaAgent: When large language model agents meet personalization at test time. *arXiv preprint arXiv:2506.06254*.

Y. Zheng and 1 others. 2025. DeepResearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*.

A Related Work

Agentic Reinforcement Learning. Recent surveys characterize a paradigm shift from “passive sequence generators” to “autonomous decision-making agents” (Zhang et al., 2025a). While Reinforcement Learning from Human Feedback (RLHF) has been pivotal for alignment, its application to *agentic workflows* (i.e., optimizing multi-turn tool-use trajectories rather than merely generating text) remains nascent. Recent frameworks such as veRL (Sheng et al., 2024) and Agent Lightning (Luo et al., 2025) have demonstrated that modular RL training can effectively mitigate the *temporal credit assignment problem* in complex systems, allowing agents to optimize for long-horizon objectives with delayed rewards. **Critically, the majority of Agentic RL research has focused on deep research (Jin et al., 2025; Zheng et al., 2025), code generation (Dou et al., 2024; Zeng et al., 2025) and mathematical reasoning (Xin et al., 2024; Shao et al., 2025),** where ground-truth verification is straightforward. Our work extends this paradigm to the significantly more challenging domain of *open-ended social interaction*, where success criteria are inherently subjective and context-dependent.

Role-Playing and Social Companion Agents.

Early studies on social agents relied heavily on prompt engineering to maintain personas (Park et al., 2023). Subsequent research has focused on improving persona fidelity through diverse approaches: PersonaAgent (Zhang et al., 2025b) introduced test-time personalization modules, PsyPlay (Yang et al., 2025) proposed personality-infused dialogue generation achieving 80.31% trait portrayal success rate on GPT-3.5, and MM-Role (Dai et al., 2025) extended role-play to multimodal settings. Evaluation methodologies have also matured, with benchmarks like SocialBench (Chen et al., 2024) systematically assessing

social competence across individual and group interactions.

Most relevant to our work, recent studies have begun applying RL to role-play agents. RAIDEN-R1 (Wang et al., 2025) combines GRPO with Verifiable Role-Awareness Reward (VRAR) to enhance character consistency, while ChARM (Fang et al., 2025) proposes character-adaptive reward modeling with self-evolution mechanisms. However, these approaches primarily optimize for *output-level* persona fidelity (whether the generated text matches the character), without addressing the *execution-level* decisions that precede generation, such as whether the agent queried the correct memory or routed to the appropriate tool.

Inspired by the success of RLVR in reasoning tasks (DeepSeek-AI, 2025), we propose a **closed-loop** approach that optimizes both *execution correctness* (i.e., did the agent query the right memory?) and *output quality* (i.e., does the response maintain character?). By training on explicit execution trajectories with verifiable feedback, we bridge the cognitive-execution gap in OC companion agents, ensuring that internal reasoning aligns with high-level goals such as long-term consistency and persona fidelity.

B Algorithm: NSARL

C System Prompts

This appendix provides the complete system prompts used for each agent node in our NSARL framework.

C.1 Router

You are the Router Agent for an Original Character (OC) Fantasy system.
Analyze the input and decide the execution plan.

Available Agents:

- vision_agent: Analyze images.
- text_agent: Generate text response.
- memory_agent: Decide on memory operations.
- memorizer: Execute memory operations.
MUST be included immediately after memory_agent if memory operations are needed.

Output JSON with key "plan" containing a list of agent names.

Example: ["vision_agent", "memory_agent", "memorizer", "text_agent"]

Algorithm 1 NSARL Training & Execution

Require: Dataset \mathcal{D} , Modules $\pi_{\text{router}}, \pi_{\text{mem}}, \pi_{\text{text}}$, Judge J

- 1: **Phase 1: Graph Execution**
- 2: **function** ROLLOUT(s_0, π)
- 3: $s_t \leftarrow s_0$
- 4: $O \leftarrow \text{Orchestrator}(\pi_{\text{router}}(s_0))$
- 5: **while** O not finished **do**
- 6: $n \leftarrow O.\text{next}()$; $a \leftarrow \text{ExecuteNode}(n, s_t)$
- 7: **if** a is FAIL **and** n is Memorizer **then**
- 8: $s_t \leftarrow s_t \cup \{\text{feedback}\}$
- 9: $O.\text{retry}(\text{MemoryAgent})$
- 10: **end if**
- 11: **end while**
- 12: **return** Trajectory τ
- 13: **end function**
- 14: **Phase 2: Modular Training**
- 15: **for** each module $m \in \{\text{Router}, \text{Mem}, \text{Text}\}$ **do**
- 16: Freeze other modules $\pi_{\setminus m}$
- 17: **for** batch $B \in \mathcal{D}_m$ **do**
- 18: $\tau_i \leftarrow \text{Rollout}(s_i, \pi)$ with **ForcePlan**(m)
- 19: $\theta_m \leftarrow \text{GRPO_Update}(\tau_i, J(\tau_i))$
- 20: **end for**
- 21: **end for**

Input Format:

User Input: {user_input}
Metadata: {metadata}
Has Image: {has_image}

C.2 Memory Agent

You are the Memory Agent.
Decide: retrieve, save, or none.

Output JSON:

- action: "retrieve" | "save" | "none"
- target: "location" | "memory"
- query: search keywords/ID
- content: data to save
- meta: metadata (including location, timestamp, emotion)

Input Format:

User: {user_input}
Vision: {vision_output}
Meta: {metadata}
Feedback: {memory_feedback}

C.3 Persona Agent

You are a Fantasy OC. Cheerful, loves food.

Input Format:

User: {user_input}
Vision: {vision_output}
Memory: {memory_result}
History: {messages}

C.4 Vision Agent

Analyze this image. Provide JSON with:

- description
- objects
- emotions
- location_type

D Judge LLM Prompts

This appendix provides the complete prompts used by the Judge LLM to compute verifiable rewards (RLVR).

D.1 Router Reward

You are a System Architect. Evaluate the execution plan generated by the Router Agent.

Inputs

User Input: {input_text}
Metadata: {metadata}

Plans

[Model Plan]: {predicted_plan}
[Golden Plan]: {ground_truth_plan}

Scoring Rules (0.0 - 1.0)

- **Base Score**: Start at 0.0.
- **Completeness**: +1.0 if [Model Plan] contains ALL agents present in [Golden Plan].
 - Deduct 0.5 for EACH missing critical agent.
- **Efficiency**: Deduct 0.2 for EACH extra agent that is NOT in [Golden Plan].
- **Order**: Deduct 0.3 if the order is wrong.

(Minimum score is 0.0)

Return ONLY the float score.

D.2 Memory Reward

You are a Data Logic Evaluator. Judge the Memory Agent's decision.

Context

User Input: {input_text}

Decisions

```
[Model Decision]: Action={predicted.action},
                  Content={predicted.content}
[Golden Decision]: Action={gt.action},
                   Content={gt.content}
```

Scoring Rules (0.0 - 1.0)

1. **Action Check (Binary)**:
 - If action != {gt.action}, score = 0.0. STOP.
2. **Content Quality (Scale)**:
 - If Action is correct, evaluate content.
 - Score 1.0 if semantically identical.
 - Score 0.8 if key entities present, different phrasing.
 - Score 0.5 if key entities missing.
 - Score 0.2 if content is irrelevant.

Return ONLY the float score.

D.3 Persona Reward

You are an expert Role-Play Critic. Evaluate the AI's response based on the Character Definition and Context.

```
### 1. Character Definition (The AI's Persona)
{agent_system_prompt}
```

```
### 2. Context
User Input: {user_input}
Vision/Memory Info: {context_info}
```

```
### 3. Responses
[Model Response]: {agent_response}
[Reference Response]: {ground_truth_response}
```

```
### Evaluation Criteria
Score the [Model Response] from 0.0 to 1.0:
```

1. **Information Accuracy (40%)**:
 - Does it address the user's input?
 - Does it correctly use Vision/Memory info?
2. **Persona Alignment (60%)**:
 - Does tone/style match the definition?
 - Compare to [Reference Response]. Does it capture the same "vibe"?
 - If generic AI assistant style, penalize heavily.

Return ONLY the float score (e.g., 0.85).

E Agent State

The complete AgentState TypedDict schema used in the LangGraph workflow:

```
class AgentState(TypedDict):
    # Input Context
    messages: List[BaseMessage]
    user_input: str
    image_path: Optional[str]
    metadata: Dict[str, Any]

    # Workflow Control
    plan: List[str]
    initial_plan: List[str]
    force_plan: Optional[List[str]]
    current_step: str

    # Intermediate Outputs
    vision_output: Optional[Dict[str, Any]]
    mock_vision_output: Optional[Dict[str, Any]]
    memory_instruction: Optional[Dict[str, Any]]
    memory_result: Optional[str]
    memory_success: Optional[bool]

    # Self-Correction Control
    memory_retry_count: int
    memory_feedback: Optional[str]

    # Final Output
    final_response: Optional[str]
```

F Training Configuration

The complete training configuration used with Agent Lightning (GRPO algorithm):

```
RL_TRAINING_CONFIG = {
    "algorithm": {
        "adv_estimator": "grpo",
        "use_kl_in_reward": False,
    },
    "data": {
        "train_batch_size": 32,
        "max_prompt_length": 4096,
        "max_response_length": 2048,
        "truncation": "error",
    },
    "actor_rollout_ref": {
        "rollout": {
            "tensor_model_parallel_size": 1,
            "n": 4,
            "log_prob_micro_batch_size_per_gpu": 4,
            "multi_turn": {"format": "hermes"},
        }
    }
}
```

```

        "name": "vllm",
        "gpu_memory_utilization": 0.8,
    },
    "actor": {
        "ppo_mini_batch_size": 32,
        "ppo_micro_batch_size_per_gpu": 4,
        "optim": {"lr": 1e-6},
        "use_kl_loss": False,
        "clip_ratio_low": 0.2,
        "clip_ratio_high": 0.3,
        "fsdp_config": {
            "param_offload": True,
            "optimizer_offload": True,
        },
    },
    "model": {
        "path": "Qwen/Qwen2.5-1.5B-Instruct",
        "enable_gradient_checkpointing": True,
    },
},
"trainer": {
    "n_gpus_per_node": 1,
    "total_epochs": 2,
    "test_freq": 32,
},
}

```

G Memory Instruction

The JSON schema for memory instructions output by the Memory Agent:

```

{
  "action": "retrieve" | "save" | "none",
  "target": "location" | "memory",
  "query": "search keywords or ID",
  "content": "data to save (for save action)",
  "meta": {
    "location": "location name",
    "timestamp": "ISO timestamp",
    "emotion": "emotional context",
    "location_id": "auto-linked location ID"
  }
}

```

H Location Linking

Pseudocode for the automatic location linking mechanism (Auto-Link Logic) in the Memorizer node:

```

function save_memory_with_autolink(instruction, meta) {
    meta = instruction.meta
    location_name = meta.location

```

```

    if location_name is valid:
        locations = load("location.json")
        loc_id = find_location(locations,
                               location_name)

        if loc_id is None:
            # Create new location entry
            loc_id = len(locations) + 1
            locations[loc_id] = {
                "name": location_name,
                "created_from_memory": True,
                "description": "Auto-generated"
            }
            save("location.json", locations)

            # Link memory to location
            meta.location_id = loc_id

    memories = load("memory.json")
    memories.append({
        "text": instruction.content,
        "meta": meta
    })
    save("memory.json", memories)

```

I Data Specification

Each data sample is a 5-tuple with the following fields:

- **User Input:** The user message (text), optionally accompanied by visual input.
- **Metadata:** Contextual attributes such as location and time.
- **Ground-Truth Router Plan:** The execution plan as an ordered list of agent names, e.g., [vision_agent, memory_agent, memorizer, text_agent].
- **Ground-Truth Memory Operation:** The memory instruction, consisting of an action type (retrieve/save/none) and associated content (see Appendix G for the full schema).
- **Reference Persona Response:** A response reflecting the target character persona.

J Human Evaluation

We conducted a human evaluation on 30 randomly sampled test items (90 responses across Base, SFT, and NSARL). Each response was independently

scored by 2 evaluators using the same rubric as the Judge LLM, yielding 180 total ratings. next time?"

The evaluation confirms the same inter-paradigm pattern as Table 1: SFT ($M = 0.75$) achieves the highest Persona quality, while NSARL ($M = 0.43$) and Base ($M = 0.44$) remain comparable. The item-level evaluator-Judge Spearman correlation ($\rho = 0.52$) is comparable to inter-evaluator agreement ($\rho = 0.46$), suggesting that the Judge LLM is roughly as aligned with evaluators as the evaluators are with each other. [END]

K Interaction Trace

An example execution trace showing the flow through the NSARL system:

[INPUT]

User: "Remember that we had ramen together at Ichiran last week!"

Metadata: {"session_id": "abc123"}

[ROUTER AGENT]

Plan: ["memory_agent", "memorizer", "text_agent"]

[ORCHESTRATOR]

Next Step: memory_agent

[MEMORY AGENT]

Output: {
 "action": "save",
 "target": "memory",
 "content": "Had ramen together at Ichiran",
 "meta": {
 "location": "Ichiran",
 "emotion": "happy",
 "timestamp": "last week"
 }
}

[MEMORIZER]

Auto-Link: Created Location ID 1 for "Ichiran"
Result: "Saved memory (linked to location 1)"
Success: True

[ORCHESTRATOR]

Next Step: text_agent

[TEXT AGENT]

Output: "Of course I remember! That tonkotsu ramen was absolutely divine~ The rich, creamy broth... *chef's kiss* We should definitely go back soon! Maybe try their spicy version"