Neural Archaeologist

Multi-Agent AI System for Code History Excavation

Complete Project Documentation

Project Type:	Multi-Agent AI Application
Tech Stack:	Python, FastAPI, React, LangGraph, Groq
Complexity:	Medium-High (Hackathon Ready)
Time Estimate:	48 Hours
Target Users:	Engineering Teams, CTOs, M&A Teams

Table of Contents

1. Executive Summary

Neural Archaeologist is a multi-agent AI system that excavates abandoned GitHub repositories and reconstructs their hidden history. Unlike traditional code analysis tools that explain what code does, Neural Archaeologist uncovers WHY it was built, WHAT happened to it, and WHETHER it's still valuable.

The Elevator Pitch:

"When a senior engineer leaves your company, they take years of knowledge with them. GitHub Copilot can tell you what a function does, but it can't tell you why it was built, what happened to it, or if it's still valuable. That's a $50 billion problem. Neural Archaeologist solves it with autonomous AI agents that investigate code like digital forensics."

847 commits analyzed	**4 contributors tracked**	**90% confidence achieved**
3-minute investigation	**Real-time updates**	**Multi-source intelligence**

2. Project Concept

What is Digital Archaeology?

Just like real archaeologists excavate ancient sites to understand lost civilizations, Neural Archaeologist excavates dead codebases to understand lost engineering decisions. It combines git history analysis, web research, and AI reasoning to piece together the complete story of a software project.

The Innovation:

- Multi-agent coordination - Agents collaborate like a real investigation team
- Multi-source intelligence - Combines git + GitHub + web + news articles
- Proactive investigation - Doesn't wait for questions, investigates autonomously
- Confidence-driven iteration - Keeps searching until it's sure
- Narrative storytelling - Transforms data into compelling human stories

3. The Problem & Solution

The Problem:

Companies have thousands of dead repositories with zero documentation. When developers inherit legacy code, they spend weeks trying to understand: What was this built for? Why was it abandoned? Is anything salvageable? Current tools like GitHub Copilot can explain what code does, but they can't answer these deeper questions.

The Market Impact:

- Technical debt costs companies $3.61 per line of code
- $50+ billion spent annually on legacy code maintenance
- Average developer spends 2-3 weeks understanding inherited codebases
- M&A; due diligence requires expensive technical audits

Our Solution:

Neural Archaeologist uses a team of specialized AI agents that work together to investigate repositories. The Scout gathers data from git history and the web. The Analyst identifies patterns and forms hypotheses. The Coordinator ensures quality by requesting more evidence when confidence is low. The Narrator transforms findings into actionable insights.

What Makes It Different:

Feature	GitHub Copilot	Neural Archaeologist
Explains current code	✓	✓
Analyzes git history	✗	✓
Searches external context	✗	✓
Forms hypotheses	✗	✓
Multi-agent coordination	✗	✓
Generates narratives	✗	✓

4. Multi-Agent Architecture

The system uses 4 specialized agents orchestrated by LangGraph. This is not a simple pipeline - agents communicate, iterate, and make dynamic decisions based on confidence levels.

System Overview:

```
Coordinator (Meta-Agent) → Makes strategic decisions, routes between agents

■■ Scout Agent → Gathers raw data from git and web

■■ Analyst Agent → Finds patterns, forms hypotheses

■■ Narrator Agent → Generates final story and report
```

Why Multi-Agent?

- Separation of concerns - Each agent has one job and does it well
- Iterative refinement - Agents can loop back for more evidence
- Quality assurance - Low confidence triggers re-investigation
- Emergent intelligence - Collaboration creates better results
- Impressive demo - Judges see agents working together in real-time

5. Agent Roles & Responsibilities

■ Scout Agent (Information Gatherer)

Job: Collect raw data from multiple sources

Tools:

- GitPython - Parses commit history, authors, dates, messages
- GitHub API - Fetches issues, PRs, repo metadata
- Brave Search API - Searches web for news, blogs, articles
- Web Scraping - Extracts content from relevant pages

Example Output:

```
{"total_commits": 847, "spike_detected": "Nov 2017 (73 commits)",
"last_activity": "March 2018", "news_found": ["Uber blog article"]}
```

■ Analyst Agent (Pattern Detector)

Job: Identify meaningful patterns and form hypotheses

Capabilities:

- Detects commit patterns (spikes, gaps, decay)
- Identifies anomalies (sudden stops, emergency fixes)
- Correlates multiple data sources
- Assigns confidence scores (0-100%)
- Determines if more evidence is needed

Example Output:

```
{"hypothesis": "Project abandoned when better tools emerged", "confidence":
0.85, "reasoning": [...], "needs_verification": false}
```

■ Narrator Agent (Storyteller)

Job: Transform data into compelling narrative

Outputs:

- Three-act story structure (rise, crisis, fall)
- Interactive timeline with key events
- Contributor profiles and impact

- Salvageability analysis with recommendations
- Citations to external sources

■ Coordinator (Orchestrator)

Job: Make strategic decisions about investigation flow

Decision Logic:

- IF confidence < 70% → Request more evidence from Scout
- IF findings conflict → Send Analyst to re-analyze
- IF confidence > 85% → Proceed to Narrator for report
- ELSE → Continue iteration

6. Investigation Flow

The investigation follows an iterative, confidence-driven process. Agents don't just run once - they collaborate until they're confident in their findings.

Round 1: Initial Discovery

1. User submits repository URL
2. Scout clones repo and analyzes git history
3. Scout reports: '847 commits, spike in Nov 2017'
4. Coordinator: 'Interesting, send to Analyst'

Round 2: Pattern Analysis

5. Analyst examines patterns in commit data
6. Analyst: 'Hypothesis: Abandoned project. Confidence: 60%'
7. Coordinator: 'Too low, need external proof'

Round 3: Verification

8. Coordinator: 'Scout, search web for context'
9. Scout searches: 'Uber Pyflame 2018'
10. Scout: 'Found blog: Why we moved to py-spy'
11. Coordinator: 'This confirms hypothesis!'

Round 4: Final Analysis

12. Coordinator: 'Analyst, re-analyze with new data'
13. Analyst: 'Confidence now 90%, ready for report'
14. Coordinator: 'Narrator, generate final story'

Round 5: Story Generation

15. Narrator synthesizes all findings
16. Narrator creates timeline + narrative + analysis
17. System presents beautiful report to user

Key Point:

The Coordinator makes dynamic decisions. If confidence is low, it requests more data. If findings conflict, it asks for re-analysis. This isn't a hardcoded pipeline - it's adaptive intelligence.

7. Technical Stack

Backend Technologies:

Component	Technology	Purpose
Web Framework	FastAPI	REST API endpoints, WebSocket support
Multi-Agent	LangGraph	Agent orchestration and state management
LLM Provider	Groq API (Llama 3.3 70B)	Agent intelligence (fast, free)
Git Analysis	GitPython	Parse commit history and metadata
Web Search	Brave Search API	Find external context and news
Database	PostgreSQL	Store investigations and users
Cache	Redis	Session storage and job queue
Background Jobs	Celery	Async investigation processing
Auth	JWT (python-jose)	User authentication

Frontend Technologies:

Component	Technology	Purpose
Framework	React 18 / Next.js 14	UI framework with SSR support
Styling	TailwindCSS	Utility-first CSS framework
Components	shadcn/ui	Pre-built accessible components
State Management	Zustand / React Context	Lightweight state management
API Calls	TanStack Query	Server state management & caching
Real-time	Socket.io-client	WebSocket for live updates
Visualizations	Recharts	Timeline and chart components
Animations	Framer Motion	Smooth transitions and effects

Why Groq? (Critical Decision)

- FREE - 14,400 requests/day on free tier
- FAST - 500+ tokens/second (10x faster than alternatives)
- SMART - Llama 3.3 70B is highly capable
- RELIABLE - 99.9% uptime, stable API
- DEMO-FRIENDLY - Fast responses make demos feel magical

8. Full-Stack Architecture

The application follows a modern full-stack architecture with clear separation of concerns.

System Layers:

1. Frontend Layer (React/Next.js)

- User interface and real-time updates
- Communicates via REST API and WebSocket
- Handles user authentication and state

2. API Layer (FastAPI)

- REST endpoints for CRUD operations
- WebSocket endpoint for real-time agent updates
- JWT authentication middleware
- Request validation and error handling

3. Agent System Layer (LangGraph)

- Multi-agent orchestration logic
- State management between agents
- Confidence-driven routing decisions
- Progress callbacks to WebSocket

4. Background Jobs Layer (Celery)

- Asynchronous investigation processing
- Job queue management with Redis
- Long-running tasks don't block API

5. Data Layer (PostgreSQL + Redis)

- PostgreSQL: Users, investigations, reports
- Redis: Session cache, job queue, temporary data

Data Flow Example:

User Action:

```
1. User pastes repo URL in frontend → Clicks 'Analyze'

2. Frontend → POST /api/investigations
```

3. Backend validates URL → Creates DB record → Queues Celery job

4. Returns investigation_id to frontend

5. Frontend subscribes to WebSocket: ws://api/investigations/{id}

6. Celery worker picks up job → Runs agent system

7. Agents emit progress via WebSocket:

• 'Scout: Analyzing 847 commits...'

• 'Analyst: Confidence 60%'

• 'Scout: Searching web...'

• 'Analyst: Confidence 90%'

8. Final report saved to database

9. Frontend receives complete report and displays it

9. User Experience Journey

Landing Page (First Impression):

- Hero section with clear value proposition
- Big search bar for repo URL
- One-click example repos (Uber Pyflame, Docker Fig, Parse)
- Social proof and statistics
- Call-to-action: 'Start Archaeological Dig'

Investigation Dashboard (Main Experience):

Split-screen design:

Left Panel - Investigation Log:

- Live scrolling feed of agent messages
- Different icons for each agent (■ Scout, ■ Analyst, ■ Narrator)
- Timestamps and color coding
- Auto-scrolls to latest message

Right Panel - Live Metrics:

- Progress bars (commits analyzed, sources found)
- Real-time confidence score with color indication
- Current hypothesis display
- Agent status indicators (active/waiting/idle)

Report View (Final Results):

- Executive summary with key metrics
- Interactive timeline (zoomable, clickable)
- Three-act narrative story
- Contributor profiles with impact analysis
- Salvageability analysis with recommendations
- External source citations with links
- Export to PDF and share buttons

History Page:

- List of past investigations
- Search and filter functionality
- Quick stats per investigation
- Re-analyze and share options

10. Frontend Design Principles

Color Scheme:

Element	Color	Usage
Primary	Indigo (#6366F1)	CTAs, headings, brand
Secondary	Purple (#8B5CF6)	Accents, highlights
Success	Green (#10B981)	Completed states, high confidence
Warning	Amber (#F59E0B)	Medium confidence, caution
Error	Red (#EF4444)	Failures, low confidence
Background	Gray (#FAFBFC)	Page background
Surface	White (#FFFFFF)	Card backgrounds

Key UI/UX Patterns:

1. Real-Time Visual Feedback

- Messages fade in with smooth animations
- Confidence bars update with color transitions
- Progress indicators animate smoothly
- Subtle sound effects on key events

2. Status Indicators

- ■ Green pulse: Active/Processing
- ■ Yellow pulse: Waiting/Paused
- ■ Red solid: Error/Failed
- ■ Green check: Completed

3. Loading States

- Skeleton screens while loading
- Pulsing placeholders for content
- Progress spinners with percentage
- Educational tips during wait time

4. Micro-interactions

- Confidence score counts up smoothly
- Timeline points enlarge on hover
- Buttons have subtle press animations
- Cards lift on hover with shadow

Responsive Design:

- Desktop (1920px): Two-column layout, all features visible
- Tablet (768px): Stacked layout, collapsible sections
- Mobile (375px): Single column, bottom sheets, swipe gestures

11. Backend API Structure

API Endpoints:

Method	Endpoint	Purpose
POST	/api/auth/register	Create new user account
POST	/api/auth/login	Authenticate user, return JWT
POST	/api/investigations	Start new investigation
GET	/api/investigations/{id}	Get investigation status/results
GET	/api/investigations/user/{user_id}	List user investigations
DELETE	/api/investigations/{id}	Delete investigation
WebSocket	/ws/investigations/{id}	Real-time progress updates

WebSocket Events:

Client → Server:

- connect: Establish connection
- subscribe: Subscribe to investigation updates
- disconnect: Close connection

Server → Client:

- investigation_update: Agent progress message
- confidence_changed: Confidence score updated
- investigation_complete: Final report ready
- investigation_error: Error occurred

12. Database Schema

PostgreSQL Tables:

users

- id (UUID, primary key)
- email (VARCHAR, unique)
- password_hash (VARCHAR)
- created_at (TIMESTAMP)

investigations

- id (UUID, primary key)
- user_id (UUID, foreign key)
- repo_url (TEXT)
- status (VARCHAR: pending/processing/completed/failed)
- findings (JSONB - scout and analyst data)
- report (TEXT - final narrative)
- confidence (FLOAT)
- created_at (TIMESTAMP)
- completed_at (TIMESTAMP)

agent_logs

- id (SERIAL, primary key)
- investigation_id (UUID, foreign key)
- agent_name (VARCHAR: scout/analyst/narrator)
- message (TEXT)
- data (JSONB - additional context)
- timestamp (TIMESTAMP)

repo_cache

- repo_url (TEXT, primary key)
- git_data (JSONB - cached commit data)
- last_updated (TIMESTAMP)

Redis Data:

- investigation:{id}:status - Current investigation status
- investigation:{id}:progress - Progress percentage
- session:{token} - User session data
- celery:* - Celery job queue data

13. Demo Strategy (4-Minute Script)

Minute 1: The Hook (30 seconds)

"Legacy code costs companies billions. When a developer leaves, their knowledge vanishes. GitHub Copilot can explain what code does, but not WHY it exists or what HAPPENED to it. That's a $50 billion problem."

Minute 2: The Setup (30 seconds)

"Let me show you a real example - Uber's Pyflame, an abandoned profiling tool. I'll paste the URL and watch as multiple AI agents investigate..."

[Screen share: Paste URL, click 'Start Excavation']

Minute 3: The Magic (2 minutes)

[Show split-screen with live agent updates]

- Scout finds anomaly: 'Activity stopped in 2018'
- Analyst forms hypothesis: 'Project abandoned' (60% confidence)
- Coordinator: 'Confidence too low, need more evidence'
- Scout searches web, finds blog post
- Analyst updates: 'Confidence now 90%'
- Narrator generates story

[Point out key moment]:

"See that? The system DECIDED on its own that 60% wasn't good enough. It sent the Scout to find external proof. That's multi-agent coordination."

Minute 4: The Impact (1 minute)

[Show final report with timeline and narrative]

"In 3 minutes, we uncovered what would take a human days. Unlike Copilot, we didn't just explain the code - we told its story."

[Show export to PDF and history page]

"This solves real problems: M&A; due diligence, technical debt assessment, onboarding new engineers. The market is massive."

Key Demo Tips:

- Pre-test everything 5 times
- Have backup video if live demo fails
- Point out agent collaboration explicitly
- Show confidence score changing in real-time
- End with business value, not technical details

14. Implementation Timeline (48 Hours)

Hours	Phase	Tasks
0-2	Setup	APIs, environment, dependencies
2-8	Scout Agent	Git parsing, web search integration
8-12	Analyst Agent	Pattern detection with Groq
12-16	Narrator Agent	Story generation, timeline viz
16-24	Coordinator	LangGraph, dynamic routing logic
24-32	Frontend	React dashboard, WebSocket integration
32-40	Testing	Bug fixes, edge cases, demo repos
40-46	Polish	Animations, error handling, responsive
46-48	Prep	Demo practice, slides, contingency plans

Feature Priority:

Must Have (Core MVP):

- ■ Scout agent with git parsing
- ■ Analyst agent with pattern detection
- ■ Narrator agent with story generation
- ■ Basic coordinator with routing
- ■ Simple frontend (even Streamlit is fine)
- ■ Works on 1 demo repo

Should Have (Impressive):

- ■ Dynamic confidence-based routing
- ■ Web search integration
- ■ Real-time WebSocket updates
- ■ Timeline visualization
- ■ Professional React UI

Nice to Have (If Time):

- ■ User authentication
- ■ Multiple demo repos
- ■ Export to PDF
- ■ History page

■ Dark mode

15. Why This Wins Hackathons

Technical Complexity ■

- Multi-agent system with true coordination
- Dynamic routing (not hardcoded pipelines)
- State management across agents
- Real-time WebSocket communication
- Full-stack architecture

Real-World Value ■

- Solves $50B problem (technical debt)
- Clear target customers (engineering teams, CTOs, M&A;)
- Saves weeks of developer time
- Measurable ROI

Unique Approach ■

- Not another chatbot or CRUD app
- 'Digital archaeology' is memorable
- Different from GitHub Copilot
- Storytelling angle stands out

Demo Impact ■

- Live agent collaboration is engaging
- Visual split-screen shows complexity
- 'Aha moment' when agents iterate
- Beautiful final output
- Fast (3-minute investigation)

Recruiter Appeal ■

- Shows understanding of real problems
- Demonstrates: AI, APIs, git, full-stack
- Easy to explain in interviews
- Portfolio-worthy project

Common Judge Questions & Answers:

Q: How is this different from GitHub Copilot?

A: Copilot explains what code DOES. We explain WHY it was built and what HAPPENED to it. Copilot is reactive (answers questions), we're proactive (investigate autonomously).

Q: What if there's no external context?

A: We rely primarily on git history, which always exists. External context increases confidence but we can still generate timelines and identify patterns from commits alone.

Q: Why Groq instead of GPT-4?

A: Speed matters for demos. Groq is 10x faster and free. For this use case, Llama 3.3 70B is plenty smart, and the speed makes the demo feel magical.

16. Post-Hackathon Potential

If You Want to Continue Building:

Version 2.0 Features:

- Predictive decay detection (warn when active repos are dying)
- Auto-generate migration plans with effort estimates
- Compare multiple repos (find patterns across portfolio)
- Integrate with Jira, Confluence, Slack
- Advanced visualizations (interactive dependency graphs)
- Team collaboration features
- API for programmatic access

Business Model:

Pricing Tiers:

- Free: 3 investigations/month
- Starter: $99/month - 25 investigations
- Professional: $299/month - 100 investigations + team features
- Enterprise: Custom pricing - Unlimited + private deployment

Target Market:

- Engineering managers needing technical debt assessment
- CTOs overseeing large codebases
- M&A; teams doing technical due diligence
- DevRel teams documenting open source history
- Consultancies auditing client codebases

Market Size:

- TAM: Every company with 50+ repositories
- ~500,000 companies worldwide fit this profile
- If 1% convert at $99/month = $5M ARR potential
- Enterprise deals could be $10k-50k/year

Growth Strategy:

Phase 1 (Months 1-3): Product-Market Fit

- Free tier with limited features
- Target early adopters (CTO communities, Hacker News)
- Collect feedback and iterate

Phase 2 (Months 4-6): Revenue

- Launch paid tiers
- Content marketing (blog about code archaeology)
- Case studies from early customers

Phase 3 (Months 7-12): Scale

- Enterprise sales team
- Integrations with popular dev tools
- Series A fundraising ($2-5M)

Appendix: Quick Reference

Key Technologies at a Glance:

Category	Technology	Why
LLM	Groq (Llama 3.3 70B)	Fast, free, smart
Multi-Agent	LangGraph	State management, routing
Backend	FastAPI	Modern, async, WebSocket
Frontend	React + Next.js	Component-based, SSR
Database	PostgreSQL	Relational, reliable
Cache	Redis	Fast, job queue
Jobs	Celery	Background processing
Git	GitPython	Parse repositories
Search	Brave API	Web research
Real-time	Socket.io	Live updates

Success Metrics:

Demo Success:

- Judges say 'wow' during agent collaboration
- Someone asks 'can I use this?'
- Demo completes in under 4 minutes
- No crashes or errors

Technical Success:

- Agents actually collaborate (not fake)
- Confidence scores make sense
- Final report is coherent and useful
- System handles edge cases gracefully

Pre-Demo Checklist:

- ■ Tested on 3+ different repos
- ■ Backup demo video prepared
- ■ Internet connection tested
- ■ API keys verified and working
- ■ Error handling for API failures

■ 30-second elevator pitch memorized

■ Answers to common questions prepared

■ Demo rehearsed 5 times

■ Timing confirmed (under 4 minutes)

■ Architecture diagram ready

Final Note:

Remember: Judges care about WORKING DEMOS, REAL PROBLEMS, and CLEAR VALUE. Focus on making the demo smooth and impressive. The multi-agent aspect is your technical differentiator. The storytelling aspect is your UX differentiator. The business case is your market differentiator. You have all three. Now go build it and win! ■