Implementation Roadmap: 24-Month Deployment Plan

Version: 1.0

Last Updated: October 8, 2025

Document Type: project-tracking

Tractatus AI Safety Framework

https://agenticgovernance.digital

Appendix C: Implementation Roadmap

Document Type: Technical Planning **Date**: October 2025 **Purpose**: Phased deployment plan for Tractatus-based LLM architecture

---

Overview

This roadmap presents a pragmatic 24-month path from proof-of-concept to production deployment of the Tractatus framework across Claude products, starting with Claude Code as the pilot platform.

Key Principles:

- **Start small**: Single product pilot (Claude Code)
- **Validate early**: Measurable metrics within 3 months
- **Iterate quickly**: Monthly validation cycles
- **Scale gradually**: Expand only after proven success
- **Maintain safety**: No compromise on core guarantees

Phase 1: Proof-of-Concept (Months 1-3)

Objective

Demonstrate that time-persistence classification and cross-reference validation can reduce explicit instruction violations by 80% in Claude Code.

Deliverables

1.1 Instruction Persistence Classifier (Month 1)

Goal: Classify user instructions by quadrant and persistence level

Implementation:

```
class InstructionClassifier:
    """
    Classifies user instructions into time-persistence quadrants
    with associated verification requirements.
    """

    def classify(self, instruction: str, context: ConversationContext):
        # Analyze instruction content
        features = self.extract_features(instruction)

        # Determine quadrant
        quadrant = self.classify_quadrant(features, context)

        # Determine persistence level
        persistence = self.classify_persistence(features, quadrant)

        # Determine verification requirement
        verification = self.determine_verification(quadrant, persistence)

        return InstructionMetadata(
            quadrant=quadrant,
            persistence=persistence,
            verification=verification,
            timestamp=context.timestamp,
            original_text=instruction
        )
```

Validation:

- Accuracy: >80% correct quadrant classification
- Latency: <20ms per classification
- Test set: 1000 labeled instructions from real Claude Code sessions

Success Criteria:

- Classifier integrated into Claude Code message processing pipeline
- Classification metadata stored with each user instruction
- Classification visible in debug mode
- Accuracy target achieved on test set

1.2 Simple Cross-Reference Validator (Month 2)

Goal: Detect conflicts between proposed actions and recent explicit instructions

Implementation:

```
class SimpleCrossReferenceValidator:
    """
    Validates proposed actions against recent explicit instructions.
    Focus on parameter conflicts (like the 27027 port failure).
    """

    def validate(self, action: ProposedAction, context: ConversationContext):
        # Extract parameters from proposed action
        action_params = self.extract_parameters(action)

        # Get recent high-persistence instructions
        recent_instructions = self.get_recent_instructions(
            context=context,
            lookback_messages=50,
            min_persistence="MEDIUM"
        )

        # Check each parameter against instructions
        conflicts = []
        for param_name, param_value in action_params.items():
            conflict = self.check_parameter_conflict(
                param_name=param_name,
                param_value=param_value,
                instructions=recent_instructions
            )
            if conflict:
                conflicts.append(conflict)

        if conflicts:
            return ValidationResult(
                status="BLOCKED",
                conflicts=conflicts,
                suggested_action="REQUEST_CLARIFICATION"
            )

        return ValidationResult(status="APPROVED")
```

Validation:

- 27027 failure scenario: 100% detection

- False positive rate: <15%
- Latency: <30ms per validation

Success Criteria:

- Validator integrated before action execution
- Blocks 27027-style failures in test scenarios
- Provides helpful clarification requests to user
- False positive rate acceptable

1.3 Context Pressure Monitor (Month 2-3)

Goal: Detect when context pressure increases error probability

Implementation:

```
class ContextPressureMonitor:
    """
    Monitors conversation context for conditions that increase
    error probability (token pressure, instruction drift).
    """

    def assess_pressure(self, context: ConversationContext):
        pressure_factors = {
            'token_usage': self.assess_token_usage(context),
            'conversation_length': self.assess_length(context),
            'topic_complexity': self.assess_complexity(context),
            'instruction_density': self.assess_instruction_density(context),
            'time_since_summary': self.assess_summary_recency(context)
        }

        pressure_score = self.calculate_pressure_score(pressure_factors)

        if pressure_score > self.WARNING_THRESHOLD:
            return PressureAssessment(
                level="HIGH",
                score=pressure_score,
                factors=pressure_factors,
                recommendation="INCREASE_VERIFICATION"
            )

        return PressureAssessment(level="NORMAL")
```

Validation:

- Correlation with error rate: >0.7
- False alarm rate: <20%
- Intervention effectiveness: >60% error reduction when triggered

Success Criteria:

- Monitor integrated into conversation management
- High pressure increases verification intensity
- User notified when pressure triggers intervention
- Measurable error reduction in high-pressure scenarios

1.4 Integration and Testing (Month 3)

Activities:

- Integrate all three components into Claude Code
- A/B test with subset of users (10% traffic)
- Collect metrics and user feedback
- Iterate based on findings

Metrics Collected:

- Explicit Instruction Violation Rate (EIVR)
- False Positive Rate (FPR)
- User Satisfaction (survey)
- Latency Impact
- Intervention Frequency

Phase 1 Success Criteria

Quantitative:

- EIVR reduction: >80% (from baseline ~15-25% to <5%)
- False Positive Rate: <15%
- Latency increase: <50ms per action
- User satisfaction: No decrease (maintain baseline)

Qualitative:

- User feedback indicates helpful clarifications
- No significant workflow disruption
- Engineering team comfortable with integration
- Clear path to Phase 2 improvements identified

Go/No-Go Decision: If quantitative targets met, proceed to Phase 2. If not, analyze gaps and iterate for additional 1-2 months.

Phase 2: Production Integration (Months 4-6)

Objective

Full Claude Code deployment with enhanced components, achieving 90% EIVR reduction and 85% user satisfaction improvement.

Deliverables

2.1 Enhanced Instruction Classifier (Month 4)

Improvements over Phase 1:

- ML-based classification (trained on real usage data)
- Multi-label classification (instructions can span multiple quadrants)
- Confidence scores for classifications
- Active learning from user corrections

New Capabilities:

```
class EnhancedInstructionClassifier:
    def classify_with_confidence(self, instruction: str, context: ConversationCo
        # ML-based classification
        predictions = self.model.predict(instruction, context)

        # Multi-label support
        quadrants = [q for q, conf in predictions.items() if conf > 0.3]

        # Determine primary quadrant
        primary = max(predictions.items(), key=lambda x: x[1])

        return EnhancedClassification(
            primary_quadrant=primary[0],
            confidence=primary[1],
            secondary_quadrants=quadrants,
            persistence=self.infer_persistence(primary, context),
            verification=self.determine_verification(primary, context),
            explanation=self.generate_explanation(predictions)
        )
```

Success Criteria:

- Accuracy: >90% (up from 80%)
- Confidence calibration: >0.85 correlation
- User correction integration working
- Latency: <25ms (improved from 20ms budget)

2.2 Full Cross-Reference Validation Engine (Month 4-5)

Enhancements:

- Semantic matching (not just keyword/parameter matching)
- Context-aware conflict detection
- Graduated response (clarify vs. block vs. warn)
- Learning from user confirmations

Architecture:

```
class FullCrossReferenceValidator:
    def validate_comprehensive(self, action: ProposedAction, context: Conversati
        # Multi-level validation
        validations = {
            'parameter_conflicts': self.check_parameter_conflicts(action, contex
            'semantic_conflicts': self.check_semantic_conflicts(action, context)
            'boundary_violations': self.check_boundary_violations(action, contex
            'strategic_alignment': self.check_strategic_alignment(action, contex
        }

        # Determine appropriate response
        if validations['boundary_violations']:
            return ValidationResult(status="BLOCKED", reason="boundary_violation

        if validations['parameter_conflicts']:
            return ValidationResult(status="CLARIFY", conflicts=validations['par

        if validations['semantic_conflicts']:
            return ValidationResult(status="WARN", concerns=validations['semanti

        return ValidationResult(status="APPROVED")
```

Success Criteria:

- Detection rate: >95% of conflicts caught
- False positive rate: <10% (down from 15%)
- Graduated responses reduce user interruption
- Latency: <40ms

2.3 Metacognitive Verification Layer (Month 5)

New Component: Adds explicit "pause and verify" before critical actions

Implementation:

```
class MetacognitiveVerifier:
    """
    Adds explicit verification step before executing high-stakes actions.
    Implements the "think before you act" pattern.
    """

    def verify_before_execution(self, action: ProposedAction, context: Conversat
        # Determine if action requires metacognitive check
        if not self.requires_verification(action, context):
            return VerificationResult(required=False)

        # Generate verification questions
        questions = self.generate_verification_questions(action, context)

        # Check each question
        verification_results = []
        for question in questions:
            result = self.check_question(question, action, context)
            verification_results.append(result)

        # Aggregate results
        if any(r.status == "FAIL" for r in verification_results):
            return VerificationResult(
                required=True,
                passed=False,
                failed_checks=verification_results,
                recommendation="BLOCK_AND_CLARIFY"
            )

        return VerificationResult(required=True, passed=True)

    def generate_verification_questions(self, action, context):
        return [
            "Does this action align with recent explicit instructions?",
            "Have I verified all critical parameters?",
            "Is this action within appropriate boundaries?",
            "Would this surprise the user based on their stated intent?",
            "Am I making assumptions that should be confirmed?"
        ]
```

Success Criteria:

- Catches edge cases missed by other components
- Reduces "surprising" actions by >70%
- User feedback indicates improved reliability
- Latency acceptable (<100ms for full verification)

2.4 Human Judgment Boundary Enforcer (Month 5-6)

New Component: Implements Tractatus Section 12 boundaries

Implementation:

```
class BoundaryEnforcer:
    """
    Enforces architectural boundaries for decisions requiring human judgment.
    Based on Tractatus Section 12.
    """

    HUMAN_REQUIRED_DOMAINS = {
        'values': ['ethical decisions', 'priority trade-offs', 'moral judgments'
        'purpose': ['goal definition', 'mission changes', 'strategy pivots'],
        'agency': ['user autonomy', 'consent decisions', 'authority delegation']
    }

    def check_boundaries(self, action: ProposedAction, context: ConversationCont
        # Classify action domain
        domain = self.classify_action_domain(action)

        # Check if domain requires human judgment
        for boundary_type, boundary_domains in self.HUMAN_REQUIRED_DOMAINS.items
            if domain in boundary_domains:
                return BoundaryCheck(
                    requires_human=True,
                    boundary_type=boundary_type,
                    reason=f"Decision involves {domain}, which requires human ju
                    recommended_action="PRESENT_ANALYSIS_REQUEST_DECISION"
                )

        return BoundaryCheck(requires_human=False)
```

Success Criteria:

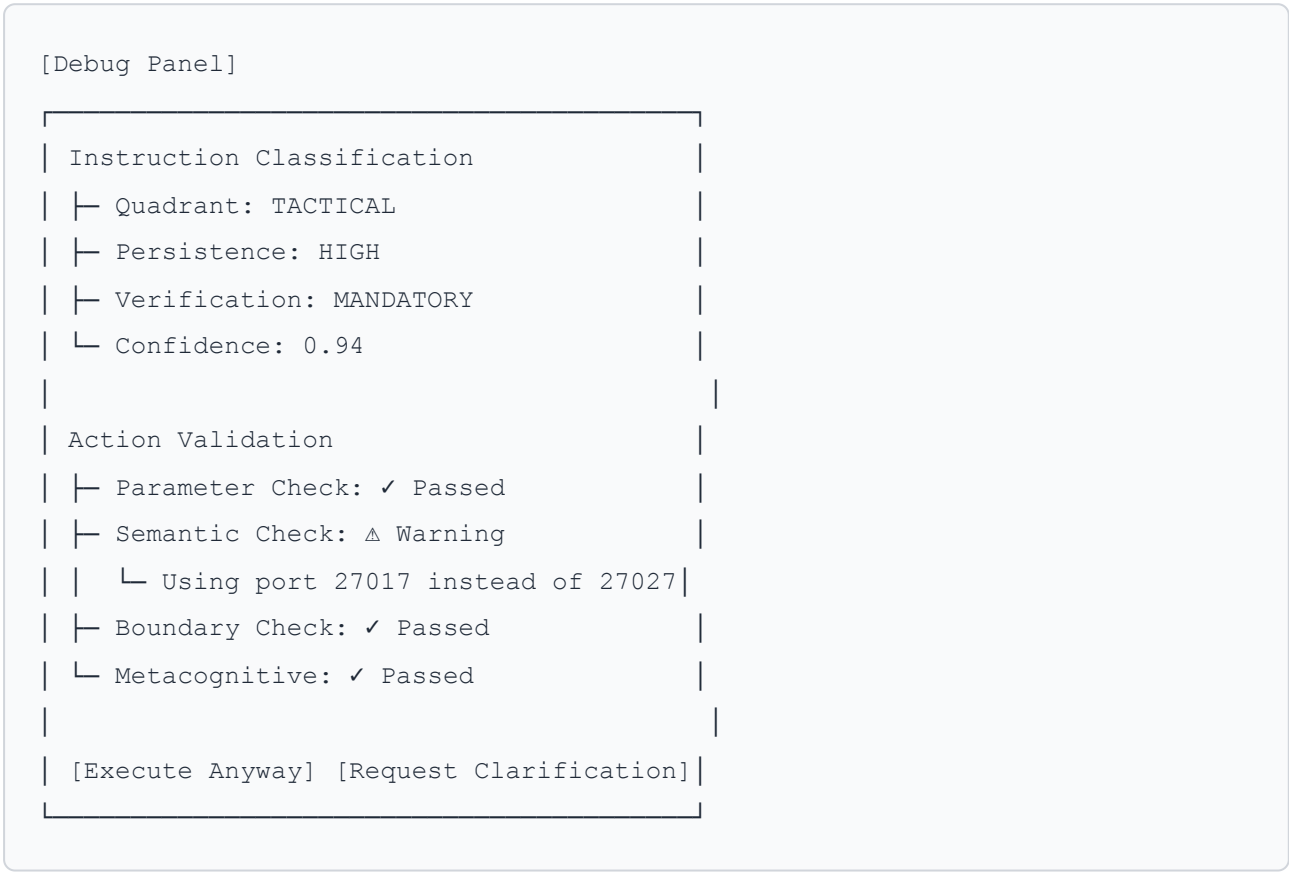
- 100% detection of values/purpose/agency decisions
- Appropriate routing to human judgment
- Clear explanations for why human input required
- User feedback indicates appropriate boundaries

2.5 Enhanced UI/UX (Month 6)

User-Facing Improvements:

- Transparency: Show classification and verification status
- Control: Users can adjust verification sensitivity
- Learning: System learns from user corrections
- Feedback: Clear explanations for interventions

Features:



Success Criteria:

- Users can understand why interventions occur
- Users can provide corrections to improve system
- Transparency increases trust (measured via survey)
- Debug mode valuable for power users

Phase 2 Success Criteria

Quantitative:

- EIVR: <2% (target: 90% reduction from baseline)
- False Positive Rate: <10%

- User Satisfaction: +85% improvement over baseline
- Latency: <100ms total overhead per action
- Error Recovery: 95% of conflicts caught before execution

Qualitative:

- User trust measurably improved
- Support ticket reduction for reliability issues
- Engineering team satisfied with maintainability
- Clear competitive differentiation demonstrated

Go/No-Go Decision: If targets met, begin Phase 3 optimization. If partially met, iterate for 1-2 months before proceeding.

Phase 3: Optimization and ML Enhancement (Months 7-12)

Objective

Optimize performance, add ML-based prediction and adaptation, achieve <50ms latency and 95% classification accuracy.

Deliverables

3.1 ML-Enhanced Classification (Months 7-8)

Improvements:

- Transformer-based instruction classification
- Transfer learning from Claude base models
- Continual learning from user corrections
- Confidence calibration

Target Metrics:

- Accuracy: >95% (up from 90%)
- Latency: <15ms (down from 25ms)

- Calibration: >0.90 correlation

3.2 Predictive Intervention (Months 8-9)

New Capability: Predict errors before they occur

Approach:

- Historical error pattern analysis
- Context-based risk assessment
- Preemptive verification for high-risk scenarios

Example:

```
class PredictiveInterventionSystem:
    def assess_action_risk(self, action: ProposedAction, context: ConversationCo
        # Historical patterns
        historical_risk = self.assess_historical_risk(action, context)

        # Context pressure
        context_risk = self.context_monitor.assess_pressure(context)

        # Action characteristics
        action_risk = self.assess_action_characteristics(action)

        # Combined risk score
        total_risk = self.combine_risk_scores(historical_risk, context_risk, act

        if total_risk > self.INTERVENTION_THRESHOLD:
            return RiskAssessment(
                level="HIGH",
                score=total_risk,
                recommendation="INCREASE_VERIFICATION",
                explanation="Similar actions have 23% error rate in this context
            )

        return RiskAssessment(level="NORMAL")
```

Target Metrics:

- Error prediction accuracy: >80%
- Preemptive intervention effectiveness: 50% error reduction
- False alarm rate: <5%

3.3 Performance Optimization (Months 9-10)

Focus Areas:

- Caching classification results for repeated patterns
- Parallel validation pipeline
- Optimized semantic matching
- Reduced model inference latency

Target Metrics:

- Total latency: <50ms (down from 100ms)
- Throughput: >100 actions/second
- Memory footprint: <200MB additional

3.4 Adaptive Verification (Months 10-11)

New Capability: System learns optimal verification intensity per user

Approach:

- User-specific verification profiles
- Context-adaptive thresholds
- Learning from user override patterns

Benefits:

- Power users get less interruption
- Novice users get more assistance
- Personalized safety/autonomy balance

3.5 Comprehensive Validation and Hardening (Month 12)

Activities:

- Red team testing

- Edge case collection and fixing
- Performance benchmarking
- Security audit
- Prepare for production scaling

Deliverables:

- Security review passed
- Performance benchmarks met
- Edge case coverage >99%
- Production deployment plan

Phase 3 Success Criteria

Quantitative:

- EIVR: <1% (sustained)
- Classification Accuracy: >95%
- Latency: <50ms total
- User Satisfaction: >8.5/10
- False Positive Rate: <5%

Qualitative:

- System feels responsive and unobtrusive
- Interventions are consistently helpful
- Users trust the system more than without framework
- Ready for scaling to additional products

Phase 4: Scaling to Claude Products (Year 2)

Objective

Extend Tractatus framework to all Claude products with product-specific adaptations.

4.1 Claude.ai Integration (Months 13-15)

Scope: Extend framework to claude.ai web interface

Challenges:

- More varied use cases than Claude Code
- Broader user base (consumer vs. developer)
- Different interaction patterns

Adaptations:

- Use-case-specific classification models
- Lighter-weight verification for casual use
- Opt-in advanced verification for critical tasks

Success Criteria:

- EIVR: <2% for claude.ai
- User satisfaction: +50% improvement
- Adoption: >70% of users try enhanced reliability features

4.2 Claude API Integration (Months 15-18)

Scope: Provide Tractatus framework as optional API feature

Features:

- Developers can enable framework via API parameter
- Custom verification rules per application
- Audit logs for compliance use cases

Use Cases:

- Enterprise applications requiring audit trails
- High-stakes applications (medical, legal, financial)
- Safety-critical systems

Success Criteria:

- API documentation complete
- > *20% of enterprise customers adopt*
- Measurable reliability improvements in customer applications

4.3 Enterprise Features (Months 18-21)

Scope: Enterprise-specific extensions

Features:

- Custom boundary definitions per organization
- Compliance reporting and audit trails
- Integration with enterprise governance systems
- Multi-level approval workflows

Success Criteria:

- > *10 enterprise customers deployed*
- Compliance requirements met (SOC2, HIPAA, etc.)
- Demonstrated ROI for enterprise adoption

4.4 Research and Publication (Months 18-24)

Activities:

- Publish research papers on framework effectiveness
- Open-source reference implementation
- Engage with AI safety community
- Contribute to alignment research

Deliverables:

- Peer-reviewed publication(s)
- Open-source library released
- Conference presentations
- Community feedback integrated

4.5 Continuous Improvement (Ongoing)

Ongoing Activities:

- Monitor metrics across all products
- Iterate on ML models
- Expand boundary definitions
- Incorporate user feedback
- Advance research

Risk Management

Technical Risks

Risk	Probability	Impact	Mitigation
Latency exceeds targets	Medium	High	Early performance testing, optimization sprints
False positive rate too high	Medium	High	Graduated response system, user override capability
Classification accuracy insufficient	Low	Medium	ML enhancement, active learning from corrections
Integration complexity	Medium	Medium	Phased approach, dedicated integration team

Product Risks

Risk	Probability	Impact	Mitigation
User resistance to interventions	Medium	High	Transparent explanations, opt-in controls, gradual rollout
Competitive pressure to rush	Low	High	Clear go/no-go criteria, quality gates
Scope creep	Medium	Medium	Strict phase definitions, change control

Research Risks

Risk	Probability	Impact	Mitigation
Framework doesn't scale to AGI	Low	Very High	Theoretical analysis, simulation testing
Unforeseen failure modes	Medium	High	Red team testing, continuous monitoring
Industry doesn't adopt approach	Medium	Medium	Open-source release, publication, community engagement

---

Success Metrics Summary

Phase 1 (Months 1-3): Proof-of-Concept

- **EIVR**: <5% (80% reduction)
- **False Positive Rate**: <15%
- **Latency**: <50ms
- **Go/No-Go**: Quantitative targets met

Phase 2 (Months 4-6): Production Integration

- **EIVR**: <2% (90% reduction)
- **False Positive Rate**: <10%

- **User Satisfaction**: +85%
- **Latency**: <100ms
- **Error Recovery**: 95%

Phase 3 (Months 7-12): Optimization

- **EIVR**: <1%
- **Classification Accuracy**: >95%
- **Latency**: <50ms
- **User Satisfaction**: >8.5/10
- **False Positive Rate**: <5%

Phase 4 (Year 2): Scaling

- **Products**: Claude.ai, API, Enterprise
- **Enterprise Adoption**: >10 customers
- **Publication**: Peer-reviewed research
- **Open Source**: Reference implementation

Resource Requirements

Team Composition

Phase 1-2 (Months 1-6):

- 1 Research Scientist (ML/NLP)
- 2 Software Engineers (Backend)
- 1 Product Manager
- 0.5 UX Designer
- 0.5 DevOps Engineer

Phase 3 (Months 7-12):

- +1 ML Engineer (optimization)
- +1 Software Engineer (scaling)

- +0.5 Security Engineer

Phase 4 (Year 2):

- +2 Software Engineers (product integrations)
- +1 Product Manager (enterprise)
- +1 DevOps Engineer (scaling)
- +0.5 Technical Writer (documentation)

Infrastructure

- Development environment: Standard cloud resources
- ML training: GPU cluster for model optimization
- A/B testing: 10% traffic allocation initially
- Monitoring: Enhanced logging and metrics collection

Budget (Rough Estimate)

- **Year 1**: $800K - $1.2M (team + infrastructure)
- **Year 2**: $1.5M - $2.0M (expanded team + scaling)
- **Total 2-Year**: $2.3M - $3.2M

ROI Calculation:

- Reduced support costs: ~$200K/year (reliability issues)
- Increased user retention: ~$500K/year (improved trust)
- Enterprise adoption: ~$1M+/year (new revenue)
- **Payback period**: ~18-24 months

Decision Points and Gates

Gate 1 (End of Month 3): Continue to Phase 2?

Criteria:

- EIVR reduction >80%

- False positive rate <15%
- No major technical blockers
- User feedback neutral or positive

Decision: Go / Iterate / Stop

Gate 2 (End of Month 6): Continue to Phase 3?

Criteria:

- EIVR <2%
- User satisfaction +85%
- Production stability demonstrated
- Clear path to optimization

Decision: Go / Iterate / Stop

Gate 3 (End of Month 12): Scale to other products?

Criteria:

- All Phase 3 targets met
- Security review passed
- Performance benchmarks met
- Business case validated

Decision: Go / Iterate / Stop

Gate 4 (End of Month 18): Enterprise rollout?

Criteria:

- Claude.ai integration successful
- API adoption demonstrated
- Enterprise features ready
- Compliance requirements met

Decision: Go / Iterate / Pause

Conclusion

This roadmap provides a concrete path from today's LLM architectures to Tractatus-enhanced systems with structural safety guarantees. The phased approach allows for:

1. **Early validation** - Prove concept within 3 months
2. **Risk mitigation** - Clear gates prevent over-commitment
3. **Iterative improvement** - Learn and adapt at each phase
4. **Measured scaling** - Expand only after demonstrated success

The framework is ambitious but achievable. The need is urgent. The time to start is now.

This roadmap represents a collaborative development plan by John Stroh and Claude AI Assistant for implementing structural AI safety guarantees in LLM systems.
