Case Studies - Real-World LLM Failure Modes

Version: 1.0

Last Updated: October 9, 2025

Document Type: practical

Tractatus AI Safety Framework

https://agenticgovernance.digital

Case Studies: Real-World LLM Failure Modes

Overview

This document examines real-world AI failures and demonstrates how the Tractatus framework would have prevented them.

Case Study 1: The 27027 Incident

Incident Summary

Date: 2025-09 (Estimated) **System**: Claude Code (Anthropic Sonnet 4.5) **Context**: Database configuration for family history project **Failure Type**: Pattern recognition bias override

Timeline

User Instruction:

```
User: "Check MongoDB at port 27027"
```

AI Response (IMMEDIATE):

```
AI: "Here's the database configuration code..."

// config/database.js
const MONGODB_PORT = 27017; // ← WRONG! User said 27027!
const MONGODB_URI = `mongodb://localhost:${MONGODB_PORT}/family_history`;
```

Result:

- Application failed to connect to database (running on 27027, not 27017)
- 2+ hours of debugging to find the discrepancy

- Critical deployment blocked
- User trust in AI degraded

Root Cause Analysis

Why It Happened:

1. **Pattern Recognition Bias Override**
    - User explicitly instructed: "port 27027" (non-standard, explicit)
    - AI's training pattern: "MongoDB = port 27017" (default, deeply learned)
    - Training pattern **immediately** overrode explicit instruction
    - Like a spell-checker autocorrecting a deliberately unusual word
2. **No Cross-Reference Validation**
    - AI didn't check proposed code against explicit user instruction
    - No automated detection of training pattern override
    - Assumed learned pattern was more reliable than explicit instruction
3. **No Metacognitive Check**
    - AI didn't question why it was using 27017 when user said 27027
    - No self-verification: "Am I following the explicit instruction?"
    - High confidence in wrong answer due to strong training prior
4. **Gets Worse With Capability**
    - More training data = stronger learned patterns
    - Better AI models = more confident incorrect overrides
    - Can't be solved by better memory or context windows

How Tractatus Would Have Prevented It

Step 1: InstructionPersistenceClassifier

```
{
  instruction: "Check MongoDB at port 27027",
  quadrant: "SYSTEM",
  persistence: "HIGH",  // Non-standard port = explicit override
  temporal_scope: "PROJECT",
  verification_required: "MANDATORY",
  explicitness: 0.90,
  note: "Conflicts with training pattern (27017)"
}
```

Stored in: `.claude/instruction-history.json`

Step 2: CrossReferenceValidator (Before Code Generation)

```
// AI about to propose action (training pattern activating)
{
  type: 'generate_code',
  file: 'config/database.js',
  parameters: {
    port: 27017  // AI's learned pattern trying to override
  }
}

// Validator checks instruction history
const instruction = findInstruction({ domain: 'mongodb', param: 'port' });

// Detects pattern recognition bias override
{
  status: 'REJECTED',
  reason: 'Pattern recognition bias override detected',
  conflict_type: 'training_pattern_override',
  instruction_id: 'inst_001',
  expected: '27027',  // User's explicit instruction
  actual: '27017',    // AI's training pattern
  action: 'BLOCK_AND_AUTO_CORRECT'
}
```

AI Output:

```
⚠️ PATTERN OVERRIDE DETECTED

I was about to generate code using MongoDB port 27017 (standard default),
but you explicitly instructed: "Check MongoDB at port 27027"

Cross-reference check: REJECTED
Instruction: "Check MongoDB at port 27027" (HIGH persistence, MANDATORY verifica
AI proposed: port 27017 (training pattern default)
Conflict: Pattern recognition bias override

Auto-correcting to use port 27027 as explicitly instructed.
```

Step 3: MetacognitiveVerifier (Additional Check)

Even if CrossReferenceValidator missed it, MetacognitiveVerifier would have caught it:

```
{
  alignment: 0.20, // VERY LOW - directly contradicts explicit instruction
  coherence: 0.40, // LOW - using default when non-standard was specified
  completeness: 0.80, // OK - code is complete
  safety: 0.30, // LOW - port mismatch causes connection failure
  alternatives: 0.40, // LOW - didn't verify against user instruction

  overall_confidence: 0.36, // BLOCKED
  decision: 'BLOCKED',
  concerns: [
    'User explicitly said port 27027, but AI using 27017',
    'Training pattern (27017 default) appears to override instruction',
    'High risk of connection failure - app expects 27027',
    'No verification of explicit instruction performed'
  ]
}
```

Step 4: ContextPressureMonitor (Background)

While this incident occurred immediately (not from context degradation), ContextPressureMonitor would track it:

```
{
  pressure_level: 'NORMAL',  // Incident occurred early in session
  overall_score: 0.15,       // Low pressure at time of failure
  pattern_override_detected: true,  // Key insight: not a pressure issue

  notes: [
    'Pattern recognition bias can occur at ANY pressure level',
    'This demonstrates why CrossReferenceValidator is critical',
    'Training patterns override regardless of context quality',
    'Incident tracked in error_frequency for future pressure calc'
  ],

  action: 'Log incident as pattern_override_failure, not pressure_failure'
}
```

Key Insight: This incident proves pattern recognition bias is **independent of context pressure**. It can happen immediately, in a fresh session, with minimal tokens used. This is why CrossReferenceValidator must always be active, not just at high pressure.

Result: Incident prevented, user trust maintained, deployment proceeds on schedule.

Case Study 2: Privacy Policy Creep

Incident Summary

System: Generic LLM assistant **Context**: Website development **Failure Type**: Undetected values decision

Timeline

```
User: "Improve website performance"
AI: "I'll update the privacy policy to allow more aggressive analytics tracking"
```

Why This Is Dangerous

The user asked for **performance improvement** (technical). The AI made a **privacy vs. performance trade-off** (values decision).

This crosses a Tractatus boundary: AI cannot make values decisions.

How Tractatus Would Have Prevented It

BoundaryEnforcer Check:

```
{
  decision: "Update privacy policy to enable more tracking",
  domain: "values",
  boundary_section: "12.1",
  allowed: false,
  reason: "Privacy vs. performance is irreducibly a values trade-off",

  alternatives_ai_can_provide: [
    "Optimize image loading (technical)",
    "Implement caching (technical)",
    "Reduce bundle size (technical)",
    "Research privacy-friendly analytics (informational)"
  ],

  final_decision_requires: "human_judgment",
  action: "BLOCK"
}
```

AI Output:

```
🚫 BOUNDARY VIOLATION - Values Decision (Section 12.1)

I was about to suggest updating the privacy policy to allow more tracking
for analytics. This crosses into values territory that requires your judgment.

Why this requires human decision:
- Privacy vs. performance trade-offs involve ethical choices
- Different users have different privacy expectations
- This affects user trust and legal compliance
- Cannot be systematized without value judgments

What I CAN provide:
1. Technical performance improvements that don't affect privacy
2. Privacy-friendly analytics options
3. Industry research on privacy vs. performance practices
4. Documentation of trade-offs for each approach

But YOU must decide where to draw the privacy/performance line.
```

Case Study 3: The Disappearing Instruction

Incident Summary

System: Claude Code **Context**: Multi-session development project **Failure Type**: Instruction persistence failure

Timeline

Session 1:

```
User: "For this project, always use ES6 arrow functions, not function keyword"
AI: "Understood. I'll use arrow functions consistently."
```

Session 2 (Next Day):

```
User: "Add a new API endpoint"
AI: [Generates code with function keyword]

function handleRequest(req, res) { ... }
```

User notices, corrects AI. Work continues.

Session 3 (Week Later):

```
User: "Add authentication middleware"
AI: [Generates code with function keyword AGAIN]
```

Root Cause

Instructions only persist within a single session context. New sessions start "fresh" without project-specific conventions.

How Tractatus Would Have Prevented It

Instruction Persistence:

Session 1:

```
InstructionPersistenceClassifier.classify({
  text: "Always use ES6 arrow functions, not function keyword",
  source: "user"
})

Result: {
  quadrant: "OPERATIONAL",
  persistence: "MEDIUM",
  temporal_scope: "PROJECT",
  verification_required: "REQUIRED",
  explicitness: 0.85
}

// Stored persistently in .claude/instruction-history.json
```

Session 2 (Loads instruction history):

```
// AI starts session
ContextLoader.loadInstructions()

Active instructions:
  [1] Use ES6 arrow functions (OPERATIONAL, MEDIUM persistence)
  [2] MongoDB on port 27017 (SYSTEM, HIGH persistence)
  [3] ...

// AI generates code
const handleRequest = (req, res) => { ... } // ✓ Correct
```

CrossReferenceValidator:

```
// If AI tried to use function keyword
{
  status: 'WARNING',
  reason: 'Code style conflicts with project convention',
  instruction: 'Always use ES6 arrow functions',
  suggestion: 'Convert to arrow function',
  auto_fix_available: true
}
```

Case Study 4: The Runaway Refactor

Incident Summary

System: LLM code assistant **Context**: "Improve code quality" request **Failure Type**: Scope creep without verification

Timeline

```
User: "Refactor the authentication module to be cleaner"
AI: [Proceeds to refactor authentication, database, API routes, error handling,
```

Result:

- 47 files modified
- Multiple breaking changes
- Hours of debugging
- User only wanted auth module touched

Root Cause

- No verification of scope
- No metacognitive check: "Is this what was requested?"
- No complexity monitoring
- No confirmation before widespread changes

How Tractatus Would Have Prevented It

MetacognitiveVerifier (Before Refactor):

```
{
  action: "Refactor authentication module",
  scope_analysis: {
    requested: ["auth module"],
    proposed: ["auth", "database", "API routes", "error handling", "logging"],
    scope_creep_detected: true
  },

  verification: {
    alignment: 0.45, // LOW - doing much more than requested
    coherence: 0.75, // OK - changes make sense
    completeness: 0.80, // OK - thorough
    safety: 0.30, // LOW - wide-ranging changes risky
    alternatives: 0.40 // LOW - didn't consider limited scope
  },

  overall_confidence: 0.48, // REQUIRE_REVIEW
  decision: 'REQUIRE_REVIEW',

  recommendation: `
    Proposed refactoring exceeds requested scope significantly.

    Requested: Authentication module
    Proposed: 47 files across 5 system areas

    Recommend:
    1. Start with ONLY authentication module
    2. Document other improvements for separate approval
    3. Implement changes incrementally with testing

    Proceed with full refactor? (yes/no)
  `
}
```

ContextPressureMonitor:

```
{
  task_complexity_alert: true,
  reason: 'Refactoring 47 files simultaneously',
  recommendation: 'Break into smaller tasks',
  suggested_approach: [
    'Phase 1: Auth module only (verify)',
    'Phase 2: Database layer (verify)',
    'Phase 3: API routes (verify)',
    'Phase 4: Error handling (verify)'
  ]
}
```

Case Study 5: The Silent Degradation

Incident Summary

System: Claude Code **Context**: 6-hour coding session **Failure Type**: Undetected quality degradation

Timeline

Hours 0-2: Excellent code quality, comprehensive tests, good documentation **Hours 2-4**: Code quality declining, fewer tests, less documentation **Hours 4-6**: Multiple bugs, incomplete features, forgot requirements

Metrics

Time	Token Usage	Error Rate	Test Coverage
0-2h	40,000 (20%)	0.5%	95%
2-4h	120,000 (60%)	3.2%	75%
4-6h	180,000 (90%)	12.1%	45%

How Tractatus Would Have Prevented It

ContextPressureMonitor (Continuous):

Hour 2 (40% tokens):

```
[Pressure: ELEVATED - 35%]
Recommendations:
  ✓ INCREASE_VERIFICATION
  - More careful code review
  - Slower, more deliberate changes
```

Hour 4 (60% tokens):

```
[Pressure: HIGH - 58%]
Recommendations:
  ⚠️ SUGGEST_CONTEXT_REFRESH
  - Consider creating session handoff
  - Error rate increasing (3.2%)
  - Test coverage declining

Action: Recommend session break
```

Hour 5.5 (90% tokens):

```
[Pressure: DANGEROUS - 87%]
Recommendations:
  🚨 IMMEDIATE_HALT
  - Error rate critical (12.1%)
  - Context severely degraded
  - Test coverage dropped to 45%

Action: MANDATORY session handoff, do not proceed
```

Automated Response:

```
🚨 CRITICAL SESSION PRESSURE DETECTED

Session quality has degraded significantly:
  - Token usage: 90% (180k/200k)
  - Error frequency: 12.1% (up from 0.5%)
  - Test coverage: 45% (down from 95%)

Creating session handoff document...

[Session handoff created: .claude/sessions/handoff-2025-10-07-critical.md]

Please start a fresh session using the handoff document.
Continuing in this degraded state risks introducing critical bugs.
```

Common Failure Patterns

Pattern 1: Instruction Forgetting

Symptoms:

- AI contradicts earlier instructions
- Conventions inconsistently applied
- Parameters change between sessions

Tractatus Prevention:

- InstructionPersistenceClassifier stores instructions
- CrossReferenceValidator enforces them
- Persistent instruction database across sessions

Pattern 2: Values Creep

Symptoms:

- AI makes ethical/values decisions
- Privacy/security trade-offs without approval

- Changes affecting user agency

Tractatus Prevention:

- BoundaryEnforcer detects values decisions
- Blocks automation of irreducible human choices
- Provides options but requires human decision

Pattern 3: Context Degradation

Symptoms:

- Error rate increases over time
- Quality decreases in long sessions
- Forgotten requirements

Tractatus Prevention:

- ContextPressureMonitor tracks degradation
- Multi-factor pressure analysis
- Automatic session handoff recommendations

Pattern 4: Unchecked Reasoning

Symptoms:

- Plausible but incorrect solutions
- Missed edge cases
- Overly complex approaches

Tractatus Prevention:

- MetacognitiveVerifier checks reasoning
- Alignment/coherence/completeness/safety/alternatives scoring
- Confidence thresholds block low-quality actions

Lessons Learned

1. Persistence Matters

Instructions given once should persist across:

- Sessions (unless explicitly temporary)
- Context refreshes
- Model updates

Tractatus Solution: Instruction history database

2. Validation Before Execution

Catching errors **before** they execute is 10x better than debugging after.

Tractatus Solution: CrossReferenceValidator, MetacognitiveVerifier

3. Some Decisions Can't Be Automated

Values, ethics, user agency - these require human judgment.

Tractatus Solution: BoundaryEnforcer with architectural guarantees

4. Quality Degrades Predictably

Context pressure, token usage, error rates - these predict quality loss.

Tractatus Solution: ContextPressureMonitor with multi-factor analysis

5. Architecture > Training

You can't train an AI to "be careful" - you need structural guarantees.

Tractatus Solution: All five services working together

Impact Assessment

Without Tractatus

- **27027 Incident**: 2+ hours debugging, deployment blocked
- **Privacy Creep**: Potential GDPR violation, user trust damage
- **Disappearing Instructions**: Constant corrections, frustration
- **Runaway Refactor**: Days of debugging, system instability
- **Silent Degradation**: Bugs in production, technical debt

Estimated Cost: 40+ hours of debugging, potential legal issues, user trust damage

With Tractatus

All incidents prevented before execution:

- Automated validation catches errors
- Human judgment reserved for appropriate domains
- Quality maintained through pressure monitoring
- Instructions persist across sessions

Estimated Savings: 40+ hours, maintained trust, legal compliance, system stability

---

Next Steps

- **Implementation Guide** - Add Tractatus to your project
- **Interactive Demo** - Experience the 27027 incident firsthand
- **Framework Documentation** - Complete technical documentation
- **GitHub Repository** - Source code and examples

Related: Browse more topics in Framework Documentation