Research Document: AI-Powered Angular Upgrade Solution Analysis

**GitHub Copilot Integration Approach**

**Benefits of GitHub Copilot**

* Provides AI-powered code suggestions and completions in real-time ( completion/ code replacement would have to be developed manually for us)
* Seamlessly integrates with Visual Studio Code and supports Angular development, which is a convenient solution for developers.
* Offers contextual assistance throughout the software development lifecycle, in general, it seems that GitHub is able to get very good context without us having to develop a parser to extract and provide llm with context for the project

**Extension Development Capabilities**

* GitHub Copilot Extensions platform allows building custom integrations, which we can use to create an angular upgrade extension
* Extensions are developed using TypeScript, not Python, which may be a small issue
* Supports natural language interactions and multi-file operations, potentially allowing us to create an agent-like solution.
* Can leverage existing tools and services through the extension framework

**Technical Implementation Analysis**

**GitHub Copilot Extension Development**

* Uses TypeScript as the primary development language
* Provides built-in APIs for code generation and manipulation
* Offers debugging tools and CLI for extension development (good for angular as we can use LLM to aid us in debugging possible compilation errors)

**VS Code Extension Alternative**

* Requires more manual implementation of AI integration features
* Needs additional setup for language processing and code generation
* Development process is more complex and time-consuming
* Requires building authentication and API integration from scratch

**Development Considerations**

**Advantages of Copilot Approach**

* Ready-made AI infrastructure (we don’t have to develop our own)
* Built-in support for code understanding, generation and debugging
* Extensive documentation and development tools

Recommendation

Based on the research, developing a GitHub Copilot extension is a very viable approach for creating an AI-powered Angular upgrade solution because:

1. The existing AI infrastructure can be leveraged without building from scratch
2. Native integration with VS Code provides a seamless developer experience
3. The extension framework is well-documented and supported
4. Multiple MIT-licensed examples are available as reference implementations
5. Development time and complexity would be significantly reduced compared to a standalone VS Code extension or a hard-coded bedrock approach

**Comparing GitHub Copilot Extension vs AWS Bedrock Integration**

**Integration and Development**  
GitHub Copilot extensions offer seamless IDE integration and immediate code assistance without requiring complex setup or infrastructure. In contrast, a hardcoded Python program using AWS Bedrock would need manual implementation of API calls, authentication, and response handling. Additionally, Copilot takes into account possible errors with llm-generated code. With our current approach, we are requiring angular

**Contextual Understanding**  
Copilot extensions demonstrate superior contextual awareness by analyzing your entire codebase, open files, and project structure. The tool adapts to your coding style and provides suggestions based on the full context of your development environment. A standalone Bedrock implementation would lack this comprehensive understanding of the development context. We would have to develop a parser/implementation that would provide the llm with a similar level of context.  
Findings and Recommendations for an Angular Upgrade Project Using GitHub Copilot

**Overview**

This document summarizes research findings on leveraging GitHub Copilot and related tools for upgrading an Angular project. It also highlights potential strategies, tools, and challenges to consider during the upgrade process.

**Key Findings**

**1. GitHub Copilot Capabilities**

* **Training Limitations**: GitHub Copilot (via Claude v1) is trained up to Angular 16 but does not support newer versions beyond this[6](https://docs.github.com/en/copilot/building-copilot-extensions/about-building-copilot-extensions).
* **Customizability**:
  + Copilot can be extended with custom prompts tailored to specific use cases.
  + Customized models can be built for individual repositories, enabling deeper understanding of the codebase[6](https://docs.github.com/en/copilot/building-copilot-extensions/about-building-copilot-extensions).
* **External LLM Integration**: While external large language models (LLMs) can provide additional context to Copilot, this approach is not clean or seamless[6](https://docs.github.com/en/copilot/building-copilot-extensions/about-building-copilot-extensions).
* **Agent Mode**: Only Copilot's Agent Mode supports iterative debugging, which is essential for complex tasks like migrations[6](https://docs.github.com/en/copilot/building-copilot-extensions/about-building-copilot-extensions).

**2. Building Copilot Extensions**

* **Documentation and Support**:
  + Creating extensions for Copilot is a relatively new area with limited documentation available[6](https://docs.github.com/en/copilot/building-copilot-extensions/about-building-copilot-extensions)[9](https://docs.github.com/en/copilot/using-github-copilot/using-extensions-to-integrate-external-tools-with-copilot-chat).
  + Resources like GitHub Docs and walkthroughs provide foundational guidance for building extensions[3](https://resources.github.com/learn/pathways/copilot/extensions/building-your-first-extension)[7](https://docs.github.com/en/copilot/building-copilot-extensions/setting-up-copilot-extensions).
* **Extension Benefits**:
  + Extensions allow integration with external tools and customization of the Copilot experience.
  + They can be private (for internal use) or public (shared via the GitHub Marketplace)[6](https://docs.github.com/en/copilot/building-copilot-extensions/about-building-copilot-extensions)[7](https://docs.github.com/en/copilot/building-copilot-extensions/setting-up-copilot-extensions)[9](https://docs.github.com/en/copilot/using-github-copilot/using-extensions-to-integrate-external-tools-with-copilot-chat).

**3. Alternative Models**

* **O3 Mini Model**:
  + Advantages: Recent training data, low cost, and a large output token limit (100k tokens compared to Claude Sonnet v2's 8k tokens).
  + Trade-offs: May not generate code as effectively as Claude but could suit specific use cases like Angular upgrades[6](https://docs.github.com/en/copilot/building-copilot-extensions/about-building-copilot-extensions).

**4. Angular Upgrade Considerations**

* **Step-by-Step Approach**:
  + Best practice involves upgrading Angular one major version at a time to handle breaking changes incrementally[5](https://stackoverflow.com/questions/77325388/how-to-upgrade-badly-outdated-angular-project).
  + Third-party library compatibility must be addressed during each step of the upgrade process.
* **Tool Recommendations**:
  + Use the Angular Update Guide (<https://update.angular.io/>) for structured guidance.
  + Consider creating a new project using the latest Angular CLI version if the existing project is significantly outdated[4](https://stackoverflow.com/questions/54015248/what-is-the-correct-method-for-upgrading-angular-project-from-v2-v7)[5](https://stackoverflow.com/questions/77325388/how-to-upgrade-badly-outdated-angular-project).

**Challenges**

1. **Limited Training on Latest Angular Versions**:
   * Copilot's current capabilities may not fully support projects using Angular versions beyond 16.
2. **Complexity of Extension Development**:
   * Limited documentation and resources make building custom extensions challenging.
3. **Third-Party Library Dependencies**:
   * Upgrading libraries alongside Angular introduces additional complexity due to API changes and compatibility issues.

**Recommendations**

**1. Leverage GitHub Copilot Extensions**

* Build custom extensions tailored to your repository for better context understanding.
* Use Agent Mode for iterative debugging during migration tasks.

**2. Explore Alternative LLMs**

* Integrate O3 Mini for tasks requiring recent training data or large output tokens, especially if cost efficiency is a priority.

**3. Plan a Structured Upgrade Process**

* Follow a sequential upgrade strategy using tools like the Angular Update Guide.
* Address third-party library compatibility issues progressively during each version upgrade.

**4. Invest in Training and Documentation**

* Train team members on building and using GitHub Copilot extensions.
* Document lessons learned during the upgrade process to streamline future migrations.