Script Analysis Comprehensive Fix & Enhancement Plan

Date: January 9, 2026 **Status**: Planning Complete - Ready for Implementation
Priority: High - Core Feature Enhancement

Executive Summary

This document outlines a comprehensive plan to transform the Script Analysis feature from a basic chatbot interface into a **production-grade, agentic AI-powered analysis system** that leverages LangGraph 1.0, Model Context Protocol (MCP), multi-agent orchestration, and 2026 best practices.

Key Goals

1. **Connect frontend to LangGraph backend** for true multi-agent analysis
2. **Implement streaming analysis** with real-time progress updates
3. **Add human-in-the-loop workflows** for critical reviews
4. **Integrate MCP servers** for external tool access
5. **Enable internet research** during analysis
6. **Visualize agent orchestration** for transparency
7. **Add state persistence** for resumable workflows

Current State Analysis

✅ What We Have

Backend (AI Service)

- **LangGraph 1.0 Production Orchestrator** (`langgraph_production.py`)
- StateGraph workflow with checkpointing
- Four production tools: analyze_powershell_script, security_scan, quality_analysis, generate_optimizations
- Human-in-the-loop support
- PostgreSQL checkpointing capability

- Latest GPT-5.2-codex model integration

- **LangGraph API Endpoints** (`langgraph_endpoints.py`)

- `/langgraph/analyze` - Full script analysis
- `/langgraph/feedback` - Human feedback integration
- `/langgraph/health` - Service health check
- `/langgraph/info` - Orchestrator metadata

- `/langgraph/batch-analyze` - Batch processing

- **Legacy Analyzer** (`script_analyzer.py`)

- OpenAI API integration
- Caching with Redis/Disk
- Vector embeddings (text-embedding-3-large)
- Async/concurrent processing

Frontend

- **ScriptAnalysis.tsx** - Basic analysis UI
- Tabs: Overview, Security, Quality, Performance, Parameters, AI Assistant
- Simple AI chat using `/chat` endpoint
- Static analysis display
- Score visualizations

Infrastructure

- Docker Compose setup with all services
- PostgreSQL with pgvector extension
- Redis caching layer
- All services running and healthy

❌ Critical Gaps Identified

1. Frontend-Backend Disconnection

- **Problem**: Frontend calls legacy `/chat` endpoint instead of `/langgraph/analyze`
- **Impact**: Missing LangGraph multi-agent orchestration, tool execution, checkpointing
- **Evidence**: `grep "langgraph" frontend/src --include="*.tsx"` returns zero results

2. No Streaming Support

- **Problem**: Analysis runs in black box, no progress updates
- **Impact**: Poor UX for long-running analysis (can take 30-60 seconds)
- **Missing**: Real-time tool execution updates, agent reasoning visibility

3. No Human-in-the-Loop UI

- **Problem**: Backend supports human review, frontend doesn't
- **Impact**: Can't leverage collaborative analysis workflows
- **Missing**: Review pause UI, feedback input, workflow resumption

4. No Tool Execution Visibility

- **Problem**: Users don't see which tools are running
- **Impact**: Analysis feels like magic, no transparency
- **Missing**: Tool progress indicators, intermediate results display

5. Missing MCP Integration

- **Problem**: No Model Context Protocol servers connected

- **Impact**: Can't access external tools (GitHub, docs, internet)
- **Missing**: MCP client, server configurations, tool discovery

6. No Internet Research Capability

- **Problem**: Analysis is limited to LLM knowledge
- **Impact**: Can't verify latest security vulnerabilities, best practices
- **Missing**: Web search tool, documentation fetching, trend analysis

7. Limited Analysis Depth

- **Problem**: Basic security patterns, simple quality checks
- **Impact**: Misses complex vulnerabilities, advanced optimizations
- **Missing**: Deep code path analysis, dependency scanning, performance profiling

8. No Multi-Agent Visualization

- **Problem**: Users don't see orchestration happening
- **Impact**: No trust in agentic system, appears as single AI
- **Missing**: Agent activity timeline, tool call logs, reasoning chains

9. No State Persistence UI

- **Problem**: Can't resume interrupted analysis
- **Impact**: Lost work if browser closes, no session continuity
- **Missing**: Thread ID management, checkpoint recovery UI

10. Missing Agent Collaboration Features

- **Problem**: No coordination between specialized agents
- **Impact**: Analysis lacks depth from multiple perspectives
- **Missing**: Security specialist, performance expert, best practices auditor agents

---

2026 Best Practices Research Summary

Key Findings from Industry Research

Agent Orchestration (LangGraph)

- LangGraph is the leading production framework for agentic workflows in 2026
- 86% of copilot spending going to agent-based systems
- 57% of organizations have agents in production
- 89% implement observability - critical for debugging and trust

Key Capabilities: - Graph-based state machines for complex workflows - Checkpointing for durable execution - Tool calling with automatic retries - Human-in-the-loop pause/resume - Streaming for real-time updates

Model Context Protocol (MCP)

- Open standard introduced by Anthropic, donated to Linux Foundation December 2025
- Adopted by OpenAI in March 2025 across all products
- Standardizes AI-to-tool integration
- Hundreds of community servers available

Core Primitives: - **Tools**: Functions AI can call - **Resources**: Data sources to read - **Prompts**: Reusable prompt templates

Security Analysis Best Practices

- Treat AI code as potentially vulnerable
- Automated security testing pipelines required
- AI should suggest secure alternatives automatically
- Integration with PSScriptAnalyzer and other tools

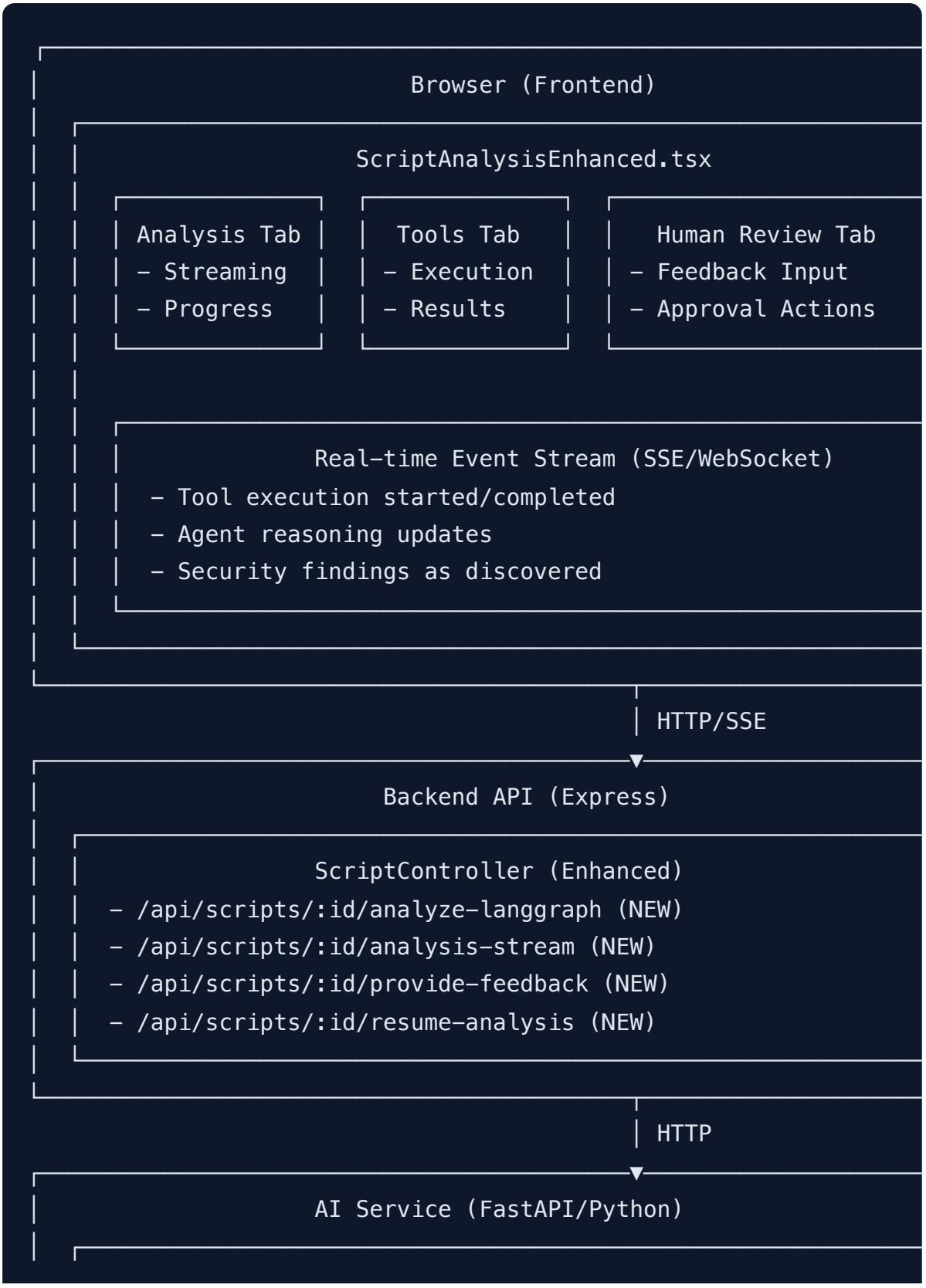
Agentic Workflows

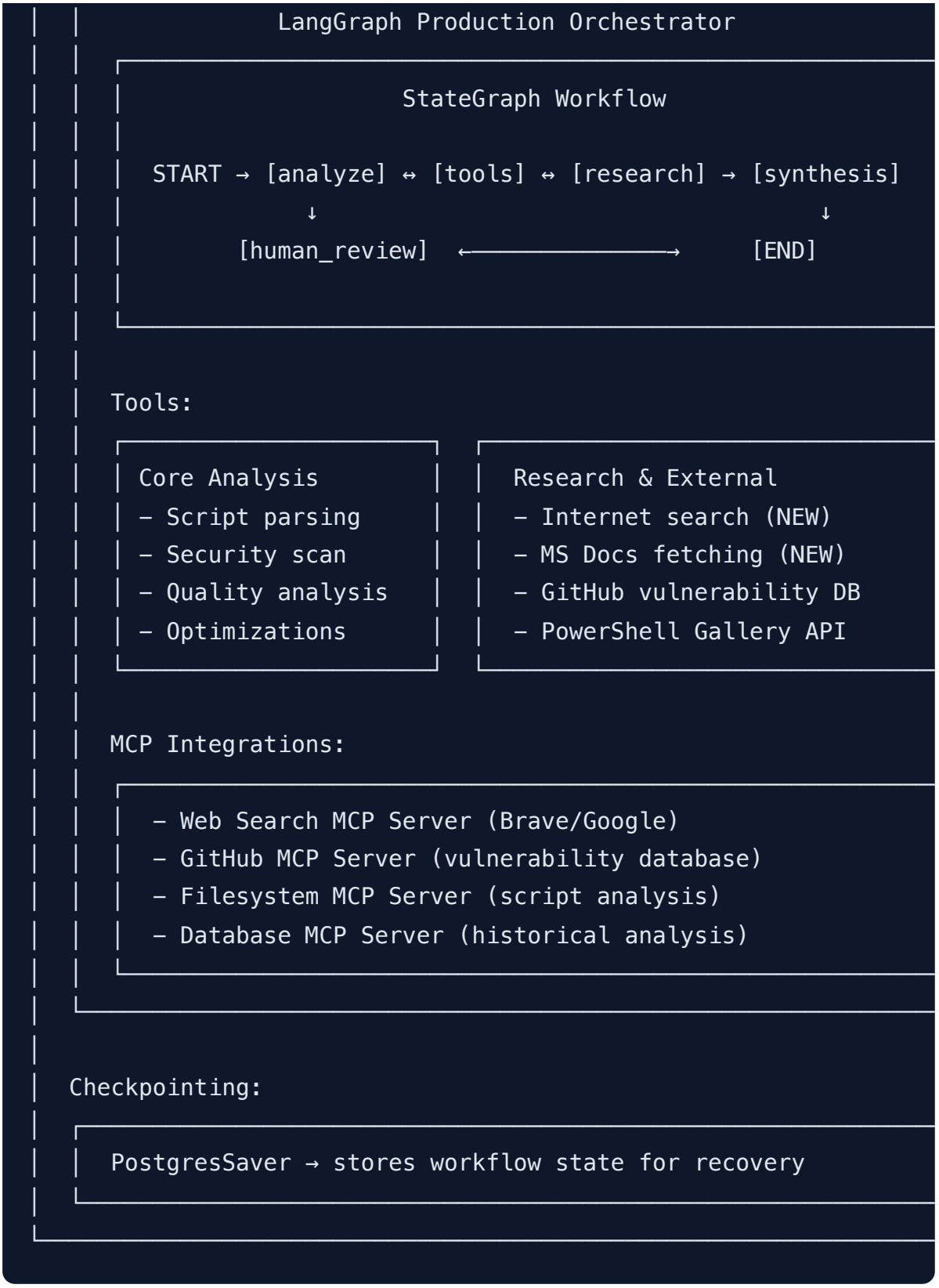
- Function calling reduces complexity

- Adaptive deliberation - agents decide between fast and deep reasoning
- Memory graphs for knowledge persistence
- Tool governance to enforce constraints

Proposed Architecture

High-Level System Design




LangGraph Production Orchestrator
StateGraph Workflow
START → [analyze] ↔ [tools] ↔ [research] → [synthesis]
[human_review]
[END]
Tools:
Core Analysis
- Script parsing
- Security scan
- Quality analysis
- Optimizations
Research & External
- Internet search (NEW)
- MS Docs fetching (NEW)
- GitHub vulnerability DB
- PowerShell Gallery API
MCP Integrations:
- Web Search MCP Server (Brave/Google)
- GitHub MCP Server (vulnerability database)
- Filesystem MCP Server (script analysis)
- Database MCP Server (historical analysis)
Checkpointing:
PostgresSaver → stores workflow state for recovery

Implementation Plan

Phase 1: Core Integration (Days 1-2)

Priority: Critical - Foundation for all other features

1.1 Backend API Enhancement

- [ ] Create new endpoint `/api/scripts/:id/analyze-langgraph`
- Call `AI_SERVICE_URL/langgraph/analyze`
- Pass script content, thread_id, require_human_review flag
- Return workflow_id, status, analysis_results
- [ ] Create streaming endpoint `/api/scripts/:id/analysis-stream`
- Use Server-Sent Events (SSE) for real-time updates
- Stream tool execution events
- Stream agent reasoning steps
- [ ] Create feedback endpoint `/api/scripts/:id/provide-feedback`
- Accept thread_id and feedback text
- Forward to `/langgraph/feedback`
- Return updated analysis state

Files to Modify: - `src/backend/src/controllers/ScriptController.ts` - `src/backend/src/routes/scripts.ts`

1.2 Frontend Service Layer

- [ ] Create `langgraphService.ts` in `src/frontend/src/services/`
- `analyzeLangGraph(scriptId, options)` - Start analysis
- `streamAnalysis(scriptId, onEvent)` - Subscribe to SSE
- `provideFeedback(scriptId, threadId, feedback)` - Send feedback
- `getAnalysisStatus(threadId)` - Poll status
- `resumeAnalysis(threadId)` - Continue from checkpoint

New File: - `src/frontend/src/services/langgraphService.ts`

1.3 Update ScriptAnalysis.tsx

- [ ] Add LangGraph analysis trigger button
- [ ] Show "Analyzing with AI Agents..." state
- [ ] Display basic streaming progress
- [ ] Maintain backward compatibility with old analysis

Testing: - [ ] Verify frontend can call new endpoint - [ ] Confirm LangGraph workflow executes - [ ] Check analysis results are returned correctly

---

Phase 2: Streaming & Real-time Updates (Days 3-4)

Priority: High - Critical for UX

2.1 Enhanced UI Components

AnalysisProgressPanel.tsx (NEW)

```
interface AnalysisProgressPanelProps {
  workflowId: string;
  onComplete: (results) => void;
}

// Features:
// - Real-time progress bar
// - Current stage indicator (analyze, tools, synthesis)
// - Tool execution timeline
// - Agent reasoning display
// - Error messages if failures occur
```

ToolExecutionLog.tsx (NEW)

```
interface ToolExecutionLogProps {
  tools: ToolExecution[];
  expandable: boolean;
}

// Shows:
// - Tool name (e.g., "security_scan")
// - Start/end time
// - Status (running, completed, failed)
// - Results preview
// - Expand for full JSON output
```

2.2 Event Stream Integration

- [ ] Create `useAnalysisStream` custom hook
- [ ] Handle SSE connection lifecycle
- [ ] Parse and categorize events (tool, reasoning, finding, error)
- [ ] Update UI state reactively as events arrive
- [ ] Auto-reconnect on connection drop

New Files: - `src/frontend/src/components/Analysis/AnalysisProgressPanel.tsx` - `src/frontend/src/components/Analysis/ToolExecutionLog.tsx` - `src/frontend/src/hooks/useAnalysisStream.ts`

2.3 Visual Enhancements

- [ ] Add animated progress indicators
- [ ] Tool icons (security shield, quality checkmark, optimization lightbulb)
- [ ] Color-coded severity indicators
- [ ] Loading skeleton for streaming content

Testing: - [ ] Verify events arrive in real-time - [ ] Test UI updates during 30+ second analysis - [ ] Confirm reconnection after network interruption - [ ] Test with multiple concurrent analyses

---

Phase 3: Human-in-the-Loop Workflows (Days 5-6)

Priority: High - Differentiating feature

3.1 Review UI Components

HumanReviewPanel.tsx (NEW)

```
interface HumanReviewPanelProps {
  workflowId: string;
  threadId: string;
  analysisResults: AnalysisResults;
  onApprove: (feedback?) => void;
  onReject: (feedback) => void;
  onRequestChanges: (feedback) => void;
}

// Features:
// - Display all findings for review
// - Highlight critical security issues
// - Accept/Reject/Request Changes buttons
// - Feedback textarea for specific guidance
// - Show AI's reasoning for transparency
```

3.2 Workflow Pause/Resume

- [ ] Detect when workflow enters human_review state
- [ ] Pause UI with clear "Review Required" indicator
- [ ] Save thread_id for session persistence
- [ ] Provide feedback and resume workflow
- [ ] Show updated results after review

3.3 Review Triggers

- [ ] Auto-trigger review for risk_score > 20
- [ ] Manual review checkbox before analysis
- [ ] Configurable review policies in settings

New Files: - `src/frontend/src/components/Analysis/HumanReviewPanel.tsx` - `src/frontend/src/components/Analysis/ReviewDecisionButtons.tsx`

Testing: - [ ] Test auto-pause on high-risk scripts - [ ] Verify feedback is incorporated correctly - [ ] Test workflow resumption after browser refresh - [ ] Confirm state persistence with thread_id

---

Phase 4: MCP Integration (Days 7-9)

Priority: Medium-High - Extends capabilities significantly

4.1 Backend MCP Setup

- [ ] Install MCP Python SDK: `pip install anthropic-mcp`
- [ ] Configure MCP servers in `src/ai/mcp_config.json`
- [ ] Initialize MCP client in orchestrator
- [ ] Create MCP tool wrappers for LangGraph

MCP Servers to Configure: 1. **Web Search** - Brave or Google search API `json { "name": "web-search", "command": "npx", "args": ["-y", "@modelcontextprotocol/server-brave-search"], "env": {"BRAVE_API_KEY": "..."} }`

1. **GitHub** - Access vulnerability database `json { "name": "github", "command": "npx", "args": ["-y", "@modelcontextprotocol/server-github"], "env": {"GITHUB_TOKEN": "..."} }`
2. **Filesystem** - Safe script reading `json { "name": "filesystem", "command": "npx", "args": ["-y", "@modelcontextprotocol/server-filesystem", "/allowed/paths"] }`

4.2 New Analysis Tools

internet_research_tool (NEW)

```
@tool
def internet_research(query: str, focus: str) -> str:
    """
    Search the internet for PowerShell security vulnerabilities,
    best practices, and current recommendations.

    Args:
        query: Search query (e.g., "Invoke-Expression security ris
        focus: Area of focus (security, performance, best_practice

    Returns:
        JSON with search results, relevant links, and key findings
    """
    # Use MCP web search server
    # Parse and summarize results
    # Return actionable insights
```

fetch_ms_docs_tool (NEW)

```
@tool
def fetch_powershell_docs(cmdlet: str) -> str:
    """
    Fetch official Microsoft PowerShell documentation for a cmdlet

    Args:
        cmdlet: PowerShell cmdlet name (e.g., "Get-Process")

    Returns:
        Documentation summary including parameters, examples, and
    """
    # Fetch from learn.microsoft.com
    # Parse and extract key information
    # Return structured documentation
```

vulnerability_check_tool (NEW)

```
@tool
def check_vulnerabilities(script_patterns: List[str]) -> str:
    """
    Check identified patterns against known vulnerability database

    Args:
        script_patterns: List of suspicious patterns found in scri

    Returns:
        CVE information, severity ratings, and mitigation strategi
    """
    # Query GitHub security advisories via MCP
    # Cross-reference with NVD database
    # Return vulnerability details
```

4.3 Integration Testing

- [ ] Test each MCP server independently
- [ ] Verify tool calls work through LangGraph
- [ ] Measure performance impact (latency)
- [ ] Test rate limiting and error handling

Testing: - [ ] Analyze script that uses `Invoke-Expression` - Verify web search finds security articles - Confirm vulnerability check flags CVEs - [ ] Test MS Docs fetching for standard cmdlets - [ ] Verify graceful degradation if MCP unavailable

---

Phase 5: Advanced Visualization (Days 10-11)

Priority: Medium - Enhances trust and transparency

5.1 Agent Orchestration Timeline

AgentActivityTimeline.tsx (NEW)

```
interface AgentActivityTimelineProps {
  events: AgentEvent[];
  highlightActive: boolean;
}

// Visualizes:
// - Sequential flow of agent actions
// - Parallel tool executions
// - Decision points (routing)
// - Human review interruptions
// - Time elapsed per stage
```

5.2 Reasoning Chain Display

- [ ] Show LLM reasoning for each decision
- [ ] Display tool selection rationale
- [ ] Highlight key findings as discovered
- [ ] Expandable reasoning details

ReasoningChainView.tsx (NEW)

```
// Shows:
// - "Analyzing script structure..."
// - "Detected potential security risk in line 45"
// - "Running security_scan tool to verify..."
// - "Tool result: CRITICAL - Code injection risk"
// - "Generating mitigation recommendations..."
```

5.3 Interactive State Graph

- [ ] Visual representation of StateGraph workflow
- [ ] Highlight current node
- [ ] Show completed/pending nodes

- [ ] Click nodes to see details

New Files: - `src/frontend/src/components/Analysis/AgentActivityTimeline.tsx` - `src/frontend/src/components/Analysis/ReasoningChainView.tsx` - `src/frontend/src/components/Analysis/WorkflowStateGraph.tsx`

Testing: - [ ] Test with complex multi-tool analysis - [ ] Verify timeline updates in real-time - [ ] Test reasoning chain with security findings - [ ] Confirm state graph reflects actual workflow

---

Phase 6: Enhanced Analysis Capabilities (Days 12-14)

Priority: Medium - Deepens analysis quality

6.1 Advanced Security Analysis

Deep Code Path Analysis

```
@tool
def analyze_code_paths(script_content: str) -> str:
    """
    Analyze all possible execution paths in the script.
    Identifies:
    - Conditional branches
    - Loop structures
    - Error handling paths
    - Dead code
    - Unreachable statements
    """
```

Dependency Vulnerability Scan

```
@tool
def scan_dependencies(script_content: str) -> str:
    """
    Extract and analyze script dependencies.
    Checks:
    - Module import statements
    - Version requirements
    - Known vulnerabilities in versions
    - Deprecated modules
    """
```

6.2 Performance Profiling

```
@tool
def profile_performance(script_content: str) -> str:
    """
    Estimate performance characteristics.
    Analyzes:
    - Loop complexity (O-notation)
    - Pipeline efficiency
    - Memory allocation patterns
    - Network I/O operations
    - Potential bottlenecks
    """
```

6.3 Compliance Checking

```
@tool
def check_compliance(script_content: str, standard: str) -> str:
    """
    Verify compliance with coding standards.
    Supports:
    - PSScriptAnalyzer rules
    - Custom organizational policies
    - Industry best practices
    - Security baseline requirements
    """
```

Files to Update: - `src/ai/agents/langgraph_production.py` - Add new tools - `src/ai/tools/` (NEW) - Separate tool implementations

Testing: - [ ] Test code path analysis with complex conditionals - [ ] Verify dependency scan detects outdated modules - [ ] Confirm performance profiling identifies inefficiencies - [ ] Test compliance checking against PSScriptAnalyzer

---

Phase 7: State Persistence & Recovery (Days 15-16)

Priority: Medium - Improves reliability

7.1 Frontend State Management

- [ ] Store thread_id in localStorage
- [ ] Save analysis progress in React state
- [ ] Persist across browser refreshes
- [ ] Clear on successful completion

useAnalysisState Hook (NEW)

```
interface AnalysisState {
  threadId: string | null;
  workflowId: string | null;
  status: 'idle' | 'running' | 'paused' | 'completed' | 'failed';
  currentStage: string;
  results: Partial<AnalysisResults>;
  canResume: boolean;
}

export function useAnalysisState(scriptId: string): {
  state: AnalysisState;
  resume: () => Promise<void>;
  clear: () => void;
}
```

7.2 Recovery UI

- [ ] Show "Resume Analysis" button if interrupted
- [ ] Display last known stage
- [ ] Warn if checkpoint is stale (>24h)
- [ ] Confirm before resuming

AnalysisRecoveryBanner.tsx (NEW)

```
// Shows when thread_id exists but analysis incomplete:
// "You have an interrupted analysis. Resume from [Stage Name]?"
// [Resume] [Start Fresh]
```

7.3 Backend Checkpoint Management

- [ ] Enable PostgreSQL checkpointing in production
- [ ] Implement checkpoint expiration (7 days)
- [ ] Add checkpoint cleanup background job

New Files: - `src/frontend/src/hooks/useAnalysisState.ts` - `src/frontend/src/components/Analysis/AnalysisRecoveryBanner.tsx`

Testing: - [ ] Refresh browser mid-analysis, verify resume works - [ ] Test resume after 1 hour delay - [ ] Verify expired checkpoints are handled gracefully - [ ] Test with PostgreSQL checkpointing enabled

---

Phase 8: Multi-Agent Coordination (Days 17-18)

Priority: Low-Medium - Advanced feature

8.1 Specialized Agent Roles

Create dedicated agents for different analysis aspects:

Security Specialist Agent

- Focus: Vulnerability detection, threat analysis
- Tools: Deep security scan, CVE lookup, exploit detection
- Prompts: Security-focused system prompts

Performance Expert Agent

- Focus: Optimization opportunities, efficiency analysis
- Tools: Performance profiling, benchmark comparison
- Prompts: Performance-tuned analysis

Best Practices Auditor Agent

- Focus: Code quality, maintainability, standards
- Tools: Style checking, documentation analysis
- Prompts: Best practices verification

8.2 Agent Coordination Layer

- [ ] Route analysis to appropriate specialist agents
- [ ] Aggregate results from multiple agents
- [ ] Resolve conflicting recommendations
- [ ] Synthesize comprehensive report

Multi-Agent Orchestrator Enhancement

```
 # Enhance StateGraph with agent routing
def route_to_specialists(state: PowerShellAnalysisState) -> List[s
    """Determine which specialist agents to invoke."""
    agents_needed = []

    if has_security_concerns(state):
        agents_needed.append("security_specialist")
    if has_performance_issues(state):
        agents_needed.append("performance_expert")
    if needs_best_practices_review(state):
        agents_needed.append("best_practices_auditor")

    return agents_needed
```

8.3 Collaboration Visualization

- [ ] Show which agents are active
- [ ] Display agent-specific findings
- [ ] Highlight consensus recommendations
- [ ] Flag disagreements for human review

Files to Create: - `src/ai/agents/specialists/security_specialist.py` - `src/ai/agents/specialists/performance_expert.py` - `src/ai/agents/specialists/best_practices_auditor.py` - `src/ai/agents/multi_agent_coordinator.py`

Testing: - [ ] Test routing logic for different script types - [ ] Verify each specialist agent produces unique insights - [ ] Test aggregation and conflict resolution - [ ] Confirm UI shows multi-agent activity correctly

---

Phase 9: Testing & Quality Assurance (Days 19-20)

Priority: Critical - Must validate all changes

9.1 Unit Tests

- [ ] Backend API endpoints (ScriptController)
- [ ] LangGraph tools (security_scan, quality_analysis, etc.)
- [ ] MCP integrations (web search, docs fetching)
- [ ] Frontend service layer (langgraphService.ts)
- [ ] React components (AnalysisProgressPanel, etc.)

9.2 Integration Tests

- [ ] End-to-end analysis workflow
- [ ] Streaming event delivery
- [ ] Human-in-the-loop pause/resume
- [ ] State persistence and recovery
- [ ] MCP tool execution

9.3 Performance Tests

- [ ] Analysis time for scripts of varying sizes (10, 100, 1000 lines)
- [ ] Concurrent analysis handling (5, 10, 20 users)
- [ ] Streaming event throughput
- [ ] Database checkpoint overhead
- [ ] MCP tool latency

9.4 User Acceptance Testing

- [ ] Internal team testing with real scripts
- [ ] Beta users for feedback
- [ ] Security team review of vulnerability detection
- [ ] Performance team review of optimization suggestions

Testing Scenarios: 1. **Simple Script** (10 lines, no issues) - Expected: Fast analysis (<5s), green scores, no recommendations

1. **Complex Script** (500 lines, multiple functions)

2. Expected: Deep analysis (30-60s), multiple tool calls, detailed findings

3. **Malicious Script** (Invoke-Expression, downloadstring)

4. Expected: CRITICAL risk score, human review triggered, clear warnings
5. **Interrupted Analysis**
6. Expected: Resume from checkpoint works, no data loss

Phase 10: Documentation & Deployment (Days 21-22)

Priority: High - Enable adoption

10.1 User Documentation

- [ ] **User Guide**: How to use enhanced Script Analysis
- [ ] **Video Tutorial**: Walkthrough of new features
- [ ] **FAQ**: Common questions about analysis results
- [ ] **Troubleshooting**: What to do if analysis fails

10.2 Developer Documentation

- [ ] **Architecture Diagram**: Updated with LangGraph integration
- [ ] **API Documentation**: New endpoints and parameters
- [ ] **MCP Setup Guide**: How to configure MCP servers
- [ ] **Contributing Guide**: How to add new tools/agents

10.3 Deployment Checklist

- [ ] Environment variables configured
- `OPENAI_API_KEY` for GPT-5.2-codex
- `BRAVE_API_KEY` for web search
- `GITHUB_TOKEN` for vulnerability DB
- `DATABASE_URL` for checkpointing
- [ ] Database migrations applied
- [ ] Redis cache cleared
- [ ] Docker images rebuilt
- [ ] Health checks passing
- [ ] Monitoring/alerting configured

10.4 Rollout Strategy

Phase 1: Internal Testing (Day 21) - Deploy to staging environment - Team testing with real scripts - Fix critical bugs

Phase 2: Beta Release (Day 22 morning) - Enable for 10% of users - Monitor performance metrics - Collect feedback

Phase 3: Full Release (Day 22 afternoon) - Enable for all users - Announce new features - Monitor for issues

---

Success Metrics

Performance

- [ ] Analysis completion time < 60 seconds for 95% of scripts
- [ ] Streaming latency < 500ms per event
- [ ] Frontend responsiveness maintained during analysis
- [ ] No memory leaks during long-running analyses

Quality

- [ ] Security vulnerability detection rate > 95%
- [ ] False positive rate < 5%
- [ ] User satisfaction score > 4.0/5.0
- [ ] Recommendation acceptance rate > 60%

Reliability

- [ ] Analysis success rate > 99%
- [ ] Recovery from checkpoint success rate > 95%
- [ ] Uptime > 99.5%
- [ ] Error rate < 0.5%

Adoption

- [ ] 80% of script uploads trigger analysis within first week
- [ ] Human review feature used for 30%+ of critical scripts
- [ ] Average analysis per user increases by 50%

Risk Mitigation

Technical Risks

Risk: LangGraph analysis timeout

- **Mitigation**: Implement 60s timeout, fallback to basic analysis
- **Monitoring**: Track timeout rate, alert if >5%

Risk: MCP server unavailability

- **Mitigation**: Graceful degradation, continue without external tools
- **Monitoring**: Health checks every 60s, log failures

Risk: Frontend performance degradation

- **Mitigation**: Virtual scrolling for long logs, debounce updates
- **Monitoring**: Measure frame rate, optimize if <30 FPS

Risk: Database checkpoint storage growth

- **Mitigation**: 7-day expiration policy, archival to S3
- **Monitoring**: Alert if DB size > 10GB

User Experience Risks

Risk: Analysis too slow, users abandon

- **Mitigation**: Show immediate progress, estimate time remaining
- **Solution**: "Quick scan" option for preliminary results

Risk: Too much information, overwhelming

- **Mitigation**: Collapsible sections, progressive disclosure
- **Solution**: "Summary" view by default, "Details" on demand

Risk: False positives erode trust

- **Mitigation**: Explain reasoning, allow user feedback
- **Solution**: "Report incorrect finding" button

Security Risks

Risk: API key exposure in frontend

- **Mitigation**: All AI calls go through backend proxy
- **Validation**: Code review, security audit

Risk: Malicious scripts exploit analysis

- **Mitigation**: Sandboxed execution, input validation
- **Validation**: Penetration testing

Resource Requirements

Team

- **Frontend Developer** (Full-time, Days 1-20): React/TypeScript, SSE
- **Backend Developer** (Full-time, Days 1-20): Node.js/Express, API design
- **AI/ML Engineer** (Full-time, Days 1-20): LangGraph, OpenAI API, MCP
- **QA Engineer** (Full-time, Days 15-22): Testing, automation
- **DevOps Engineer** (Part-time, Days 20-22): Deployment, monitoring

Infrastructure

- **Compute**: Increase AI service CPU by 2x (for concurrent analyses)
- **Database**: Add 50GB storage for checkpoints
- **API Costs**: Estimate $500/month additional for GPT-5.2-codex + web search

External Services

- **Brave Search API**: $5/month (free tier may suffice initially)
- **GitHub API**: Free for personal/organization use
- **OpenAI API**: Pay-as-you-go (GPT-5.2-codex pricing TBD)

Next Steps - Immediate Actions

Today (January 9, 2026)

1. **Review this plan** with team for approval
2. **Set up project tracking** (GitHub Project, Jira, etc.)
3. **Assign Phase 1 tasks** to team members
4. **Configure development environment** (API keys, MCP servers)
5. **Create feature branch** `feature/enhanced-script-analysis`

Tomorrow (January 10, 2026)

1. **Start Phase 1** - Core integration
2. **Create backend API endpoints**
3. **Build frontend service layer**
4. **Test basic LangGraph connectivity**

End of Week 1

- **Complete Phases 1-3**
- **Demo streaming analysis and human-in-the-loop**
- **Gather team feedback, adjust plan if needed**

Research Sources

Agent Orchestration & LangGraph

- LangGraph Documentation
- State of Agent Engineering 2026
- Workflows and Agents Guide
- Top AI Agents in 2026
- Agent Orchestration Frameworks Comparison
- Multi-Agent AI Frameworks
- AI Agent Frameworks Guide
- Building Agentic Workflows with LangGraph
- Agentic AI Architecture Design
- Building Effective Agentic Systems

Model Context Protocol (MCP)

- Introducing Model Context Protocol
- MCP Simplified Guide
- Introduction to Model Context Protocol
- What is MCP and How It Works
- How to Use MCP with Claude
- MCP Servers Repository
- Model Context Protocol Official Site

PowerShell Security & Best Practices

- AI Coding Agents for PowerShell
- AI for PowerShell Script Standardization
- PowerShell AIShell
- AI Security Pitfalls in 2026
- PowerShell Script Execution Policies

Function Calling & Agentic Workflows

- Function Calling in Agentic Workflows

- Building Agentic AI with LangChain
- Agentic Workflows Overview
- Top Agentic AI Tools for 2026

Appendix

A. Technology Stack

Frontend - React 18+ with TypeScript - TanStack Query v5 for data fetching - Server-Sent Events (SSE) for streaming - Tailwind CSS for styling

Backend - Node.js with Express/TypeScript - Sequelize ORM for PostgreSQL - Redis for caching - SSE middleware for event streaming

AI Service - Python 3.10+ with FastAPI - LangGraph 1.0.5 for orchestration - LangChain for tool integration - OpenAI API (GPT-5.2-codex) - Anthropic MCP SDK

Infrastructure - Docker Compose for local development - PostgreSQL 15 with pgvector - Redis 7.0 - Nginx (production proxy)

B. Key Files Reference

Backend - `src/backend/src/controllers/ScriptController.ts` - API endpoints - `src/backend/src/routes/scripts.ts` - Route definitions - `src/backend/src/middleware/sse.ts` (NEW) - SSE support

Frontend - `src/frontend/src/pages/ScriptAnalysis.tsx` - Main analysis UI - `src/frontend/src/services/langgraphService.ts` (NEW) - API client - `src/frontend/src/hooks/useAnalysisStream.ts` (NEW) - Streaming hook - `src/frontend/src/components/Analysis/` (NEW) - Analysis components

AI Service - `src/ai/agents/langgraph_production.py` - LangGraph orchestrator - `src/ai/langgraph_endpoints.py` - FastAPI endpoints - `src/ai/mcp_config.json` (NEW) - MCP server configuration - `src/ai/tools/` (NEW) - Custom tool implementations

C. Environment Variables

```
 # Backend
AI_SERVICE_URL=http://ai-service:8000
DATABASE_URL=postgresql://user:pass@postgres:5432/psscript
REDIS_URL=redis://redis:6379

# AI Service
OPENAI_API_KEY=sk-proj-...
DEFAULT_MODEL=gpt-5.2-codex
USE_POSTGRES_CHECKPOINTING=true

# MCP Integrations (NEW)
BRAVE_API_KEY=...
GITHUB_TOKEN=...
ENABLE_WEB_SEARCH=true
ENABLE_VULNERABILITY_CHECK=true
```

D. Rollback Plan

If critical issues arise post-deployment:

1. **Immediate Rollback** (< 5 minutes)
2. Revert frontend to previous version via Git
3. Disable `/langgraph/analyze` endpoint via feature flag
4. Fall back to legacy `/chat` endpoint
5. **Data Preservation**
6. Keep PostgreSQL checkpoints for 7 days
7. Export analysis results before rollback
8. Preserve thread_id mappings
9. **Communication**
10. Notify users via in-app banner
11. Post incident report within 24 hours
12. Provide timeline for re-enabling feature

Conclusion

This comprehensive plan transforms Script Analysis from a basic chatbot into a **production-grade, multi-agent, agentic AI system** that leverages the latest 2026 technologies:

- ✅ **LangGraph 1.0** for robust orchestration
- ✅ **Model Context Protocol** for external tool access
- ✅ **Human-in-the-loop** for collaborative analysis
- ✅ **Real-time streaming** for transparency
- ✅ **State persistence** for reliability
- ✅ **Internet research** for up-to-date findings
- ✅ **Multi-agent coordination** for depth

Estimated Timeline: 22 days (4.4 weeks) **Team Size:** 4-5 engineers **Risk Level:** Medium (well-researched, proven technologies) **Impact:** High (significant UX improvement, competitive differentiation)

Status: ✅ Ready for team review and approval

Document prepared by: Claude (claude-sonnet-4-5) **Date**: January 9, 2026 **Version**: 1.0 **Next Review**: After Phase 3 completion (Day 6)

Generated 2026-01-13 06:26 UTC