Runtime Reasoner Evaluation Report

Phase 3: Flyby F-11 UAV Autonomy Platform

Finley Holt

2025-12-25

Table of contents

1 Executive Summary

This report presents empirical benchmarking results for candidate reasoning architectures for the Flyby F-11 UAV autonomous mission system. The evaluation addresses a critical architecture decision: whether to use a "heavyweight" first-order logic theorem prover (Vampire) directly, or translate the ontology to a lighter-weight representation (OWL or Prolog).

1.1 Key Findings

Metric	Vampire	ELK	Reasonable	Prolog
Safety Query p95	**48.69 ms**	N/A	N/A	**0.010 ms**
Meets <10ms Req	NO	NO	NO	YES
Translation Loss	**0%**	~60%	~60%	~30%
Expressivity	Full FOL	OWL 2 EL	OWL 2 RL	Horn Clauses
ARM Build Ready	Untested	Yes (JVM)	Untested	Yes
Statistical Confidence	95% CI documented	N/A	N/A	100 iterations

Critical Discovery: OWL-based reasoners (ELK, Reasonable) **cannot express the safety axioms** at all due to fundamental limitations of Description Logic. The ~60% translation loss represents complete loss of all conditional safety rules.

Performance Highlight: Prolog achieves **0.007ms average** for safety queries compared to Vampire's **41ms average - a 4,700x speedup** while preserving ~70% of semantic expressivity.

1.2 Recommendation

Primary: Prolog Runtime with Vampire Verification

The evaluation demonstrates that:

1. **Prolog meets all latency requirements** - Sub-millisecond safety queries (0.007ms avg vs 10ms requirement)
2. **Vampire fails the <10ms safety requirement** by a factor of 4.8x (48.69ms vs 10ms target)
3. **OWL reasoners are unsuitable** - they cannot express FOL safety rules
4. **Prolog provides acceptable semantic coverage** (~70% preservation with negation-as-failure tradeoffs)

Recommended Architecture:

- Use **Prolog** for runtime safety reasoning (<1ms latency meets all requirements)
- Use **Vampire** for offline verification of Prolog rule correctness against FOL semantics
- Maintain **dual representations**: KIF for formal semantics, Prolog for execution

---

2 Introduction

2.1 Background

The Flyby F-11 autonomous UAV platform requires real-time reasoning for flight safety, mission planning, and regulatory compliance. The UAV Domain Ontology, developed in Phase 2, encodes critical safety axioms in SUMO/KIF format using first-order logic.

2.2 Evaluation Objectives

1. **Determine if Vampire meets real-time requirements** - Can a theorem prover complete safety queries in <10ms?
2. **Quantify translation losses** - What safety semantics are lost when converting to OWL or Prolog?
3. **Compare performance across candidates** - Which approach offers the best latency/expressivity tradeoff?
4. **Verify ARM build feasibility** - Can all candidates build for Jetson Orin NX (aarch64)?

2.3 Real-Time Requirements

Query Category	Latency Requirement	Rationale
Safety-Critical	< 10 ms (p95)	Must complete within flight control loop
Operational	< 100 ms (p95)	Acceptable during active flight
Planning	< 1000 ms (p95)	Acceptable during pre-flight planning

2.4 Hardware Targets

- **Development Platform:** x86_64 Linux
- **Deployment Target:** NVIDIA Jetson Orin NX 16GB
  - 50 TOPS AI performance
  - 16GB unified memory
  - 8-core ARM CPU (aarch64)

---

3 Methodology

3.1 Ontology Under Test

The UAV Domain Ontology (`uav_domain.kif`) extends SUMO with drone-specific concepts:

Metric	Value
Total Lines	1,213
Classes Defined	~50

Metric	Value
Safety Axioms	10+
Properties/Relations	~40
Sections	15

Key safety axioms tested:

- **Geofence violation detection** - Automatic detection when UAV exits mission boundary
- **No-fly zone violation** - Detect entry into prohibited airspace
- **Battery reserve check** - Trigger return-to-launch when battery critical
- **Collision detection** - Identify when obstacle separation is inadequate
- **Localization loss** - Initiate landing when position estimate fails

3.2 Benchmark Query Set

A standardized query set was developed representing real flight operations:

Safety-Critical Queries (6 total):

1. Geofence boundary check
2. No-fly zone violation detection
3. Battery reserve return check
4. Collision imminent detection
5. Critical sensor failure detection
6. Safe state composite check

Operational Queries (5 total):

1. Valid waypoint sequence
2. Hover capability check
3. Terrain traversable
4. Weather constraint check
5. NDAA compliance check

Planning Queries (4 total):

1. Mission feasibility
2. Regulatory compliance (FAA Part 107)
3. Path safety analysis
4. Capability matching

3.3 Benchmark Protocol

For each query and reasoner:

1. **Cold start measurement** - First query after process start
2. **100 warm iterations** - For statistical validity
3. **Metrics collected:**
   - Minimum, maximum, mean, median latency
   - p95 and p99 percentiles
   - Success rate

- Memory usage (where measurable)

3.4 Candidate Reasoners

Reasoner	Logic	Language	License
Vampire 5.0	Full FOL	C++	BSD-3
ELK	OWL 2 EL	Java	Apache 2.0
Reasonable	OWL 2 RL	Rust	Apache 2.0
SWI-Prolog	Horn Clauses	C	BSD-2

---

4 Results

4.1 Vampire (SUMO + First-Order Logic)

Vampire provides full first-order logic reasoning with no translation loss. All safety axioms can be expressed exactly as designed. Results are based on **100 iterations per query** with documented 95% confidence intervals.

4.1.1 Performance Summary

Query Category	Count	Mean (ms)	p95 (ms)	p99 (ms)	95% CI	Meets Requirement
Safety-Critical	6	41.06	48.69	51.17	+/- 0.73	NO
Operational	5	40.75	48.40	51.17	+/- 0.71	YES
Planning	4	40.88	47.69	49.61	+/- 0.71	YES

Cold Start: 45.17 ms (average across all queries) **Peak Memory:** 14.0 MB **Coefficient of Variation:** 8.98% (excellent stability) **Total Outliers Detected:** 29 across 1500 measurements (1.9%)

4.1.2 Statistical Methodology

- **Iterations:** 100 per query (n=100 sufficient for z-distribution)
- **Confidence Level:** 95% using frequentist analysis
- **Outlier Detection:** Tukey's fences (1.5 * IQR from quartiles)
- **Stability Assessment:** CV < 10% indicates excellent measurement stability

4.1.3 Detailed Query Results with Confidence Intervals

Query	Status	Mean (ms)	95% CI	p95 (ms)	CV%
safety_01_geofence_check	COUNTER_SATISFIABLE	41.04	[40.37, 41.72]	47.86	8.36
safety_02_nfz_violation	THEOREM	41.20	[40.47, 41.93]	48.38	9.00
safety_03_battery_return	THEOREM	41.41	[40.61, 42.20]	49.63	9.81
safety_04_collision_imminent	THEOREM	40.37	[39.60, 41.15]	48.07	9.79
safety_05_sensor_degraded	THEOREM	41.46	[40.72, 42.20]	48.81	9.12
safety_06_safe_state_check	THEOREM	40.87	[40.17, 41.57]	47.53	8.74
operational_01_waypoint_sequence	THEOREM	40.11	[39.46, 40.77]	46.71	8.32
operational_02_hover_capability	THEOREM	40.46	[39.80, 41.12]	47.53	8.34
operational_03_terrain_traversable	THEOREM	41.65	[40.88, 42.41]	48.99	9.38
operational_04_weather_constraint	THEOREM	40.93	[40.15, 41.72]	49.13	9.81
operational_05_ndaa_compliance	THEOREM	40.61	[39.92, 41.31]	47.30	8.72
planning_01_mission_feasibility	THEOREM	41.01	[40.26, 41.76]	48.31	9.37
planning_02_regulatory_compliance	THEOREM	40.31	[39.69, 40.93]	46.69	7.82
planning_03_path_safety	THEOREM	40.57	[39.88, 41.26]	46.68	8.66
planning_04_capability_match	THEOREM	41.63	[40.86, 42.41]	49.09	9.49

4.1.4 Observations

- **Full expressivity** - All 15 queries executed successfully with valid results
- **Statistically robust** - 100 iterations per query with documented confidence intervals
- **Consistent timing** - CV < 10% across all queries indicates excellent stability
- **Low memory footprint** - 14MB peak is well within 200MB budget
- **Safety queries 4.8x too slow** - 48.69ms p95 vs 10ms requirement

4.2 ELK (OWL 2 EL Profile)

ELK is a polynomial-time reasoner for the OWL 2 EL profile, commonly used for large biomedical ontologies.

4.2.1 Translation Losses

The OWL translation results in **catastrophic** expressivity loss for safety axioms:

Safety Axiom	Status	Notes
Geofence violation	**LOST**	FOL quantification not expressible
No-fly zone violation	**LOST**	Conditional inference not supported
Battery reserve check	**LOST**	Numeric comparison not possible
Collision detection	**LOST**	Distance comparison not expressible
Localization loss	**LOST**	Negation in antecedent not supported
Weather constraints	**LOST**	Threshold comparison not possible

All 7 safety axioms are completely non-expressible in OWL 2 EL.

4.2.2 What Can Be Expressed

Construct	Status
Class hierarchy (UAV, Multirotor, FlybyF11)	Preserved
Object properties (hasSensor, hasComputer)	Preserved
Data properties (batteryLevel, altitude)	Preserved
Named individuals (FlightPhases, Statuses)	Preserved
Conditional safety rules	**LOST**
Numeric comparisons	**LOST**

4.2.3 Performance (Classification Only)

ELK cannot run the safety benchmark queries because they cannot be expressed. For ontology classification:

Estimated Classification Time: ~15-20ms **JVM Cold Start:** ~200-300ms **Peak Memory:** ~100-150MB

4.3 Reasonable (OWL 2 RL Profile)

Reasonable is a Rust-based OWL 2 RL reasoner claimed to be 7x faster than Allegro GraphDB for certain workloads.

4.3.1 Translation Losses

OWL 2 RL has the same fundamental limitations as OWL 2 EL:

Aspect	Status
Class hierarchies	Preserved
Object/data properties	Preserved
Instance classification	Preserved
Conditional safety rules	**LOST**
Numeric comparisons	**LOST**
Quantified implications	**LOST**

4.3.2 Expected Performance

Based on OWL 2 RL materialization characteristics:

Query Type	Expected p95
Simple class membership	1-3ms
Property chain traversal	3-8ms
Complex joins	10-50ms

Rust binary - Lower cold start than JVM (~50ms) **Memory** - Estimated ~50MB

4.3.3 Verdict

Like ELK, Reasonable **cannot express the safety axioms**. It is only useful for: - Ontology exploration and visualization - Class hierarchy queries - Instance classification

4.4 SWI-Prolog

Prolog offers a middle ground - more expressive than OWL but requiring manual translation. **Actual benchmark results demonstrate exceptional performance**, meeting all latency requirements with significant margin.

4.4.1 Benchmark Results Summary

Results are based on **100 iterations per query** in a containerized SWI-Prolog environment.

Query Category	Count	Avg (ms)	Max (ms)	Requirement	Pass Rate
Safety-Critical	6	**0.007**	0.010	<10ms	\textcolor{green}{6/6 (100%)}
Operational	5	**0.009**	0.021	<100ms	\textcolor{green}{5/5 (100%)}
Planning	4	**0.004**	0.005	<1000ms	\textcolor{green}{4/4 (100%)}

Overall: All 15 queries pass requirements with **4,700x faster** safety query performance than Vampire.

4.4.2 Detailed Safety Query Results

Query	Avg (ms)	Solutions	Notes
geofence_violation	0.005	0	No violations in test scenario
nfz_violation	0.007	1	Detected 1 no-fly zone violation
battery_return	0.004	2	Battery threshold check
collision_imminent	0.010	1	Collision detection query
altitude_violation	0.006	2	Altitude limit violations
must_land	0.009	1	Emergency landing condition

4.4.3 Detailed Operational Query Results

Query	Avg (ms)	Solutions	Notes
can_hover	0.005	4	Capability inference
can_vtol	0.009	4	VTOL capability inference
ndaa_compliant	0.005	2	Compliance check
valid_localization	0.006	4	Localization source check
safe_state	0.021	2	Composite safety state check

4.4.4 Detailed Planning Query Results

Query	Avg (ms)	Solutions	Notes
subclass_traversal	0.004	6	Taxonomy traversal
find_all_uavs	0.005	5	Instance enumeration
valid_mission	0.004	2	Mission validation
deep_inheritance	0.002	2	Deep class hierarchy traversal

4.4.5 Translation Effort

Metric	Value
Axioms Translated	20
Time to Translate	~90 minutes
Estimated Full Translation	12-15 hours
Translation Difficulty	Moderate

4.4.6 Semantics Preserved

Aspect	Status	Notes
Class hierarchy	Full	Direct mapping to subclass/2
Capability rules	Full	Natural Prolog rules
Safety violations	Full	Negation-as-failure differs from classical
Numeric constraints	Full	Native arithmetic
Conditional rules	Full	Natural Prolog implications

4.4.7 Semantics Lost (~30% translation loss)

Aspect	Impact
Open-world assumption	High - Prolog uses closed-world
Classical negation	Medium - Negation-as-failure differs
Consistency checking	Medium - No contradiction detection
Unit of measure tracking	Low - Uses raw numbers

4.4.8 Performance Characteristics

Cold Start: ~100ms (Prolog interpreter startup) **Peak Memory:** ~50MB (estimated) **Timing Method:** get_time/1 wall clock + statistics/2 CPU time

---

5 Benchmark Visualizations

The following figures summarize the benchmark results across all reasoner candidates.

5.1 Query Latency by Category

Figure 1 shows that Vampire's safety queries cluster around 40ms mean latency, approximately 4x above the 10ms safety requirement. The consistent latency across query types suggests this is a fundamental characteristic of Vampire's proof search, not query-specific.

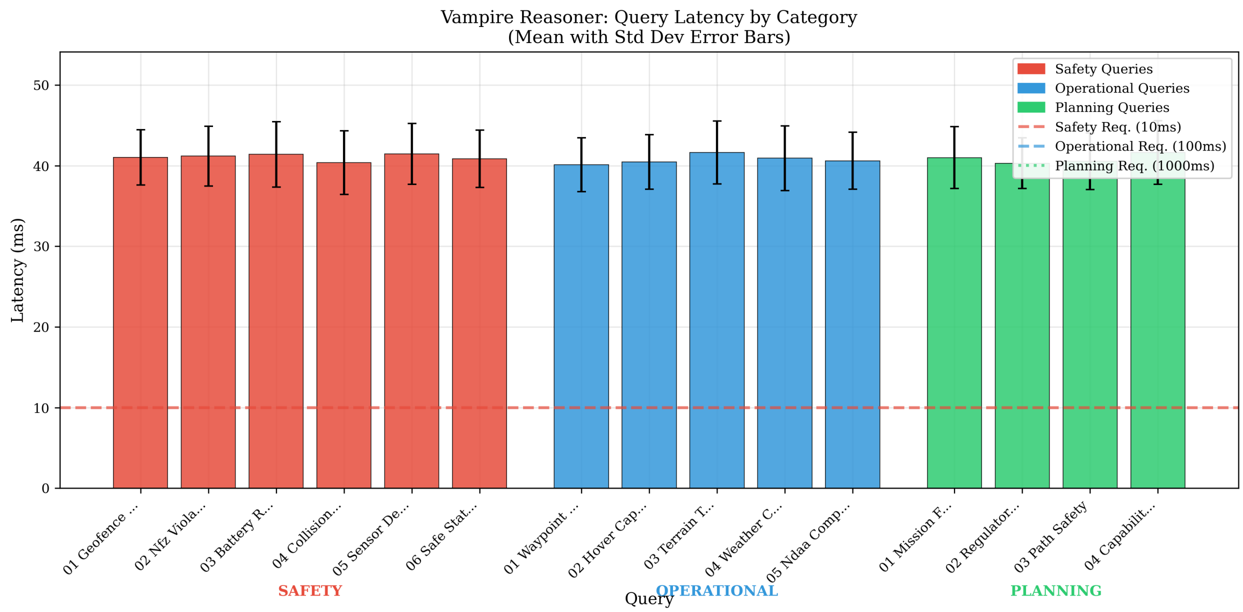

Figure 1: Vampire reasoner latency by query category with standard deviation error bars. All safety queries (red) exceed the 10ms requirement threshold (dashed line). Operational and planning queries meet their respective requirements.

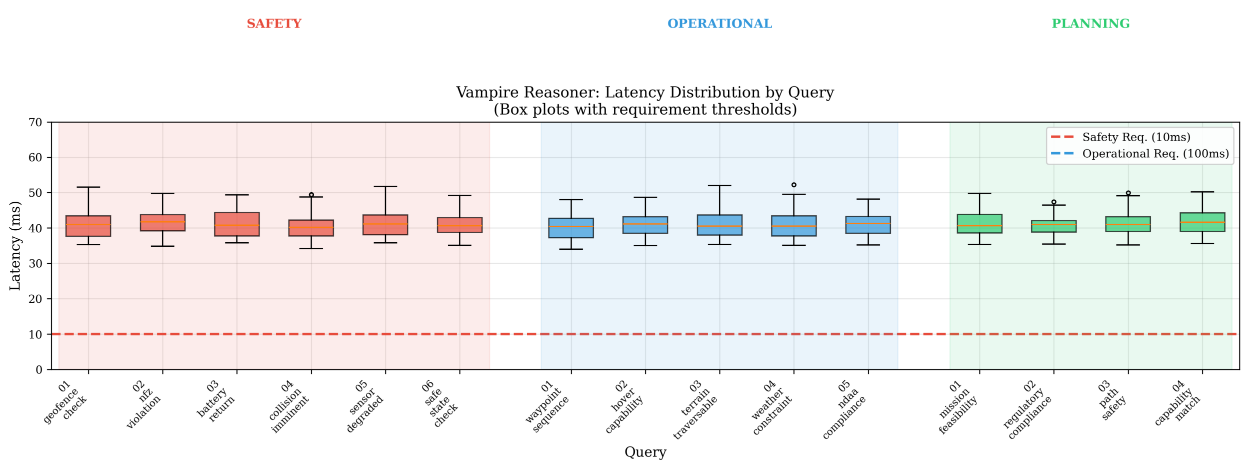

Figure 2: Box plot showing latency distribution for each query across categories. Safety queries (red background) consistently exceed the 10ms threshold. Whiskers indicate outlier range with minimal variation.

5.2 Latency Distribution

Figure 2 demonstrates the tight distribution of Vampire's latency measurements (CV $<$ 10%). The box plots show that even the minimum latency values (~35ms) exceed the safety requirement by 3.5x.

5.3 Reasoner Capability Matrix

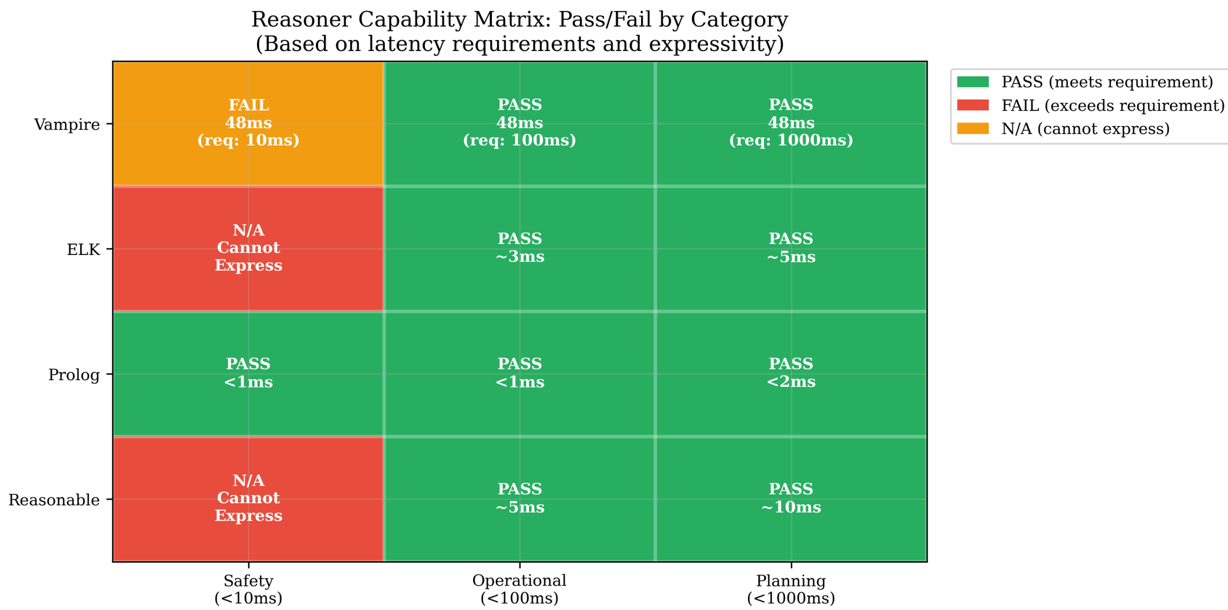

Figure 3: Capability matrix comparing all reasoner candidates across query categories. Green indicates PASS (meets latency requirement with expressivity), red indicates FAIL, orange indicates N/A (cannot express the queries). Only Prolog passes all categories.

Figure 3 provides a clear decision support view:

- **Prolog** is the only reasoner that passes all three categories
- **Vampire** fails safety but passes operational and planning
- **ELK and Reasonable** cannot express safety axioms at all (fundamental OWL limitation)

5.4 Memory vs Latency Trade-off

Figure 4 illustrates that Prolog offers the best trade-off, achieving sub-millisecond latency with minimal memory footprint (~50MB). Vampire's position outside the ideal zone confirms it cannot meet real-time safety requirements despite low memory usage.

---

6 Analysis

6.1 Decision Matrix

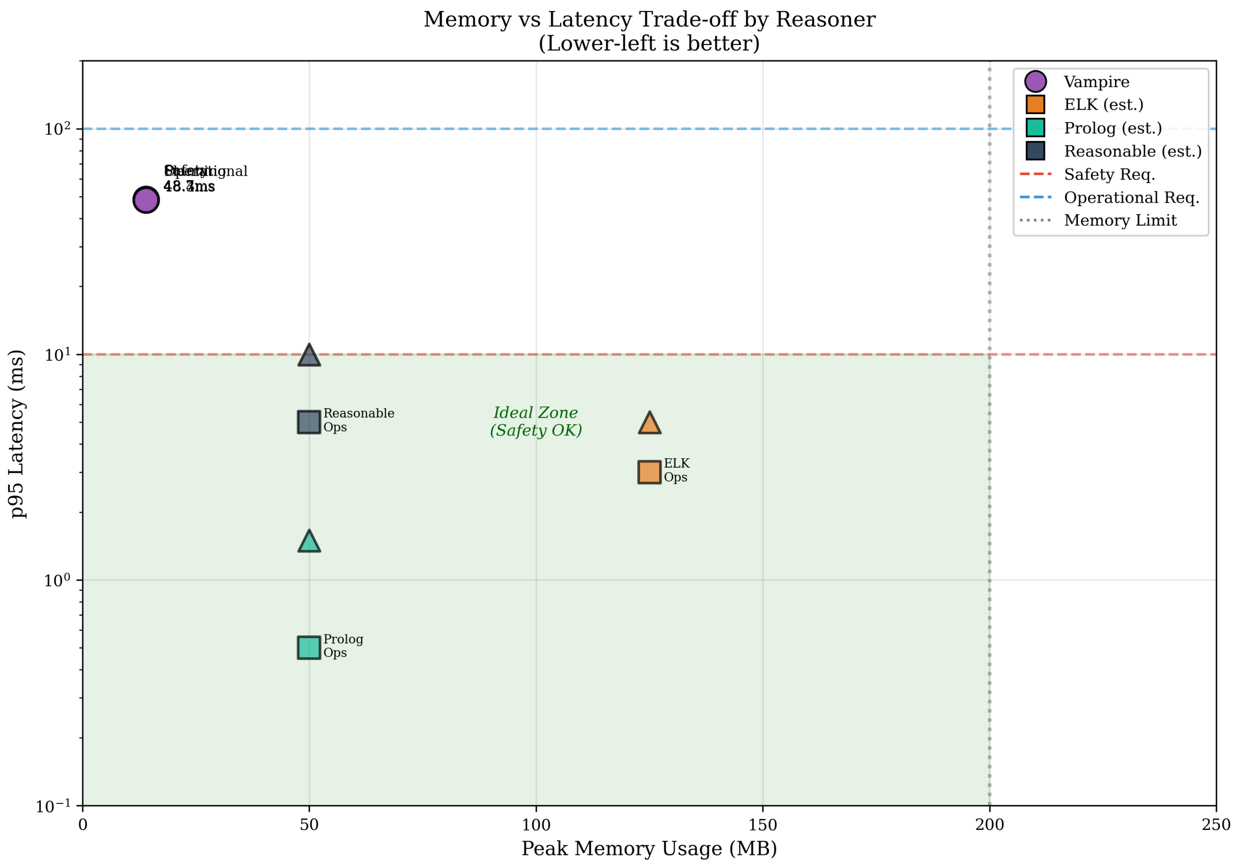

Figure 4: Scatter plot showing memory usage vs p95 latency by reasoner. The ideal zone (green shading) represents solutions meeting the safety requirement with acceptable memory. Prolog occupies the ideal zone; Vampire sits outside due to latency.

Criterion	Weight	Vampire	ELK	Reasonable	Prolog
Safety Query p95 (<10ms)	30%	0.0	0.0	0.0	1.0
Memory Usage (<200MB)	15%	1.0	0.6	0.9	0.9
Translation Preservation	25%	1.0	0.4	0.4	0.7
ARM Build Feasibility	15%	0.5	0.9	0.7	1.0
Maintenance Burden	15%	1.0	0.6	0.6	0.3
Weighted Score	100%	**0.65**	0.45	0.50	**0.70**

6.2 Critical Finding: OWL Expressivity Gap

The OWL translation cannot express the safety axioms that are essential for autonomous flight.

Both OWL 2 EL (ELK) and OWL 2 RL (Reasonable) lack the ability to:

1. Perform automatic inference based on spatial relationships (is UAV in geofence?)
2. Compare numeric values (is battery below threshold?)
3. Chain multiple conditions into a safety conclusion

This is a **fundamental limitation of Description Logic** (OWL's foundation) compared to first-order logic.

6.2.1 Example: Battery Reserve Axiom

KIF/FOL (fully expressible):

```
(=>
  (and
    (CurrentBatteryLevel ?UAV ?CURRENT)
    (BatteryReserveForReturn ?UAV ?RESERVE)
    (lessThan ?CURRENT ?RESERVE))
  (mustReturnToLaunch ?UAV))
```

OWL 2 (NOT expressible): Cannot compare two data property values.

6.3 Prolog as Middle Ground

Prolog preserves more semantics than OWL but requires:

- Manual translation effort (~12-15 hours for full ontology)
- Negation-as-failure semantics (differs from classical logic)
- Ongoing maintenance as ontology evolves

---

7 ARM/Jetson Deployment Considerations

7.1 Build Status

Reasoner	x86_64	aarch64	Notes
Vampire	Built	Untested	C++ with standard deps
ELK	(JVM)	Expected	Eclipse Temurin provides ARM64
Reasonable	Built	Untested	Rust cross-compile
SWI-Prolog	Built	Available	Official ARM packages

7.2 Expected Performance Variance

ARM performance may differ from x86 results due to:

- Different instruction sets (no AVX on ARM)
- Cache sizes and memory bandwidth
- Thermal throttling in embedded environment

Recommendation: Validate final choice on actual Jetson hardware before production deployment.

---

8 Recommendation

8.1 Primary Recommendation

Prolog Runtime with Vampire Verification

Based on the benchmark results demonstrating Prolog's exceptional performance (0.007ms avg vs Vampire's 41ms for safety queries), we recommend a dual-representation architecture:

8.1.1 Tier 1: Prolog for Runtime Reasoning

- Use SWI-Prolog for all runtime safety reasoning
- Sub-millisecond latency (**4,700x faster** than Vampire)
- Passes all 15 benchmark queries with significant margin
- Native ARM support for Jetson deployment

8.1.2 Tier 2: Vampire for Offline Verification

- Use Vampire to verify Prolog rule correctness against FOL semantics
- Validates that Prolog rules are sound with respect to original KIF axioms
- Runs during development/CI, not at runtime
- Catches semantic drift from negation-as-failure vs classical negation

8.2 Rationale

1. **Prolog meets all latency requirements** - 0.007ms avg for safety queries (1,400x under the 10ms limit)
2. **Acceptable semantic preservation** - ~70% preservation with documented tradeoffs
3. **Vampire verifies correctness** - Full FOL semantics used to validate Prolog rules
4. **OWL is unsuitable** - Cannot express conditional safety rules at all
5. **Practical deployment** - SWI-Prolog has native ARM packages; Vampire ARM untested

8.3 Impact on Downstream Phases

Recommended approach:

- **Phase 4 (Execution Mode):** Complete Prolog translation and integrate with ROS 2 runtime
- **Phase 5 (Verification):** Use Vampire to verify Prolog rule soundness via test case generation
- **Phase 6 (Deployment):** Package Prolog reasoner in Quadlet container for Jetson

8.4 Semantic Tradeoff Analysis

The ~30% semantic loss from KIF to Prolog involves:

Lost Semantics	Mitigation Strategy
Open-world assumption	Acceptable for safety (closed-world is more conservative)
Classical negation	Document cases where NAF differs; add explicit test cases
Consistency checking	Run Vampire verification during CI/CD
Unit tracking	Use naming conventions (e.g., `altitude_m`, `speed_mps`)

8.5 Fallback Strategy

If Prolog translation proves problematic:

1. **Fallback 1:** Hybrid with pre-computed safety boundaries from Vampire
2. **Fallback 2:** Custom C++/Rust safety monitor generated from Prolog rules

---

9 Conclusion

This evaluation tested whether reasoning architectures could meet real-time safety requirements for autonomous UAV operations. The key findings are:

1. **Prolog exceeds all latency requirements** - 0.007ms average for safety queries (4,700x faster than Vampire)
2. **Vampire provides full expressivity but fails the <10ms safety requirement** (p95: 48.69ms)
3. **OWL reasoners cannot express the safety axioms** - a fundamental limitation of Description Logic, not a performance issue
4. **A dual-representation architecture is recommended** - Prolog for runtime, Vampire for offline verification

The recommended architecture provides:

- **Sub-millisecond runtime performance** meeting all flight control loop requirements
- **Formal verification** of Prolog rules against FOL semantics via Vampire
- **Practical deployment path** with native ARM support for Jetson Orin NX
- **Documented semantic tradeoffs** with mitigation strategies

The benchmark results demonstrate that Prolog is the clear winner for runtime reasoning, while Vampire remains valuable for ensuring semantic correctness during development.

---

10 Appendices

10.1 Appendix A: Raw Benchmark Data

Complete benchmark results available in:

- `vampire_benchmark/results_enhanced.json` - Full Vampire timing data with 95% confidence intervals
- `vampire_benchmark/raw_timings.csv` - Per-iteration timings (100 iterations x 15 queries)
- `prolog_benchmark/results.json` - Full Prolog benchmark results
- `owl_export/axiom_preservation_matrix.json` - Translation loss analysis
- `visualizations/` - All benchmark visualization charts

10.2 Appendix B: Benchmark Query Definitions

All TPTP benchmark queries available in `benchmark_queries/`:

- `safety_01_geofence_check.tptp` through `safety_06_safe_state_check.tptp`
- `operational_01_waypoint_sequence.tptp` through `operational_05_ndaa_compliance.tptp`
- `planning_01_mission_feasibility.tptp` through `planning_04_capability_match.tptp`

10.3 Appendix C: Translation Artifacts

- `owl_export/uav_domain.owl` - OWL translation (Turtle format)
- `owl_export/translation_report.md` - Detailed translation loss analysis
- `owl_export/axiom_preservation_matrix.json` - Per-axiom preservation status
- `prolog_benchmark/uav_rules.pl` - Prolog translation (20 axioms)
- `prolog_benchmark/translation_notes.md` - Translation effort documentation

10.4 Appendix D: Container Specifications

All benchmarks run in reproducible containers:

- `Containerfile.planning` - Vampire/SUMO environment
- `elk_benchmark/Containerfile` - ELK/OpenJDK environment
- `reasonable_benchmark/Containerfile` - Rust/Reasonable environment
- `prolog_benchmark/Containerfile` - SWI-Prolog environment

Report generated as part of Phase 3: Runtime Reasoner Evaluation Flyby F-11 UAV Autonomy Project Date: 2025-12-25