MCP GitHub Repository Analyzer

A smart GitHub repository analyzer that uses AI to understand and explain your code. Ask questions in plain English and get intelligent answers about any GitHub repository.

What It Does

This tool fetches your GitHub repository, analyzes the code structure, and uses AI (Google Gemini) to answer questions about your codebase intelligently.

Example: Ask "What does my urls.py do?" and get a detailed explanation of your Django URL configuration instead of generic responses.

Quick Start

1. Install Dependencies

```
pip install aiohttp google-generativeai python-dotenv
```

2. Get API Key

- Go to Google AI Studio
- Create a new API key
- Save it for the next step

3. Setup Environment

Create a `.env` file:

```
GOOGLE_API_KEY=your_api_key_here
GITHUB_TOKEN=your_github_token_here  # Optional
```

4. Run the Analyzer

```
python client.py server.py <username> <repository>
```

Example:

```
python client.py server.py octocat Hello-World
```

🔍 Step-by-Step Walkthrough

Step 1: Repository Loading

```
$ python client.py server.py octocat Hello-World
Loading repository: octocat/Hello-World...
🔄 Fetching latest repo data...
```

What happens: The system fetches all files from GitHub and analyzes their structure.

Sample output:

```
🔄 Generated fresh summary with 8 files
📁 File types: .py(3), .md(2), .json(1), .txt(2)
✅ Repository 'Hello-World' loaded successfully!
```

Step 2: User Question

```
💬 I'm ready to help! Ask me anything about your code.

You: What does my main.py file do?
```

Step 3: Smart File Selection (LLM Call #1)

```
🤖 Step 1: Deciding relevant files with LLM...
✅ LLM selected files: ['main.py', 'utils.py']
```

What happens: AI analyzes your question and repository structure to find the most relevant files.

Sample LLM input:

```
{
  "question": "What does my main.py file do?",
  "repository_summary": {
    "total_files": 8,
    "file_summaries": [
      {"path": "main.py", "functions": ["main", "init"], "description": "Main entry point
      {"path": "utils.py", "functions": ["helper", "validate"], "description": "Helper fu
    ]
  }
}
```

Sample LLM output:

```
main.py,utils.py
```

Step 4: File Content Fetching

```
 Step 2: Fetching 2 files from GitHub...
    main.py: 245 chars, 240 non-whitespace
    utils.py: 156 chars, 150 non-whitespace
```

What happens: The system fetches the actual code content of the selected files.

Step 5: Intelligent Analysis (LLM Call #2)

```
 Step 3: Generating final answer with LLM...
```

Sample LLM input:

```
Repository Summary: {...}

File Content:
 main.py:
```

def main():
print("Hello World!")
validate_input()

if **name** == "**main**":
main()

```

 utils.py:
```

def validate_input():
return True

```

User Question: What does my main.py file do?
```

Step 6: AI Response

```
 AI: Your main.py file serves as the entry point of your application. Here's what it doe

**Main Function**:
- Prints "Hello World!" to the console
- Calls validate_input() from utils.py for input validation

**Entry Point**:
- Uses the standard Python idiom `if __name__ == "__main__"` to run main() when executed
- This makes it the primary executable file of your project

**Dependencies**:
- Imports and uses validate_input() from utils.py, showing a modular code structure
```

What Gets Generated

Repository Summary Cache (`repo_summary.json`)


```
{
  "username": "octocat",
  "repo_name": "Hello-World",
  "total_files": 8,
  "file_summaries": [
    {
      "path": "main.py",
      "type": ".py",
      "size": 245,
      "functions": ["main"],
      "classes": [],
      "imports": ["utils"],
      "description": "Main application entry point"
    },
    {
      "path": "utils.py",
      "type": ".py",
      "size": 156,
      "functions": ["validate_input", "helper"],
      "classes": [],
      "description": "Helper functions"
    }
  ],
  "file_types": {
    ".py": 3,
    ".md": 2,
    ".json": 1
  }
}
```

Example Conversations

Analyzing a Django Project

```
You: Tell me about my Django project structure
 AI: Your Django project follows standard conventions:

**Main Components:**
- manage.py: Command-line utility for administrative tasks
- settings.py: Configuration file with database, middleware, and app settings
- urls.py: URL routing configuration mapping URLs to views
- views.py: Contains business logic functions that handle HTTP requests

**Apps Structure:**
- You have a 'symptoms' app handling disease prediction logic
- Models define database schema for symptoms and diseases
- Templates provide HTML structure for user interface
```

Analyzing Data Files

```
You: What datasets do I have?
 AI: Your repository contains 4 datasets:

**Training Data:**
- symptoms_data.csv: 1,200 rows with symptom combinations and disease labels
- disease_descriptions.csv: Descriptions and metadata for 42 diseases

**Reference Data:**
- precautions.csv: Recommended precautions for each disease
- severity_weights.csv: Severity scores for different symptoms (1-10 scale)

**Usage**: These files work together for a disease prediction system where symptoms are w
```

Best Use Cases

- **Code Review**: "Find any potential bugs in my code"
- **Documentation**: "Explain how this authentication system works"
- **Onboarding**: "What does this repository do and how do I get started?"
- **Analysis**: "What are the main functions in my API?"
- **Learning**: "How is the database schema structured?"

Advanced Features

Smart File Detection

- Automatically identifies relevant files based on your question
- Understands relationships between files (imports, dependencies)
- Prioritizes main files, configuration, and domain-specific code

Intelligent Caching

- Creates summaries on first run for faster subsequent queries
- Regenerates cache automatically to stay up-to-date
- Lightweight storage of file metadata and structure

Debug Mode

Add this to see exactly what's happening:

```
# In client.py, uncomment debug lines to see:
# - Exact file content being analyzed
# - LLM prompt and response details
# - Character counts and content validation
```

Files You Need

`server.py` (GitHub API Handler)

```
#!/usr/bin/env python3
# Handles GitHub API calls and file fetching
# No changes needed - use as provided
```

`client.py` (Main Application)

```
#!/usr/bin/env python3
# Main application with AI integration
# Contains the two-step LLM workflow
```

`.env` (Configuration)

```
GOOGLE_API_KEY=your_google_gemini_api_key
GITHUB_TOKEN=optional_github_token_for_private_repos
```

⚡ Performance

- **Startup**: 2-5 seconds (loads repository once)
- **Per Query**: 3-8 seconds (2 LLM calls + file fetching)
- **Cost**: ~2-5 cents per conversation (depending on repository size)
- **Accuracy**: High - analyzes actual code content, not just file names

🔧 Troubleshooting

"Rate limit exceeded"
→ Wait 24 hours or upgrade your Google API plan

"File appears empty"
→ File might contain only comments/whitespace - this is normal

"No relevant files found"
→ Try rephrasing your question or asking about specific files

Ready to analyze your code? Just run the command and start asking questions about your repository! 🚀

⁂

1. https://github.com/waveupHQ/github-repo-analyzer
2. https://github.com/anuraghazra/github-readme-stats
3. https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-readmes
4. https://ubos.tech/mcp/github-repository-analyzer/
5. https://www.youtube.com/watch?v=eVGEea7adDM
6. https://github.com/topics/readme-template?l=markdown&o=desc&s=
7. https://dev.to/akdevcraft/github-repository-readme-template-6h2
8. https://gist.github.com/danielecook/94272f387d3366070d2546e2eadefe57
9. https://blog.devops.dev/building-an-ai-powered-github-repository-analyzer-with-fastapi-react-openai-52efd6796636