Ontology-driven Reinforcement Learning for Personalized Student Support

Finley Holt

2025-12-25

Table of contents

1 Ontology-driven Reinforcement Learning for Personalized Student Support

Source: https://arxiv.org/abs/2407.10332 **Authors**: Ryan Hare and Ying Tang (Rowan University) **Year**: 2024 **Venue**: IEEE Systems, Man, and Cybernetics Conference **arXiv ID**: 2407.10332v2 [cs.CY]

1.1 Overview

This paper presents a general-purpose framework for personalized student support in virtual educational systems, addressing the National Academy of Engineering's 2008 grand challenge of advancing personalized learning. The system combines ontologies, data collection, and multi-agent reinforcement learning to create a modular, widely-applicable solution for intelligent tutoring without requiring direct instructor intervention.

The framework is designed to augment (rather than replace) human intelligence in educational settings where resource constraints, limited budgets, and large classroom sizes make one-on-one tutoring infeasible. By integrating with virtual educational systems such as serious games and intelligent tutoring systems, the approach provides students with timely, personalized feedback regardless of location (classroom, study group, or home).

1.2 Decision Impact for Flyby-F11

1.2.1 ADOPT - High Confidence

1. Ontology-Structured Mission Architecture - **Evidence**: Paper demonstrates directed acyclic graph (DAG) ontology successfully organizes educational concepts hierarchically; same structure directly applicable to mission planning (objectives → phases → behaviors → primitives) - **Implementation**: Use ontology G = C, E, , D, where C = mission concepts (waypoints, zones, targets), E = semantic relationships (precedes, requires, conflicts-with), = properties (altitude constraints, speed limits, risk level), D = behavior database, = RL agents - **Rationale**: Flyby-F11's communications-denied operations require interpretable mission structure; ontologies provide human-readable semantic organization for mission intent verification

2. Multi-Agent RL with Specialized Task Assignment - **Evidence**: Paper shows multiple topic-specific RL agents outperform monolithic approaches; each agent specializes in narrow domain while sharing experience - **Implementation**: Assign RL agents to mission-critical behaviors (obstacle avoidance, target tracking, path following, emergency landing); use PPO or SAC for continuous control - **Rationale**: Jetson Orin NX (50 TOPS) has sufficient compute for multiple lightweight RL agents running concurrently; specialization improves performance in safety-critical scenarios

3. Experience Sharing Between Agents - **Evidence**: Prior work [16, 17] demonstrated experience sharing eliminates poor initial performance, provides "jump-start" for new agents - **Implementation**: Use shared replay buffer across related agents (e.g., different altitude-hold agents share experiences); transfer learned policies when adding new mission types - **Rationale**: Critical for flyby-f11 deployment—cannot afford extended learning periods on real hardware; must leverage simulation-trained agents with rapid real-world adaptation

4. Data Transformation Layer for Sensor Fusion - **Evidence**: Paper's transformation functions $f : y \rightarrow x$ successfully standardize diverse input formats into consistent state representation - **Implementation**: Transform heterogeneous sensor data (camera, lidar, IMU, GPS, depth) into standardized state vectors for RL agents; enables platform-agnostic agent design - **Rationale**: Flyby-f11's multi-sensor suite requires robust normalization; transformation layer isolates agents from sensor-specific formatting, enabling reuse across platforms

1.2.2 CONSIDER - Needs Validation

1. Deep Policy Gradient Methods (DDPG, TD3, SAC) - **Promise**: Continuous action spaces suitable for UAV control (thrust vectoring, velocity commands, gimbal angles) - **Concern**: Sample efficiency—these methods require extensive training data which may be impractical for real-world UAV deployment - **Validation Needed**: Benchmark in PX4 SITL with realistic wind/turbulence; measure training time vs. model-based controllers; test sim-to-real transfer gap - **Alternative**: Consider hybrid approach with model-based controller as baseline + RL for adaptive tuning

2. Online Learning During Flight Operations - **Promise**: Paper emphasizes online adaptation to shifting trends; could enable in-flight learning for environmental adaptation - **Concern**: Safety-critical systems require certified performance bounds; online learning introduces unpredictability - **Validation Needed**: Prototype with constrained parameter updates (safe RL methods); test in non-critical mission phases only - **Alternative**: Use offline RL with periodic model updates between flights rather than continuous online learning

3. LLM-Based Assistance Generation for Mission Interpretation - **Promise**: Paper mentions LLMs for dynamic content generation; could translate natural language mission intent to ontology - **Concern**: LLMs require significant compute (may strain Jetson Orin NX); latency concerns for real-time operation - **Validation Needed**: Benchmark quantized LLMs (Llama 3.2 3B, Phi-3-mini) on Orin NX; measure inference latency and power consumption - **Alternative**: Pre-process mission commands on ground station; transmit structured ontology representation rather than raw text

4. Reward Function Design for Multi-Objective Missions - **Promise**: Paper uses simple performance-based rewards; UAVs require balancing safety, efficiency, mission success - **Concern**: Reward engineering is notoriously difficult; poor design leads to unexpected behaviors - **Validation Needed**: Systematic testing of reward formulations in simulation; validate against hand-coded policies - **Alternative**: Use inverse RL to learn reward from expert demonstrations; apply reward shaping with domain knowledge

1.2.3 AVOID - Evidence Against

1. Fully Autonomous Ontology Construction - **Why Avoid**: Paper assumes pre-designed ontology; automated ontology learning is an open research problem with unreliable results - **Evidence**: No demonstration of automatic knowledge graph construction; educational ontologies are

human-designed - **Flyby Impact**: Mission ontologies must be manually designed by domain experts; attempting automated construction will delay deployment - **Alternative**: Design mission ontology collaboratively with MCTSSA operators; iterate based on field feedback

2. Discrete Action Spaces for UAV Control - **Why Avoid**: Paper emphasizes continuous action vectors; discretization loses fidelity in control - **Evidence**: Educational assistance uses continuous weights [0, 1] for nuanced personalization - **Flyby Impact**: UAV control requires smooth, continuous commands (velocity setpoints, attitude targets); discretization causes chattering/instability - **Alternative**: Use policy gradient methods (PPO, SAC) designed for continuous control

3. Unlimited State/Action Dimensionality - **Why Avoid**: Paper doesn't address computational constraints; assumes sufficient compute for arbitrary state spaces - **Evidence**: Deep RL scales poorly beyond ~100-dimensional states without careful architecture design - **Flyby Impact**: Jetson Orin NX has limited power budget (10-25W); high-dimensional state spaces increase inference latency - **Alternative**: Apply dimensionality reduction (PCA, autoencoders) to sensor data; use hierarchical state representation

1.2.4 INVESTIGATE - Open Questions

1. Optimal Agent Granularity for Mission Hierarchy - **Question**: Paper states agents should be "granular enough for specialized assistance while broad enough to avoid requiring massive numbers of agents"—what is optimal for UAV missions? - **Approach**: Prototype with different agent assignments (one per flight phase vs. one per primitive behavior) and compare performance, training time, computational overhead - **Success Criteria**: Balance between specialization benefit and system complexity; target 10 concurrent agents on Orin NX

2. Ontology Update Mechanisms for Dynamic Missions - **Question**: How to modify mission ontology in-flight when objectives change without disrupting RL agent performance? - **Approach**: Test adding/removing ontology nodes during runtime; measure agent adaptation speed; validate safety constraints maintained - **Success Criteria**: Agents adapt to new mission constraints within acceptable time window; no unsafe behaviors during transition

3. Experience Sharing Across Dissimilar Platforms - **Question**: Can RL agents trained on project-drone (Orin Nano Super, T265+D455) transfer effectively to flyby-f11 (Orin NX, different sensors)? - **Approach**: Train agents in project-drone simulation; deploy to flyby-f11 simulation with different dynamics; measure performance degradation and adaptation time - **Success Criteria**: <20% performance degradation; adaptation within 100 episodes or transferable directly with minimal fine-tuning

4. Real-Time Inference Latency on Jetson Orin NX - **Question**: Can multi-agent RL system maintain control loop rates (50-100 Hz for PX4 offboard control)? - **Approach**: Profile RL agent inference time on Orin NX with realistic sensor processing pipeline; test different network architectures (MLP, CNN, lightweight transformers) - **Success Criteria**: Total inference latency <10ms (100 Hz) for all agents combined; <5ms for critical safety agents (collision avoidance)

5. Sim-to-Real Transfer for Wind/Turbulence Adaptation - **Question**: Paper demonstrates educational system with deterministic dynamics; UAVs face stochastic wind disturbances—will learned policies transfer? - **Approach**: Train agents in PX4 SITL with domain randomization (varying wind conditions); test on real flyby-f11 hardware in controlled environment - **Success**

Criteria: Policies trained purely in simulation achieve 80% of performance of policies fine-tuned on real hardware

6. Formal Verification of Safety Constraints - **Question**: How to guarantee RL agents respect hard safety constraints (altitude limits, no-fly zones, collision avoidance)? - **Approach**: Investigate safe RL methods (constrained MDPs, safety layers, shield synthesis); integrate with PX4's geofencing and failsafe mechanisms - **Success Criteria**: Mathematically provable safety guarantees under specified operating conditions; zero constraint violations in 1000+ simulation runs

1.3 System Architecture

The proposed system consists of four interconnected components:

1.3.1 1. Educational Concept Ontology

The ontology serves as the core structural representation of domain knowledge, formally modeled as a directed acyclic graph:

G = C, E, , D, where:

- **C = {ci}**: Set of concepts (nodes) representing specific knowledge or topics within the domain
- **E = {ci, cj, pk}**: Set of directional edges connecting topics with semantic relationships defined by property pk
- **(ci) = { j}**: Set of attributes/properties associated with each concept (difficulty level, relevance to educational objectives, etc.)
- **D**: Database of educational materials where each concept ci has a subset Dci containing related materials (text tutorials, videos, example problems, interactive simulations)
- : Set of reinforcement learning agents assigned to select concepts in C

Design Considerations: - Agent-augmented nodes must be granular enough for specialized assistance while broad enough to avoid requiring massive numbers of agents - Ontology structure informs lesson plan organization and content presentation order - Semantic connections create meaningful hierarchical structure similar to traditional lesson planning

Example: In a mathematical functions ontology, RL agents might be assigned to "linear function", "quadratic function", "exponential function", and properties like "domain" and "range", with sub-topics used to inform tutoring recommendations.

1.3.2 2. Data Collection, Transformation, and Storage

Student Data Vector (xci): Each concept ci stores a general-purpose performance metrics vector applicable to any topic.

Example metrics: - a: Competency measure (quiz score) - b: Timing measure (time-to-completion) - c: Engagement indicator (number of interactions, idle time)

Data Transformation: The system maps diverse input data formats (y) to standardized system format (x) using semantic descriptors and transformation functions f : y → x.

Example transformation equations (from paper): - Competency: x = y (quiz score maps directly) - Engagement: x = (y /y + y /y)/2 (combines interaction count and time-on-content ratio) - Emotional state: x adjusts based on whether time spent exceeds average, potentially indicating frustration

This transformation layer allows different educational systems with varied data collection methods to interface with the same RL architecture.

1.3.3 3. Multi-agent Reinforcement Learning

The "brain" of the system uses deep policy gradient methods suitable for continuous action spaces. Each RL agent is assigned to a specific ontology topic.

Formal MDP Definition for each agent ci:

MDP = S, A, R, P, where:

- **S**: Finite state space; state observation st = xci at time t
- **A**: Finite action space; action a is a vector where each element represents a property of personalized assistance
- **R(s, a, s')**: Reward function encouraging desired behaviors (e.g., positive reward for student performance improvements)
- **P(s'|s, a)**: Transition probability dynamics (challenging to predict in human-centric environments)
- **[0, 1]**: Discount factor weighting future rewards

RL Algorithm Selection: - Uses deep policy gradient methods for continuous action outputs - Suitable algorithms: DDPG (Deep Deterministic Policy Gradient), TD3 (Twin Delayed DDPG), SAC (Soft Actor-Critic), PPO (Proximal Policy Optimization) - Prior work demonstrated multi-agent experience sharing to improve training performance and avoid poor initial performance periods

Key Advantages: - Online iterative learning eliminates need for complex hand-coded decision logic - Adapts to shifting trends in student preferences and lesson plan changes - Guarantees eventual optimal performance through convergence properties - Multiple specialized agents provide topic-specific expertise

1.3.4 4. Personalized Assistance Generation

This module translates the RL agent's action vector (a*) and database content (Dci) into actual student assistance.

Implementation approaches: - **Rule-based systems**: Assign threshold values to content; show materials based on action vector weights - **Large language models**: Dynamically regenerate content with varied guidance/encouragement levels - **Hybrid approaches**: Combine rule-based content filtering with LLM-based presentation adaptation - **Interface variations**: NPC dialogue in games, pop-up prompts in tutoring systems, adaptive content delivery

Example action vector (a* = {a , a , a , a , a }): - a : Weight for visual content inclusion [0, 1] - a : Weight for examples (videos, step-by-step guides) [0, 1] - a : Weight for practice problems [0, 1] - a : Weight for guidance level (low = materials only, high = detailed guidance) [0, 1] - a : Weight for encouraging/emotional dialogue [0, 1]

The module flexibility allows integration with emerging technologies like generative AI for dynamic content creation.

1.4 Methodology

1.4.1 Experience Sharing Between Agents

Building on prior work [16, 17], the system implements experience sharing where: - Newly-added agents leverage shared experience from existing agents - Provides "jump-start" at beginning of training - Avoids initial period of poor performance - Enables modular extensibility without performance degradation

1.4.2 Ontology-Driven Organization

The ontology serves three critical functions: 1. **Knowledge representation**: Hierarchical structure mirrors educational lesson plans 2. **Agent assignment**: Determines which concepts receive specialized RL agents 3. **Content organization**: Semantic relationships inform assistance generation

1.4.3 Data Flow

1. Virtual educational system collects student performance data
2. Transformation module converts diverse metrics to standard format (x)
3. RL agent observes state (xci) and selects action (a*)
4. Assistance generation module creates personalized content
5. Student receives assistance; performance feeds back as reward signal
6. RL agent updates policy based on reward

1.5 Key Findings

1.5.1 Theoretical Contributions

1. **Modularity**: System can be adapted to any virtual educational software without redesigning core components
2. **Generalizability**: Data transformation layer enables diverse data sources to interface with standardized RL architecture
3. **Scalability**: Topics can be easily added/adjusted; ontology structure naturally accommodates expansion
4. **Adaptability**: Online RL ensures system responds to shifting student behavior trends

1.5.2 Prior Validation

The authors reference their prior work demonstrating: - Positive impact of experience sharing on agent training performance [16] - Successful application in educational serious game showing positive impact on student performance [17] - Effective multi-agent RL approach for personalized support in digital logic education

1.5.3 Advantages Over Traditional Approaches

- **No instructor intervention required**: AI provides timely feedback without waiting for teacher availability
- **Active prompting**: Helps students who fail to formulate proper questions
- **Handles outliers**: Addresses students left behind by one-size-fits-all methods
- **Prevents dropout**: Reduces student frustration from falling behind

- **Resource efficiency**: Allows well-performing students to address difficulties while poor-performing students catch up

1.6 Technical Implementation Details

1.6.1 State Space Dimensionality

- Determined by dimensionality and variable range of xci
- Deep RL methods handle large, non-discrete state spaces
- Continuous state observations represent student performance metrics

1.6.2 Action Space Design

- Continuous vector output (not discrete action selection)
- Each element controls different aspect of personalized assistance
- Policy gradient methods naturally output continuous values

1.6.3 Reward Function Design

- Application-dependent but should encourage beneficial behaviors
- Natural configuration: positive reward for student performance improvements
- Can incorporate multiple objectives (competency, engagement, emotional state)

1.6.4 Database Integration

- Each ontology concept links to educational materials subset
- Materials can include: text, videos, problems, simulations
- Properties enable filtering/selection based on action vector

1.7 Relevance to Drone Autonomy

1.7.1 Direct Parallels

1. **Hierarchical Mission Structure**:
   - Educational ontology → Mission ontology (flight phases, objectives, constraints)
   - Student topics → UAV behaviors (takeoff, navigation, obstacle avoidance, landing)
   - Lesson progression → Mission phase sequencing
2. **Multi-Agent Specialization**:
   - Topic-specific RL agents → Behavior-specific control agents
   - Experience sharing between educational agents → Knowledge transfer between mission scenarios
   - Ontology-guided agent assignment → Mission hierarchy determining control delegation
3. **Adaptive Decision-Making**:
   - Student performance metrics → Flight performance/environmental state
   - Personalized assistance → Adaptive control policies
   - Online learning for student trends → Online adaptation to environmental conditions
4. **Modular Architecture**:
   - Adding educational topics → Adding mission capabilities
   - Data transformation layer → Sensor fusion/data normalization
   - Assistance generation → Control command generation

1.7.2 Specific Applications to UAV Systems

Mission Planning: - Ontology represents mission objectives, constraints, and available behaviors - Multi-agent RL handles specialized tasks (path planning, obstacle avoidance, target tracking) - Experience sharing enables rapid adaptation when new mission types are introduced

Behavioral Hierarchies: - High-level ontology nodes: mission objectives (reconnaissance, delivery, inspection) - Mid-level nodes: flight phases (transit, loiter, engage) - Low-level nodes: primitive behaviors (altitude hold, waypoint navigation, collision avoidance) - RL agents assigned at appropriate granularity for specialization

Adaptive Control: - State vector represents flight state, environmental conditions, mission progress - Action vector controls behavior parameters (aggressiveness, path deviation tolerance, sensor modes) - Reward function balances mission success, safety, efficiency

Communications-Denied Operations (relevant to flyby-f11): - Ontology encodes mission intent and autonomy levels - RL agents make decisions without operator input (similar to students learning without instructor) - System adapts to unexpected situations using learned policies

Data Handling: - Sensor data transformation analogous to student data transformation - Diverse sensor types (camera, lidar, IMU, GPS) $\rightarrow$ standardized state representation - Enables consistent RL architecture across different UAV platforms

1.7.3 Key Insights for UAV Implementation

1. **Semantic Organization**: Ontologies provide interpretable structure for autonomous decision-making, critical for safety-critical UAV systems
2. **Modular Extensibility**: Adding new behaviors/capabilities without retraining entire system
3. **Specialization vs. Generalization**: Balance between specialized agents (better performance) and fewer agents (simpler system)
4. **Experience Transfer**: Accelerate learning for new missions using knowledge from related missions
5. **Online Adaptation**: Continuously improve performance as system accumulates flight experience

1.7.4 Challenges for UAV Adaptation

- **Safety constraints**: Educational errors are recoverable; UAV errors can be catastrophic
- **Real-time requirements**: Student assistance timing flexible; UAV control requires strict latency bounds
- **Physical dynamics**: Student learning is cognitive; UAV learning must account for physics
- **Verification**: Harder to validate RL policies for safety-critical flight operations
- **Reward function design**: Clear educational outcomes; UAV mission success harder to quantify

1.8 Related Work

The paper builds on established research in: - **Ontologies in education**: Student modeling [10, 11], lesson planning [12], content representation [13] - **RL in education**: Intelligent tutoring systems [6, 7], adaptive content recommendation [24] - **Deep RL methods**: DDPG [18], TD3 [19], SAC [20],

PPO [21] - **AI in education**: Augmented intelligence [1], serious games [8], LLMs [22], generative AI [23]

1.9 Future Directions

The authors identify several areas for future work:

1. **Empirical Validation**: Testing in actual educational contexts with real students
2. **Assistance Generation Optimization**: Deeper exploration of content generation methods
3. **RL Improvements**: Additional optimizations to multi-agent learning
4. **LLM Integration**: Leveraging large language models for dynamic content creation
5. **Broader Applications**: Extending framework to additional educational domains

1.10 Implementation Readiness Assessment

Component	Jetson Orin NX Compatibility	ROS 2 Integration Status	Development Effort	Risk Level	Key Dependencies
Ontology Graph Structure	Excellent - Lightweight graph operations; minimal compute overhead	Native - Use `rclcpp` parameter system or YAML configs for ontology definition	2-3 weeks - Design mission ontology structure, define semantic relationships, implement graph traversal	Low - Well-established graph libraries (NetworkX, Boost Graph); deterministic behavior	`networkx` (Python) or `Boost.Graph` (C++); ROS 2 parameter server
Multi-Agent RL Framework	Moderate - Orin NX (50 TOPS) supports 5-10 lightweight agents; requires optimized inference	Partial - ROS 2 RL frameworks exist (`ros2learn`, custom action interfaces) but require integration	8-12 weeks - Implement agent architecture, design state/action spaces, integrate with PX4	Medium - RL training stability; sim-to-real gap; computational constraints	PyTorch/TensorFlow Lite; ONNX Runtime; PX4 MAVSDK; Gazebo simulation

Component	Jetson Orin NX Compatibility	ROS 2 Integration Status	Development Effort	Risk Level	Key Dependencies
PPO/SAC Policy Gradient	Good - Continuous control suitable for UAVs; TensorRT optimization available for inference	Partial - Training frameworks mature (`stable-baselines3`, `RLlib`) but deployment requires custom ROS 2 nodes	6-8 weeks - Train policies in simulation, optimize for edge inference, validate performance	Medium - Hyperparameter tuning; reward engineering; requires extensive simulation testing	`stable-baselines3`; TensorRT; CUDA 11.8+; PX4 SITL
Experience Sharing Mechanism	Excellent - Shared replay buffer is memory-efficient; synchronization overhead minimal	Native - Use ROS 2 bag files for experience storage; shared memory for cross-agent communication	3-4 weeks - Implement replay buffer, design experience sampling strategy, test transfer learning	Low - Standard RL technique; well-documented in literature [16, 17]	Shared memory (`boost::interprocess`); ROS 2 bag recorder; HDF5 for experience storage
Data Transformation Layer	Excellent - Preprocessing is computationally cheap; can run on CPU alongside perception pipeline	Native - ROS 2 message filters and `message_filters::Synchronizer` for sensor fusion	4-5 weeks - Design transform functions, implement normalization, validate across sensor types	Low - Deterministic math operations; easily testable	ROS 2 `message_filters`; `sensor_msgs`; `tf2` for coordinate transforms

Component	Jetson Orin NX Compatibility	ROS 2 Integration Status	Development Effort	Risk Level	Key Dependencies
Deep Policy Networks (MLP)	Good - 3-5 layer MLPs run efficiently on Orin NX; TensorRT INT8 quantization achieves <5ms inference	Partial - Need custom inference node; ONNX Runtime integrates well with ROS 2	4-6 weeks - Design network architecture, train in simulation, optimize with TensorRT, deploy	Medium - Quantization accuracy loss; need to validate control stability with quantized models	TensorRT 8.6+; ONNX Runtime 1.16+; CUDA/cuDNN; ROS 2 action servers
Ontology-Driven Agent Assignment	Excellent - Graph lookup is O(log n); negligible runtime overhead	Native - ROS 2 lifecycle nodes can dynamically activate/deactivate agents based on mission phase	2-3 weeks - Implement agent manager, define assignment rules, test runtime switching	Low - Straightforward logic; well-defined by ontology structure	ROS 2 lifecycle nodes; `rclcpp_lifecycle`; mission state machine
Continuous Action Space Output	Excellent - Native for policy gradients; maps directly to PX4 offboard velocity/position setpoints	Native - Use `geometry_msgs/TwistStamped` or `px4_msgs/VehicleCommand` for control outputs	2-3 weeks - Define action vector format, implement PX4 command translation, validate control loop	Low - Standard UAV control interface; well-supported by MAVSDK	MAVSDK; `px4_msgs`; `geometry_msgs`; PX4 offboard mode
Reward Function Computation	Excellent - Lightweight scalar computation; can run at control loop rate (100 Hz)	Native - Subscribe to mission status topics, compute reward in custom node	3-4 weeks - Design multi-objective reward, implement shaping functions, tune weights	Medium - Poor reward design leads to unsafe behaviors; requires extensive testing	ROS 2 topic subscriptions; mission state monitoring; safety constraint validators

Component	Jetson Orin NX Compatibility	ROS 2 Integration Status	Development Effort	Risk Level	Key Dependencies
Database of Behaviors (D)	Excellent - Behavior library stored as ROS 2 launch files or behavior tree XMLs; minimal storage/lookup cost	Native - Use `BehaviorTree.CPP` XML library or ROS 2 parameter YAML files	3-4 weeks - Define behavior primitives, implement selection logic, validate compositions	Low - Static database; deterministic retrieval	`BehaviorTree.CPP`; YAML parsing; filesystem I/O
Ontology Properties ( )	Excellent - Store as key-value pairs in ontology nodes; fast access	Native - Use ROS 2 parameters with namespacing (e.g., `/mission/waypoint_altitude_limit`)	1-2 weeks - Define property schema, implement getters/setters, validate constraints	Low - Simple data structure; no complex logic	ROS 2 parameter server; YAML schema validation
Safe RL Constraints	Moderate - Constraint checking adds computational overhead; may need GPU acceleration for complex constraints	Minimal - Limited ROS 2 integration for safe RL; requires custom implementation	10-14 weeks - Research safe RL methods, implement safety layers, integrate with PX4 failsafes, validate	High - Unproven in UAV domain; verification challenges; false positives may prevent mission completion	`safety-gym` (adaptation required); PX4 geofencing; custom constraint solvers; formal verification tools

Component	Jetson Orin NX Compatibility	ROS 2 Integration Status	Development Effort	Risk Level	Key Dependencies
LLM Mission Interpretation	Moderate - Quantized 3B models feasible but strain memory (16GB Orin NX); inference ~500ms-2s	Minimal - No standard ROS 2 LLM integration; requires custom pipeline	8-10 weeks - Select/quantize model, implement inference, design prompt engineering, validate accuracy	Medium - Latency concerns; hallucination risk for safety-critical commands; requires extensive validation	`llama.cpp`; Hugging Face Transformers; GGUF quantization; prompt templates; ground station preprocessing option
Online Learning Update	Moderate - Policy updates require GPU; may disrupt real-time control if not carefully scheduled	Minimal - No standard pattern for online RL in ROS 2 UAVs	12-16 weeks - Implement safe update mechanism, design trigger conditions, validate stability, test edge cases	High - Safety risk from unpredictable policy changes; certification challenges; requires extensive flight testing	Policy gradient optimizers; experience buffer; asynchronous update scheduler; flight mode validators

1.10.1 Summary Statistics

- **High Readiness (4 weeks, Low Risk)**: 7/14 components (50%)
  - Ontology graph, data transformation, experience sharing, agent assignment, action spaces, behavior database, ontology properties
- **Moderate Readiness (4-8 weeks, Medium Risk)**: 4/14 components (29%)
  - Deep policy networks, reward functions, PPO/SAC implementation, LLM interpretation
- **Low Readiness (8 weeks, High Risk)**: 3/14 components (21%)
  - Multi-agent RL framework integration, safe RL constraints, online learning

1.10.2 Recommended Development Sequence

Phase 1 (Weeks 1-6): Foundation 1. Design mission ontology structure 2. Implement data transformation layer 3. Set up behavior database 4. Create ontology property system 5. Build

agent assignment manager

Phase 2 (Weeks 7-14): Core RL System 1. Train initial PPO/SAC policies in PX4 SITL 2. Optimize networks with TensorRT 3. Implement experience sharing 4. Design reward functions 5. Validate continuous action space control

Phase 3 (Weeks 15-22): Advanced Features 1. Integrate multi-agent coordination 2. Implement safe RL constraints (if validation successful) 3. Prototype LLM mission interpretation (optional) 4. Conduct sim-to-real transfer experiments

Phase 4 (Weeks 23-30): Validation & Deployment 1. Hardware-in-the-loop testing 2. Real-world flight validation 3. Performance benchmarking 4. Safety certification preparation

1.10.3 Critical Path Items

- **Blocker**: Multi-agent RL framework integration (8-12 weeks) - Must complete before advanced features
- **Risk**: Safe RL constraints (10-14 weeks) - High complexity; consider deferring to Phase 2 iteration
- **Dependency**: PX4 SITL environment setup - Required for all RL training; must prioritize early

1.10.4 Resource Requirements

- **Compute**: NVIDIA Jetson Orin NX 16GB (target platform); desktop GPU (RTX 3080+) for training
- **Software**: ROS 2 Humble, PX4 v1.14+, Gazebo Garden, PyTorch 2.0+, TensorRT 8.6+
- **Data**: ~100 hours simulated flight time for policy training; ~20 hours real flight data for validation
- **Personnel**: 1-2 robotics engineers with RL expertise; estimated 6-9 months for full system integration

1.11 Conclusions

This work demonstrates a theoretically sound approach for integrating ontologies with multi-agent reinforcement learning in a domain (education) with clear parallels to autonomous systems. The framework's emphasis on: - Semantic knowledge representation - Modular agent specialization - Data transformation for system flexibility - Adaptive online learning

…provides valuable insights for UAV autonomy architectures, particularly for mission-intent interpretation and hierarchical behavioral control in communications-denied environments.

The paper's contribution is primarily architectural—showing how ontologies can structure multi-agent RL systems in a modular, extensible way. While the educational application differs from UAV control, the underlying principles of hierarchical knowledge organization, specialized agent assignment, and adaptive decision-making translate directly to autonomous drone missions.

1.12 References

[1] K.-L. A. Yau et al., "Augmented intelligence: Surveys of literature and expert opinion," IEEE Access, vol. 9, pp. 136744-136761, 2021.

[6] F. AlShaikh and N. Hewahi, "AI and machine learning techniques in the development of intelligent tutoring system," 2021 3ICT, pp. 403-410, IEEE, 2021.

[10] H. Yago et al., "ONSMMILE: Ontology network-based student model for multiple learning environments," Data & Knowledge Engineering, vol. 115, pp. 48-67, 2018.

[14] C. J. C. H. Watkins, "Learning from Delayed Rewards," PhD thesis, King's College, Oxford, 1989.

[16] R. Hare and Y. Tang, "Reinforcement learning with experience sharing for intelligent educational systems," 2023 IEEE SMC.

[17] R. Hare, Y. Tang, and S. Ferguson, "An intelligent serious game for digital logic education," IEEE Trans. on Education, 2024.

[18] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2019.

[19] S. Fujimoto et al., "Addressing function approximation error in actor-critic methods," 2018.

[20] T. Haarnoja et al., "Soft actor-critic: Off-policy maximum entropy deep RL," 2018.

[21] J. Schulman et al., "Proximal policy optimization algorithms," 2017.