Cassie Agent Gap Analysis & Improvement Roadmap

Date: January 27, 2026

Prepared For: Engineering & Support Operations Leadership

Status: CRITICAL GAPS IDENTIFIED - IMMEDIATE ACTION REQUIRED

⚠️ Important Context

Broader Initiative: This gap analysis is part of a larger proposal on how AI projects should be architected in a **layered manner** with clear separation of concerns. I am developing a **reusable skills library framework** that enables systematic knowledge capture and rapid deployment across AI agent systems. This analysis serves as a practical example of applying that framework to accelerate improvements in existing products.

Analysis Goal: Identify quick wins and structural improvements that can help Cassie achieve measurable results **at minimal time investment** while establishing patterns for future AI agent development across the organization.

Analysis Methodology: This gap analysis was conducted through independent codebase review, MCP server testing, production case data analysis, and EDA on historical Slack support cases. It represents a technical architecture assessment based on observed system behavior and documented patterns.

Validation Status: This analysis has **NOT yet been validated** against the engineering team's current implementation plans, strategic direction, or architectural decisions. The findings and recommendations should be reviewed as **preliminary observations** pending alignment discussions.

Required Next Steps:

- Alignment discussion with Goutham, Ashish, and Sudhanshu on strategic vision and scope

- Validation of identified gaps against planned improvements
- Refinement of recommendations based on team feedback and architectural constraints
- Integration with reusable skills library framework proposal
- Prioritization alignment with existing roadmap and resource allocation

Feedback Requested: Please review this document and identify: (1) findings that align with your vision, (2) areas where additional context is needed, (3) recommendations that conflict with planned direction, (4) interest in the layered architecture and skills library approach, and (5) gaps in understanding that should be addressed before finalizing this analysis.

Executive Summary

Current State: The Cassie Agent platform demonstrates sophisticated classification capabilities and seamless enterprise data integration, yet operates with **exploratory investigation guidance** rather than **precision diagnostic protocols**. The system accurately identifies issue categories but lacks the embedded intelligence to autonomously execute targeted diagnostic sequences.

Business Impact: 100% of reviewed cases escalate to manual intervention despite the platform possessing comprehensive data access and all necessary technical capabilities for autonomous resolution. This represents a fundamental execution gap between classification intelligence and problem resolution.

Root Cause: The investigation layer employs **general exploration strategies** rather than **pattern-specific diagnostic protocols**. Instead of executing predetermined sequences of validation checks derived from expert knowledge, the system conducts broad searches across available data sources. This approach lacks the surgical precision required for high-volume, pattern-based support scenarios.

Strategic Recommendation: Transform the investigation framework from exploratory search to protocol-driven diagnostics by embedding domain expertise directly into the autonomous investigation layer. This evolution enables the platform to execute expert-defined diagnostic sequences systematically, achieving 60-70% autonomous resolution while maintaining intelligent escalation for complex scenarios requiring human expertise.

Part 1: Analysis of Production Failures

Case 1: GPS Provider Null Timestamp (Case #2693837)

What Cassie Did:

```
✓ Identified: FOURKITES_LOAD_NOT_TRACKING
✓ Reasoning: "loads exist but are not receiving GPS/location updates"
✗ Action: Escalated to manual intervention
```

What React Agent SHOULD Have Done (with enhanced prompt):

```
Step 1: Cassie routes to React Agent with "gps_provider_prompt.py" (DONE) ✓
Step 2: gps_provider_prompt.py should contain embedded diagnostic logic:

   PLAYBOOK: GPS_PROVIDER_NULL_TIMESTAMP
   - Check: Query GPS provider API logs via Redshift MCP
   - Look for: "undefined method `to_datetime' for nil:NilClass"
   - If found: Root cause = GPS provider returning null timestamps
   - Draft response template: "GPS provider [PROVIDER_NAME] returned invalid data..."

Step 3: React Agent executes playbook and generates response automatically
```

Gap Identified:

- ❌ gps_provider_prompt.py lacks embedded diagnostic playbooks
- ❌ No pattern-specific checks (just generic "search logs")
- ❌ No decision trees for common GPS provider errors
- ❌ React Agent has tools but no guidance on WHAT to check

Bot Feasibility from Categories Sheet: LOW to Should be MEDIUM/HIGH with proper diagnostics

Case 2: Old Location Rejection (Case #2688628)

What Cassie Did:

```
✓ Identified: FOURKITES_LOAD_NOT_TRACKING
✓ Reasoning: "location updates not being ingested despite positive ping tests"
✗ Stopped at: "tracking integration failure" (generic)
✗ Did NOT drill into: WHY locations are being rejected
```

What React Agent SHOULD Have Done (with enhanced prompt):

```
Step 1: Cassie routes to React Agent with general_investigation_prompt.py (DONE) ✓
Step 2: general_investigation_prompt.py should contain embedded diagnostic logic:

   PLAYBOOK: OUTLIER_DETECTION_OLD_LOCATIONS
   - Step 1: Check outlier detection logs via Redshift MCP
   - Step 2: Look for: "Speed-based ping rejection" or "Stale coordinates"
   - Step 3: If found: Sub-category = "Old/Stale Location Timestamps"
   - Step 4: Match to pattern from categories sheet (Pattern ID: Outlier Detection >
   - Step 5: Draft response template with GPS provider guidance

Step 3: React Agent executes playbook systematically instead of generic search
```

Gap Identified:

- ✗ general_investigation_prompt.py has no embedded playbooks
- ✗ Says "search databases" but not WHAT to search for
- ✗ No pattern matching logic from Arpit's category sheet
- ✗ React Agent stopped at generic "tracking integration failure"

Bot Feasibility: Currently marked as MEDIUM to Should be HIGH with pattern matching

---

Case 3: Trailer vs Truck GPS (Case #2692749)

What Cassie Did:

```
✗ Got stuck in decision loop
✗ Did not progress to diagnostic checks
✗ Escalated to manual intervention
```

What React Agent SHOULD Have Done (with enhanced prompt):

```
Step 1: Cassie routing logic gets stuck (decision loop bug) - ROUTING ISSUE ✗
Step 2: IF properly routed, React Agent needs embedded diagnostic logic:

   PLAYBOOK: ASSET_MISMATCH_TRAILER_VS_TRUCK
   - Step 1: Check asset assignment via Redshift MCP (truck vs trailer)
   - Step 2: Check carrier capabilities (supports truck GPS or trailer GPS?)
   - Step 3: If mismatch: Root cause = Wrong tracking method applied
   - Step 4: Draft response: "Load assigned trailer but carrier only supports truck GI

Step 3: THIS CASE NEEDS TWO FIXES:
   Fix 1: Cassie decision loop bug (routing layer)
   Fix 2: React Agent prompt needs asset mismatch playbook
```

Gap Identified:

- ❌ **Routing Bug:** Cassie decision loop prevented case from reaching React Agent
- ❌ **Prompt Gap:** Even if routed, React Agent prompt lacks asset verification playbook
- ❌ No carrier capability check in any specialized prompt
- ❌ Basic configuration check logic missing

Bot Feasibility: Should be MEDIUM/HIGH - this is a configuration check

---

Case 4: ELD Not Enabled (Case #2682612)

What Cassie Did:

```
✗ Got stuck in decision loop
✗ Did not execute basic first-step check
✗ Escalated to manual intervention
```

What React Agent SHOULD Have Done (with enhanced prompt):

```
Step 1: Cassie routing logic gets stuck (decision loop bug) - ROUTING ISSUE ✗
Step 2: IF properly routed, React Agent needs THIS embedded in prompt:

   PLAYBOOK: ELD_NOT_ENABLED_AT_NETWORK_LEVEL
   - Step 1: Query network configuration via Redshift MCP
   - Step 2: Check: Is ELD tracking enabled for shipper-carrier pair?
   - Step 3: If NO: Root cause identified (Pattern: Configuration/Network Issues)
   - Step 4: Draft response: "ELD tracking not enabled. Please enable in network conf
   - Step 5: Bot Feasibility: HIGH (simple boolean check)

Step 3: THIS IS THE MOST CRITICAL GAP - simplest case but 0% automated
```

Gap Identified:

- ✗ **Routing Bug:** Cassie decision loop prevented proper routing
- ✗ **CRITICAL Prompt Gap:** No ELD configuration check playbook in ANY prompt
- ✗ This is HIGH bot feasibility (simple boolean) but not implemented
- ✗ Should be first-step diagnostic in OTR tracking prompts

Bot Feasibility: HIGH (as per category sheet) - this is a critical failure

---

Part 2: Pattern Analysis from Categories Sheet

High-Value Patterns (Bot Feasibility: HIGH)

These should be **100% automated** but currently escalate to manual:

Pattern	Current Bot	Should Be	Gap
Check Call Expiry	N/A	100% Auto	No expiry check implemented
Configuration/Network Issues	Categorizes	100% Auto	Doesn't check Connect config
ELD/GPS Provider Issues	Categorizes	90% Auto	Doesn't check asset assignment
Sibling Load Issues	N/A	80% Auto	No sibling group logic

Impact: 40-50% of tickets could be auto-resolved with proper configuration checks.

Medium-Value Patterns (Bot Feasibility: MEDIUM)

These should provide **diagnostic guidance** and **draft responses**:

Pattern	Current Bot	Should Be	Gap
Outlier Detection	Stops at category	Draft response	No log analysis
Status Code Processing	Stops at category	Configuration check	No status mapping check
Geofence Issues	N/A	Config verification	No geofence check
Infrequent Updates	N/A	Frequency analysis	No ping interval check

Impact: 30-40% of tickets could get detailed diagnostic guidance instead of generic escalation.

Complex Patterns (Bot Feasibility: LOW)

These require **engineering investigation** but bot should still provide context:

Pattern	Current Behavior	Should Provide
Infrastructure Issues	Generic escalation	Specific log evidence + engineering ticket template
Kafka Lag	Not detected	Lag metrics + impact analysis
Database Timeout	Not detected	Timeout logs + query performance data

Impact: 10-20% of tickets - bot provides rich context for engineering escalation.

Part 3: Root Cause Analysis - Why Is The System Failing?

Current System Architecture: Intelligence Without Action

The Cassie Agent platform represents a sophisticated AI investigation system with enterprise-grade infrastructure already in place. The architecture demonstrates advanced classification capabilities, seamless integration with five production data sources, and intelligent routing mechanisms. However, the system currently operates as an **intelligent observer** rather than an **autonomous problem solver**.

Current State: Classification Excellence, Execution Gap

What The System Does Well:

- **Intelligent Case Classification:** Accurately categorizes 80-90% of support cases into the correct high-level category
- **Multi-Source Intelligence:** Orchestrates real-time data access across five enterprise systems (data warehouse, CRM, knowledge base, FourKites APIs, collaboration platforms)
- **Adaptive Routing:** Dynamically selects specialized investigation strategies based on case characteristics
- **Domain Expertise:** Maintains 14 specialized investigation domains covering the full spectrum of support scenarios

Where The System Stops:

After achieving accurate classification, the system transitions from **strategic intelligence** to **generic exploration**. Rather than executing targeted diagnostic protocols, it conducts broad searches across data sources. This approach—effective for research but inefficient for problem resolution—results in 100% escalation to manual intervention despite having all necessary data and capabilities for autonomous resolution.

Target State: From Intelligence to Autonomous Action

The evolution requires embedding **domain-specific diagnostic intelligence** into the investigation layer. Instead of exploratory searches, the system will execute **precision diagnostic protocols**—predetermined sequences of checks that systematically eliminate possibilities and identify root causes with measurable confidence.

Strategic Capabilities of Enhanced Architecture:

- **Autonomous Diagnostic Execution:** Self-directed investigation following expert-defined playbooks, eliminating guesswork
- **Evidence-Based Decision Making:** Systematic collection and evaluation of diagnostic evidence with confidence scoring
- **Pattern Recognition at Scale:** Instant matching against 100+ known issue patterns with proven resolution paths
- **Intelligent Escalation:** Automatic resolution for high-confidence cases (60-70%), detailed diagnostic reports for ambiguous scenarios
- **Continuous Learning:** Pattern library expansion as new issues emerge and resolution strategies are validated

Issue 1: Investigation Strategies Lack Precision Diagnostic Protocols

Problem: The investigation layer operates with exploratory guidance rather than definitive diagnostic protocols, creating an execution gap between classification intelligence and problem resolution.

Current Approach: Exploratory Search Methodology

The system currently employs a **broad-spectrum search strategy**—instructing the investigation engine to explore available data sources and identify relevant patterns. While this approach enables flexibility across diverse scenarios, it lacks the **surgical precision** required for high-volume, pattern-based support cases.

Characteristics of Current Methodology:

- Directs investigation toward *general data exploration* rather than specific diagnostic checks
- Relies on pattern discovery rather than pattern matching against known issues
- Positions the AI as a research assistant rather than an autonomous diagnostician
- Results in thorough analysis but inconsistent outcomes and extended resolution times

Target Approach: Protocol-Driven Diagnostic Intelligence

The enhanced system will operate with **expert-encoded diagnostic protocols**—predetermined investigation sequences that mirror how top-performing support engineers approach specific issue categories. Each protocol represents validated diagnostic logic, refined from thousands of historical cases.

Protocol Characteristics:

- **Deterministic Logic:** Clear decision trees with defined checks, expected outcomes, and branching paths
- **Evidence-Based Confidence:** Quantifiable confidence scores based on diagnostic evidence strength
- **Fail-Fast Design:** Early identification of conclusive patterns, minimizing unnecessary investigation steps
- **Escalation Intelligence:** Automatic distinction between high-confidence autonomous resolution and cases requiring human expertise

Example Protocol Categories:

- **Configuration Validation Protocols:** System-level feature enablement, network-level settings verification
- **Asset Assignment Protocols:** Tracking prerequisite validation, carrier capability confirmation
- **Data Quality Protocols:** Provider API health checks, data integrity validation
- **Integration Status Protocols:** Service connectivity verification, authentication validation

Strategic Requirement: Transform investigation guidance from exploratory search directives to precision diagnostic protocols, embedding domain expertise directly into the autonomous investigation framework.

Issue 2: Enterprise Data Access Without Strategic Utilization

Problem: The system possesses comprehensive access to five enterprise intelligence sources but lacks strategic direction on how to leverage these capabilities for targeted diagnostics.

Current Infrastructure Capabilities:

The platform maintains production-grade integration with critical enterprise systems:

- **Data Warehouse Intelligence:** Real-time access to network configurations, load tracking metrics, GPS provider logs, historical tracking data
- **CRM Integration:** Complete customer case history, account relationships, support interaction patterns
- **Knowledge Repository:** Internal documentation, troubleshooting procedures, best practices, resolution frameworks
- **FourKites API Gateway:** Live system status, real-time tracking data, operational metrics
- **Collaboration Platform:** Engineering tickets, known issues, feature documentation

The Infrastructure-Strategy Gap:

Despite having **comprehensive data access**, the investigation layer operates with **general exploration directives** rather than **targeted intelligence gathering protocols**. This creates a scenario where the system has the keys to every door but lacks the roadmap indicating which doors to open for specific scenarios.

Strategic Transformation Required:

The evolution requires translating **data availability** into **intelligence utilization protocols**—prescriptive frameworks that direct the autonomous investigation engine to specific data sources, specific queries, and specific evidence indicators for each diagnostic scenario.

Protocol-Driven Data Utilization:

- **Configuration Intelligence Protocols:** Directed queries to network configuration repositories with specific validation checks
- **Operational Metrics Protocols:** Targeted extraction of tracking status indicators, assignment verification, capability validation
- **Log Analysis Protocols:** Pattern-specific searches for known error signatures, data quality indicators, integration health markers
- **Historical Pattern Protocols:** Comparative analysis against similar cases with validated resolutions

Outcome: Transform from *"search these sources for relevant information"* to *"execute this specific sequence of checks across these data points to validate or eliminate these hypotheses."*

Issue 3: Domain Expertise Exists in Static Form, Not Operational Intelligence

Problem: The organization has captured comprehensive diagnostic intelligence across 100+ issue patterns, but this expertise exists as reference documentation rather than operational automation protocols.

Existing Intellectual Capital:

The support organization has systematically documented diagnostic knowledge representing thousands of hours of expert analysis:

✓ **Comprehensive Pattern Taxonomy:** Complete categorization of support scenarios across all product domains

✓ **Root Cause Frameworks:** Documented relationships between symptoms, diagnostic checks, and underlying issues

✓ **Automation Feasibility Assessment:** Expert evaluation of which patterns are candidates for autonomous resolution

✓ **Standardized Resolution Protocols:** Validated approaches for addressing each pattern category

The Knowledge-to-Action Gap:

This diagnostic intelligence exists in a form optimized for **human consumption** rather than **machine execution**. The autonomous investigation system operates independently of this documented expertise, essentially starting from first principles for each investigation rather than leveraging validated diagnostic frameworks.

Current State Limitations:

✗ Documented patterns serve as reference material but don't drive autonomous investigation strategy

✗ Automation feasibility ratings inform planning but don't trigger differentiated autonomous behavior

✗ Standardized resolution protocols exist but aren't embedded in automated response generation

❌ The system rediscovers patterns that have already been extensively documented and validated

Strategic Transformation: From Documentation to Operational Intelligence

Required Evolution: Convert static diagnostic documentation into executable investigation protocols that drive autonomous system behavior.

Transformation Approach:

- **Protocol Encoding:** Convert documented patterns into machine-executable diagnostic sequences
- **Domain Specialization:** Create investigation strategies tailored to specific problem domains (infrastructure, configuration, integration, data quality)
- **Confidence-Driven Automation:** Leverage feasibility assessments to determine autonomous resolution vs. expert escalation thresholds
- **Template-Driven Resolution:** Embed validated resolution frameworks into automated response generation

Strategic Benefits:

- Thousands of hours of expert analysis becomes instantly available to every investigation
- Consistent diagnostic approach across all autonomous investigations
- Continuous improvement as patterns are validated and refined through operational data
- Institutional knowledge persists independently of individual expert availability

---

Issue 4: Classification Engine Exhibits Decision Paralysis

Problem: The intelligent routing layer occasionally enters recursive classification cycles, preventing cases from progressing to investigation execution despite having sufficient information for categorization.

Architectural Layer: This represents a decision-making framework issue in the classification engine, independent of investigation capability gaps.

Observed Behavior:

In specific scenarios, the classification system cycles through multiple categorization attempts without reaching a decisive outcome. Cases become trapped in analytical loops, with the system repeatedly re-evaluating the same evidence without progressing to investigation execution.

Documented Impact:

- Cases #2692749 and #2682612 demonstrate indefinite classification cycling
- System remains in "analysis" state rather than transitioning to "action" state
- Even with enhanced diagnostic protocols available, affected cases cannot access them

Root Cause Analysis:

The classification engine employs an **elimination-based decision framework** (determining what a case is NOT) rather than a **positive-match framework** (determining what a case IS with measurable confidence). Without explicit confidence thresholds and fallback mechanisms, ambiguous cases lack a definitive path forward.

Strategic Resolution:

Required Evolution: Transform classification logic from elimination-based reasoning to confidence-scored positive matching with intelligent fallback mechanisms.

Enhanced Decision Framework:

- **Confidence-Based Routing:** Quantify classification confidence and route based on explicit thresholds
- **Intelligent Fallback:** Ambiguous cases default to general investigation rather than classification retry
- **Iteration Limits:** Prevent analytical paralysis through explicit classification attempt boundaries
- **Progressive Degradation:** Lower confidence thresholds with each iteration to ensure eventual progression

Note: This routing optimization operates independently of diagnostic protocol enhancement. Both improvements are required for comprehensive system advancement.

Strategic Framework: Optimal Placement of Diagnostic Intelligence

The system architecture comprises four distinct capability layers, each with specific responsibilities. Understanding the optimal placement of diagnostic intelligence requires clarity on the strategic role of each layer:

Layer 1: Classification & Routing Intelligence

Strategic Function: Rapid case categorization and intelligent routing to appropriate investigation strategies

Core Capability: Pattern recognition, intent classification, domain identification

Design Principle: Lightweight, fast decision-making focused exclusively on *what type of case* rather than *what is wrong*

Diagnostic Responsibility: None—classification layer does not execute investigations

Layer 2: Investigation Execution Engine

Strategic Function: Autonomous execution of investigation protocols with access to enterprise data sources

Core Capability: Multi-source intelligence gathering, evidence collection, systematic diagnostic execution

Design Principle: Flexible execution engine that adapts behavior based on strategic guidance

Diagnostic Responsibility: Protocol execution—receives diagnostic guidance and orchestrates data collection accordingly

Layer 3: Diagnostic Intelligence & Strategic Guidance

Strategic Function: Domain expertise encoding—the bridge between expert knowledge and autonomous execution

Core Capability: Pattern-specific diagnostic protocols, decision trees, evidence evaluation frameworks, resolution strategies

Design Principle: THIS IS THE OPTIMAL LAYER FOR DIAGNOSTIC INTELLIGENCE

Diagnostic Responsibility: Owns all diagnostic logic—what to check, in what sequence, how to evaluate results, when to escalate

Layer 4: Enterprise Data Intelligence

Strategic Function: Unified access to distributed enterprise systems

Core Capability: Data retrieval, query execution, cross-system integration

Design Principle: Pure data access layer without business logic or diagnostic intelligence

Diagnostic Responsibility: None—provides data when requested but does not drive investigation strategy

Strategic Recommendation: Embed Diagnostic Intelligence in the Strategic Guidance Layer (Layer 3)

Rationale for This Architectural Decision:

Why Not Classification Layer?

- Classification focuses on *categorization*, not *resolution*
- Adding diagnostic logic would compromise classification performance and speed
- Classification layer lacks direct access to enterprise data sources

Why Not Data Layer?

- Data systems provide *information*, not *intelligence*
- Diagnostic logic often requires synthesis across multiple data sources
- Business context and domain expertise belong in business logic layer, not data layer

Why Strategic Guidance Layer?

- Enables domain-specific specialization (OTR tracking intelligence, ocean shipping intelligence, network configuration intelligence)
- Bridges expert knowledge and autonomous execution through protocol encoding
- Provides investigation engine with clear, actionable diagnostic roadmaps
- Supports continuous improvement through protocol refinement without system architecture changes
- Maintains separation of concerns—routing, execution, intelligence, and data remain distinct

Part 4: Recommendations for Immediate Improvement

Priority 1: Fix Decision Loop Bug (Week 1)

Problem: Cases getting stuck in classification loop.

Fix:

1. Replace elimination-based logic with positive pattern matching
2. Add timeout/max-iterations guard
3. Add fallback: "Unable to classify" to route to Load Not Tracking playbook by default

Impact: Eliminate 50% of "stuck" cases immediately.

Priority 2: Embed Basic Diagnostic Playbooks in Prompts (Weeks 2-3)

Start with 5 HIGH-feasibility patterns embedded in specialized prompts:

1. ELD Not Enabled (Case #2682612 failure)

- **Prompt to enhance:** Create new otr_tracking_issues_prompt.py
- **Playbook:** Check network-level ELD configuration via Redshift MCP
- **Query guidance:** SELECT eld_tracking_enabled FROM network_configurations...
- **Auto-resolve:** YES (Bot Feasibility: HIGH)
- **Response template:** "Enable ELD tracking at network level"

2. Asset Not Assigned (T000008)

- **Prompt to enhance:** otr_tracking_issues_prompt.py

- **Playbook:** Check truck/trailer/device assignment via Redshift MCP
- **Query guidance:** SELECT truck_number, trailer_number, device_id FROM loads...
- **Auto-resolve:** YES (Bot Feasibility: HIGH)
- **Response template:** "Contact carrier to assign asset"

3. Check Call Expiry (T000007)

- **Prompt to enhance:** fourkites_connect_carrier_onboarding_prompt.py
- **Playbook:** Check last delivered load date via Redshift MCP
- **Query guidance:** SELECT MAX(delivery_date) FROM loads WHERE network_id = ...
- **Auto-resolve:** YES (Bot Feasibility: HIGH)
- **Response template:** "Network inactive - verify carrier is active"

4. Duplicate SCAC in Network

- **Prompt to enhance:** fourkites_connect_carrier_onboarding_prompt.py
- **Playbook:** Query carriers with same SCAC via Redshift MCP
- **Query guidance:** SELECT carrier_name FROM carriers WHERE scac = ... GROUP BY scac HAVING COUNT(*) > 1
- **Auto-resolve:** PARTIAL (needs human verification)
- **Response template:** "Found duplicate SCACs: [list] - verify correct carrier"

5. Feature Flag Disabled

- **Prompt to enhance:** general_investigation_prompt.py OR create config_issues_prompt.py
- **Playbook:** Check feature flags via Redshift MCP
- **Query guidance:** SELECT feature_enabled FROM feature_flags WHERE shipper_id = ...
- **Auto-resolve:** YES (if simple enable) (Bot Feasibility: HIGH)
- **Response template:** "Enable feature X in configuration"

> **Impact:** 40-50% of cases auto-resolved with these 5 playbooks embedded in prompts.

Priority 3: Add MCP Query Guidance to Prompts (Weeks 3-4)

Current: React Agent connects to MCPs successfully, but prompts don't guide WHAT to query.

Fix - Embed specific query guidance in prompts:

```
File: otr_tracking_issues_prompt.py

** PLAYBOOK: GPS_PROVIDER_LOG_ANALYSIS

Step 1: Query GPS provider logs via Redshift MCP
   USE THIS QUERY:
   SELECT timestamp, provider_name, error_message, location_data
   FROM gps_provider_api_logs
   WHERE load_id = {load_id}
     AND timestamp >= NOW() - INTERVAL '7 days'
     AND error_message IS NOT NULL
   ORDER BY timestamp DESC
   LIMIT 100

Step 2: Analyze query results
   LOOK FOR THESE PATTERNS:
   - "null timestamp" or "timestamp is nil" → GPS Provider Null Timestamp Issue
   - "API timeout" or "connection refused" → GPS Provider API Failure
   - "invalid coordinates" → GPS Provider Data Quality Issue

Step 3: If pattern matched:
   Root Cause: GPS Provider Issue (specific error)
   Confidence: 90%
   Draft Response: Use template_gps_provider_issue with provider name and error details

This explicit query guidance tells React Agent EXACTLY what to do,
removing guesswork and ensuring consistent diagnostics.
```

Impact: Enable systematic log analysis and root cause drilling (fixes Case #2693837, #2688628).

Priority 4: Convert Category Sheet to Prompt Playbooks (Week 4)

Action Items:

1. Export Arpit's category sheet to structured format (YAML or JSON)
2. Create domain-specific prompt files (8-10 new prompts):
    - `otr_tracking_issues_prompt.py` - OTR-specific playbooks
    - `ocean_tracking_prompt.py` - Ocean-specific playbooks
    - `network_configuration_prompt.py` - Config checks
    - `gps_provider_issues_prompt.py` - Enhanced GPS playbooks
    - `outlier_detection_prompt.py` - Data quality checks
    - etc.
3. Each prompt file includes embedded playbooks from category sheet:

```
File: otr_tracking_issues_prompt.py

def get_otr_tracking_issues_prompt() -> str:
    return """
    OTR TRACKING ISSUES DIAGNOSTIC SPECIALIST

    ** PLAYBOOK 1: ASSET_NOT_ASSIGNED
    Pattern ID: T000008 - ELD/GPS Provider Issues > No Asset Assigned
    Bot Feasibility: HIGH

    DIAGNOSTIC STEPS:
    1. Query via Redshift MCP:
       SELECT truck_number, trailer_number, device_id
       FROM loads WHERE load_id = {load_id}

    2. Evaluate: Are all fields NULL?

    3. If yes:
       Root Cause: No asset assigned by carrier
       Confidence: 100%
       Action: AUTO-RESOLVE

    4. Draft response:
       "The carrier has not assigned a truck, trailer, or device to this load.
        This is preventing ELD tracking from starting.
        Resolution: Contact carrier to assign an asset."

    ** PLAYBOOK 2: ELD_NOT_ENABLED
    ...

    Execute playbooks in order until root cause found.
    """
```

4. Update Cassie's prompt_router.py to map categories to new prompts

Impact: Structured diagnostic framework embedded in prompts (fixes generic investigation problem).

Priority 5: Add Auto-Response Generation (Week 5)

Current: Bot identifies issue then escalates.

Should Be: Bot drafts customer response automatically.

Implementation:

```
# In each pattern YAML
auto_response_template: |
  Subject: Re: Load Not Tracking - {load_number}

  Thank you for contacting FourKites Support.

  We have identified the issue with load {load_number}:

  Root Cause: {root_cause_description}

  Evidence:
  {evidence_list}

  Resolution Steps:
  {resolution_steps}

  Next Steps:
  {next_steps}

  If this does not resolve the issue, please reply and we will investigate further.
```

Quality Gate: Human reviews auto-generated response before sending (initially).

Impact: Reduce manual response drafting time from 10-15 min to 2-3 min review.

Part 5: Proposed Revised Timeline

Phase 1: Quick Wins (Weeks 1-2) - NEW

Task	Outcome	Timeline
Fix decision loop bug	Eliminate stuck cases	Week 1
Implement 5 basic diagnostic checks	40-50% auto-resolution	Weeks 1-2
Integrate OTR MCP calls	Enable log analysis	Week 2

Milestone: Bot goes from 0% auto-resolution to 40-50% auto-resolution.

Phase 2: Skills Library Integration (Weeks 3-5)

Task	Outcome	Timeline
Convert category sheet to YAML skills	Structured playbooks	Week 3
Implement playbook execution engine	Sequential diagnostic checks	Week 4
Add auto-response generation	Draft customer responses	Week 5
Testing on 50 historical cases	Validate 80%+ accuracy	Week 5

Milestone: Bot provides diagnostic guidance + draft responses for 80% of cases.

Phase 3: Scale to All Categories (Weeks 6-8)

Task	Outcome	Timeline
Add 20+ HIGH-feasibility patterns	60-70% auto-resolution	Weeks 6-7
Add 30+ MEDIUM-feasibility patterns	Diagnostic guidance for 90%	Weeks 7-8
Improve log parsing and error detection	Better root cause accuracy	Week 8

Milestone: Bot handles 90% of OTR Load Not Tracking cases with minimal manual intervention.

Phase 4: Production Release (Weeks 9-11)

Task	Outcome	Timeline
Shadow mode: Bot provides suggestions, human decides	Build confidence	Week 9
Phased rollout: 10% to 50% to 100% of cases	Monitor quality	Week 10
Feedback loop: Capture false positives/negatives	Continuous improvement	Week 11

Milestone: Bot in production with 60%+ auto-resolution rate, 85%+ customer satisfaction.

Part 6: Comparison - Current Plan vs Recommended Plan

Aspect	Current Plan	Recommended Plan	Impact
Timeline	11 weeks	11 weeks	Same
Auto-Resolution Target	Not specified	60-70% by Week 11	Clear success metric
Quick Wins	None (starts with refactor)	40-50% auto-res by Week 2	Immediate value
Diagnostic Execution	Implied but not explicit	Explicit playbook engine	Core functionality
Category Sheet Usage	Review only	Convert to Skills Library	Structured approach
MCP Integration	Data sources mentioned	Explicit OTR MCP calls	Clear implementation
Auto-Response	Not mentioned	Implemented in Week 5	Reduce manual work
Validation	Live testing at end	Historical + live testing	Better confidence

Part 7: Success Metrics

Baseline (Current State - January 2026)

Metric	Current Value
Auto-Resolution Rate	0% (all cases escalate to manual)
Average Investigation Time	15-30 minutes per case
Category Accuracy	80-90% (bot identifies correct high-level category)
Root Cause Accuracy	0% (bot doesn't drill to root cause)
Customer Satisfaction	N/A (no auto-responses generated)

Target (Phase 1 Complete - April 2026)

Metric	Target Value
Auto-Resolution Rate	60-70% for HIGH-feasibility cases
Diagnostic Guidance	90% of cases get specific diagnostic steps
Average Investigation Time	5-8 minutes (70% reduction)
Category + Root Cause Accuracy	85%+ (bot identifies specific pattern)
Customer Satisfaction	80%+ on auto-generated responses
Manual Intervention	Only for LOW-feasibility or ambiguous cases (10-20%)

Measurement Approach

Week-by-Week Tracking:

- **Week 1:** Fix decision loop to Measure: % of stuck cases eliminated
- **Week 2:** Add 5 basic checks to Measure: % auto-resolved (target: 40%)
- **Week 3:** Category sheet conversion to Measure: Pattern coverage %
- **Week 4:** Playbook execution to Measure: Root cause accuracy
- **Week 5:** Auto-response to Measure: Customer satisfaction on drafts
- **Weeks 6-8:** Scale patterns to Measure: Auto-resolution climbing to 60%
- **Weeks 9-11:** Production rollout to Measure: All metrics + false pos/neg rate

---

Part 8: Risk Analysis & Mitigation

Risk 1: Over-Engineering Too Soon

Problem: The current implementation plan includes "collaborative agents framework" in Week 3.

Concern: Multi-agent orchestration may be premature when basic checks aren't working.

Mitigation:

- Start with simple playbook execution (single-agent model)
- Prove value with basic diagnostics first (Weeks 1-5)
- Add multi-agent complexity only if needed (Phase 2+)

Principle: "Do the simplest thing that could possibly work"

Risk 2: Data Source Dependencies

Problem: "Build missing data sources" could become a blocker.

Current State: 10 MCP servers exist (Athena, Redshift, Tracking API, etc.)

Mitigation:

- Verify existing MCPs can provide required data
- Use workarounds (direct API calls) if MCP gaps exist
- Don't block on MCP development - use what's available

Principle: "Work with what you have, improve incrementally"

Risk 3: Log Access Complexity

Problem: "Improve logs" is vague and could become scope creep.

Mitigation:

- Define specific logs needed for top 10 patterns (Week 1)
- Verify log access via existing OTR MCP (Week 2)
- If logs not accessible, adjust playbooks to work with available data
- Don't wait for perfect log access before launching

Principle: "Good enough to start, perfect over time"

Risk 4: Category Sheet Interpretation

Problem: 100+ row sheet with subjective "Bot Feasibility" ratings.

Mitigation:

- Start with 20 HIGH-feasibility patterns (clear automated checks)
- Human-review MEDIUM-feasibility patterns before automating
- Leave LOW-feasibility for manual intervention (expected)
- Validate bot feasibility ratings with actual case data

Principle: "Trust but verify"

Part 9: Technical Architecture Recommendations

Current Architecture (Inferred)

```
User Input to Cassie Agent (Classification) to Category Identified to "Manual
Review Required" to STOP
```

Problem: No execution layer after classification.

Recommended Architecture

```
User Input ↓ Cassie Agent (Intent Classification) ↓ Category Router ↓ Playbook
Executor ↓ ┌──────────────────────────────────────┐ │ Diagnostic Checks
(Parallel) │ │ - Check 1: ELD enabled? │ │ - Check 2: Asset assigned? │ │ -
Check 3: GPS logs for errors? │ │ - Check 4: Outlier detection logs? │
└──────────────────────────────────────┘ ↓ Evidence Collector ↓ Pattern Matcher
(against Skills Library) ↓ Confidence Scorer ↓
┌──────────────────────┬──────────────────────┐ │ Confidence ≥ 90% │ Confidence
< 90% │ │ Auto-Response │ Manual Review │ │ (draft for human │ (with evidence)
│ │ approval) │ │ └──────────────────────┴──────────────────────┘
```

Key Components to Build

1. Playbook Executor

Purpose: Run diagnostic checks sequentially until root cause found or evidence exhausted.

```python
class PlaybookExecutor:
    def execute(self, category: str, load_id: str):
        # Load playbook for category
        playbook = load_playbook(category)

        # Execute checks sequentially
        for step in playbook.diagnostic_steps:
            result = self.execute_check(step, load_id)

            if result.is_conclusive:
                return result  # Early exit on definitive answer

        # If no conclusive result, return collected evidence
        return Evidence(
            category=category,
            checks_performed=...,
            recommendation="Manual review required"
        )
```

2. MCP Integration Layer

Purpose: Standardize how Cassie calls OTR MCP and other data sources.

```python
class MCPClient:
    def __init__(self):
        self.otr_mcp = OTRMCPServer(url=..., auth=...)
        self.connect_mcp = ConnectMCPServer(url=..., auth=...)

    def check_eld_enabled(self, shipper_id: str, carrier_id: str) -> bool:
        result = self.connect_mcp.query(
            "network_configuration",
            params={"shipper": shipper_id, "carrier": carrier_id}
        )
        return result.get("eld_tracking_enabled", False)

    def get_asset_assignment(self, load_id: str) -> dict:
        result = self.otr_mcp.query(
            "asset_info",
            params={"load_id": load_id}
        )
        return {
            "truck_number": result.get("truck_number"),
            "trailer_number": result.get("trailer_number"),
            "device_id": result.get("device_id")
        }
```

3. Pattern Matcher

Purpose: Match diagnostic results to patterns from Skills Library.

```
class PatternMatcher:
    def __init__(self):
        self.patterns = load_all_patterns_from_skills_library()

    def match(self, evidence: Evidence) -> Pattern:
        for pattern in self.patterns:
            if pattern.matches(evidence):
                return pattern

        return Pattern.UNKNOWN
```

4. Auto-Response Generator

Purpose: Generate customer response from pattern template.

```
class ResponseGenerator:
    def generate(self, pattern: Pattern, evidence: Evidence) -> str:
        template = pattern.auto_response_template

        # Fill template variables
        response = template.format(
            load_number=evidence.load_id,
            root_cause_description=pattern.description,
            evidence_list=self.format_evidence(evidence),
            resolution_steps=pattern.resolution_steps,
            next_steps=pattern.next_steps
        )

        return response
```

Part 10: Actionable Next Steps (This Week)

For Engineering Team

Monday:

1. Review this gap analysis report
2. Identify decision loop bug location in Cassie codebase
3. Prioritize 5 basic diagnostic checks for Week 1-2 implementation

Tuesday-Wednesday:

4. Fix decision loop bug (replace elimination logic with positive matching)
5. Add timeout guard and fallback routing

Thursday-Friday:

6. Implement first diagnostic check: "ELD enabled at network level?"
7. Test on Case #2682612 (should now auto-resolve)
8. Implement second check: "Asset assigned?"
9. Test on historical cases

Target: By Friday EOW, bot auto-resolves 2 pattern types (20-30% of common cases).

For Support Operations Team

This Week:

1. Validate the 5 HIGH-feasibility patterns proposed for Week 1-2:
   - ELD Not Enabled
   - Asset Not Assigned
   - Check Call Expiry

    - Duplicate SCAC
    - Feature Flag Disabled
2. Provide 10 historical cases for each pattern (50 cases total) for testing
3. Review and approve auto-response templates for these 5 patterns

Target: **Engineering has validated test cases and approved templates by Friday.**

For Product Team

This Week:

1. Define success metrics for Phase 1:
    - What % auto-resolution is acceptable for production release?
    - What customer satisfaction threshold must be met?
    - What false positive rate is tolerable?
2. Approve phased rollout plan:
    - Week 9: Shadow mode (bot suggests, human decides)
    - Week 10: 10% of cases to monitor quality
    - Week 10: 50% of cases to monitor quality
    - Week 11: 100% rollout

Target: **Clear go/no-go criteria defined by Friday.**

Part 11: Conclusion

What's Working

✓ **Routing Architecture:** Cassie correctly routes cases to React Agent with specialized prompts

✓ **MCP Infrastructure:** 5 production MCP servers (Redshift, Salesforce, Knowledge, Support AI, Atlassian) operational

✓ **React Agent:** Successfully connects to all MCPs and executes queries

✓ **Category Taxonomy:** Comprehensive 100+ row sheet with root cause patterns (Arpit's sheet)

✓ **Specialized Prompts:** 14 domain-specific prompts exist as framework

What's Broken

✕ **Generic Prompts:** Specialized prompts lack embedded diagnostic playbooks (just say "search databases")

✕ **No Pattern-Specific Logic:** Prompts don't contain decision trees for specific issue patterns

✕ **Category Sheet Not Embedded:** Arpit's 100+ patterns exist but aren't in prompts

✕ **Decision Loop Bug:** Cassie routing logic gets stuck on some cases (prevents reaching React Agent)

✕ **No Auto-Resolution:** 100% of cases escalate to manual despite having data and tools

Critical Path Forward

Timeframe	Actions	Outcome
Week 1-2	Fix decision loop + implement 5 basic checks	**40-50% auto-resolution**
Week 3-5	Skills Library integration + playbook execution	**Diagnostic guidance for 80%**
Week 6-8	Scale to all HIGH/MEDIUM patterns	**60-70% auto-resolution**
Week 9-11	Production rollout with monitoring	**Live at scale**

Key Insight

The system architecture is CORRECT (Cassie routes, React Agent executes, MCPs provide data). The GAP is in the SPECIALIZED PROMPTS - they lack embedded diagnostic playbooks.

The current implementation plan has the right phases but needs MORE SPECIFICITY on WHERE diagnostic logic goes:

1. **NOT in Cassie:** Cassie is a router, diagnostics belong in React Agent layer
2. **NOT in MCPs:** MCPs are data sources, not logic owners
3. **YES in Prompts:** Embed playbooks in specialized prompts (otr_tracking_issues_prompt.py, etc.)
4. Category sheet conversion: FROM Excel TO embedded prompt playbooks (not YAML skills)
5. Auto-response generation: Via response templates in playbooks
6. Week-by-week auto-resolution metrics (currently vague)

Recommendation

Approve the proposed 11-week timeline BUT require explicit focus on prompt enhancement:

1. **Week 1:** Fix Cassie decision loop bug (routing layer)

2. **Weeks 2-3:** Create 2-3 domain-specific prompts with embedded playbooks:
   - otr_tracking_issues_prompt.py (5 HIGH-feasibility playbooks)
   - network_configuration_prompt.py (config checks)
   - gps_provider_issues_prompt.py (enhanced GPS playbooks)
3. **Weeks 4-5:** Convert Arpit's category sheet patterns to prompt playbooks (20+ patterns)
4. **Weeks 6-8:** Scale to all domains (50+ patterns across 8-10 prompts)
5. **Metrics:** 40% auto-resolution by Week 2, 60% by Week 8
6. **Clarify:** Diagnostic logic goes IN PROMPTS, not in Cassie or MCPs

With these clarifications targeting the correct layer (specialized prompts), the plan is solid and achievable.

Report Prepared By: AI R&D Solutions Engineer

Date: January 27, 2026

Next Review: February 3, 2026 (after Week 1 quick wins)