1. **Executive summary**

**2. Introduction**

2.1 Background

Angular undergoes regular version updates that introduce new features, performance improvements, and architectural changes. For organizations like BMO with extensive Angular codebases, upgrading between major versions represents a significant technical challenge requiring substantial code modifications. Traditional manual approaches are time-consuming and error-prone, creating a need for AI-assisted solutions that can streamline this process while maintaining code quality.

2.2 Purpose of Report

This report has been prepared specifically for BMO to assist in selecting the optimal AI solution for Angular upgrade projects. We provide a comprehensive comparison between two potential approaches: a GitHub Copilot extension customized for Angular upgrades and a bespoke software package built on Anthropic's Sonnet v2 model. Our analysis aims to equip BMO's technology leadership with the information necessary to make an informed decision that aligns with the organization's technical requirements and strategic objectives.

2.3 Scope of Comparison

This comparison evaluates both solutions across key dimensions including:

* Technical capabilities and accuracy in Angular upgrade scenarios
* Implementation timelines and resource requirements
* Operational considerations including maintenance and workflow integration
* Cost structure including development costs and ongoing expenses
* Control over the solution and flexibility to adapt to specific requirements

The scope focuses specifically on Angular upgrade assistance within BMO's technology environment, examining how each solution addresses the challenges of migrating between Angular versions.

3.1 GitHub Copilot

GitHub Copilot is an AI-powered coding assistant developed by GitHub in collaboration with OpenAI. For the specific task of upgrading Angular projects, we've evaluated both the base GitHub Copilot application and the potential for a custom GitHub Copilot extension.

The base GitHub Copilot application provides real-time code suggestions directly within the developer's IDE, drawing on context from the current file, related files, and comments. However, our analysis indicates that for the specialized task of Angular version upgrades, the base application produces results that left much to be desired. The suggestions often fail to address the specific complexities involved in migrating between Angular versions, such as handling deprecated APIs, adapting to architectural changes, and implementing new best practices.

Through extensive testing and development, we found that creating a custom GitHub Copilot extension dramatically improved these results. While the regular GitHub Copilot did not have very good accuracy for Angular upgrades, our custom extension approach was able to achieve far greater accuracy by specifically targeting the unique challenges of Angular migration. This extension would enhance the base Copilot functionality by:

1. Providing specialized prompts tailored to Angular upgrade scenarios
2. Incorporating Angular version-specific migration knowledge
3. Maintaining a centralized repository of upgrade patterns and solutions
4. Offering contextual guidance based on the specific Angular components being upgraded

The extension would integrate with the existing GitHub Copilot infrastructure while adding domain-specific intelligence for Angular migrations. This approach leverages the existing development environment integration that Copilot provides while enhancing its capabilities for the specialized task of Angular upgrades.

3.2 Custom Software Package using Sonnet v2

The alternative solution we've evaluated is a completely custom software package built using Anthropic's Sonnet v2 model. This approach involves developing a purpose-built application from the ground up, specifically designed for Angular upgrade assistance.

Sonnet v2 is a more advanced large language model that demonstrates superior accuracy and understanding of complex software engineering tasks compared to the models underlying GitHub Copilot. Our custom solution would harness this capability through a specialized application architecture that includes:

1. A dedicated front-end interface for Angular project analysis and upgrade planning
2. Custom prompt engineering specifically optimized for Angular version migrations
3. Integration with Angular-specific static analysis tools to identify upgrade requirements
4. Version-specific transformation rules and patterns built into the system
5. A feedback mechanism to continuously improve upgrade suggestions
6. Comprehensive documentation generation for completed upgrades

This solution would be deployed on AWS infrastructure, with costs incurred based on actual usage of compute resources and API calls rather than per-seat licensing. The custom nature of this solution means that every aspect of the system would be tailored specifically for Angular upgrade workflows, without the constraints of adapting to a pre-existing platform.

The custom Sonnet v2 solution offers greater accuracy and specialization at the cost of longer development time and higher initial investment. It represents a ground-up approach where every component is designed specifically for the Angular upgrade use case, rather than extending a general-purpose tool.

Detailed Analysis

4.1 Flexibility in Using LLMs

GitHub Copilot offers a streamlined approach to LLM integration but with certain constraints on model selection. The platform is built on OpenAI's models, which limits the ability to switch between different LLM architectures or providers. While GitHub Copilot extensions can enhance functionality, they still operate within the boundaries of the underlying model ecosystem that GitHub has established.

In contrast, a custom solution using Sonnet v2 provides significantly greater flexibility in LLM selection and implementation. This approach allows for complete freedom to choose, fine-tune, or even switch between different models as requirements evolve. The custom solution can be architected to accommodate multiple models simultaneously or to transition between models without major rearchitecting. This flexibility extends to model versioning, allowing the organization to adopt newer versions of Sonnet or even different models entirely as they become available, without being dependent on GitHub's update schedule or model choices.

4.2 Cost Analysis

From a cost perspective, the two approaches present different financial models that should be carefully considered. GitHub Copilot offers lower upfront development costs since much of the infrastructure is already in place. Development of a GitHub Copilot extension requires less time and fewer specialized resources compared to building a complete custom solution. However, this comes with the ongoing cost of GitHub Copilot licenses for each developer or user who needs access to the system, which follows a per-seat pricing model regardless of actual usage volume.

The custom Sonnet v2 solution involves higher initial development costs due to the need to build everything from scratch, including infrastructure, integration points, and user interfaces. However, the long-term cost structure is based on actual AWS usage rather than per-seat licensing. This means costs scale directly with computational resources consumed and API calls made, potentially offering better economics for high-volume usage scenarios. Organizations with existing AWS infrastructure and expertise may find this model more predictable and manageable over time, especially if the number of users is large relative to the actual compute requirements.

4.3 Control Over Software

Control over the software represents one of the starkest differences between the two approaches. With GitHub Copilot, control is inherently limited by the platform's architecture and policies. While extensions provide a degree of customization, fundamental aspects of how the model operates, its training data, and its response characteristics remain under GitHub's control. Changes to GitHub's policies, features, or pricing could impact the solution with limited recourse for the organization.

A custom Sonnet v2 solution offers comprehensive control over every aspect of the software. This includes control over model deployment, fine-tuning parameters, integration points, security protocols, and data handling practices. The organization can implement specific governance requirements, compliance measures, and operational procedures tailored to their exact needs. This level of control extends to future development roadmaps, allowing the organization to prioritize features and improvements based on their specific business requirements rather than being dependent on GitHub's development priorities.

4.4 Ease of Use

GitHub Copilot excels in ease of use due to its seamless integration with development environments and established workflows. Developers familiar with GitHub's ecosystem can quickly adopt Copilot with minimal additional training. The extension-based customization approach builds upon this familiar foundation, reducing the learning curve for both developers and end-users. The platform's design prioritizes user experience, with features like inline suggestions and context-aware completions that enhance productivity without requiring significant changes to existing workflows.

The custom Sonnet v2 solution presents more challenges in terms of ease of use, particularly during initial implementation. Without the benefit of GitHub's established user interfaces and integration points, the custom solution requires careful attention to user experience design. However, this challenge also represents an opportunity to create interfaces and workflows specifically tailored to the organization's needs and preferences. While initially more complex to implement, a well-designed custom solution can ultimately provide a more streamlined experience for specific use cases than a general-purpose tool like GitHub Copilot.

4.5 Turn Around Time (Number of Iterations: HIL)

Turn around time and the number of human-in-the-loop (HIL) iterations required represent important operational considerations. GitHub Copilot, even with extensions, typically requires more HIL iterations to achieve optimal results for specialized tasks. The general-purpose nature of the underlying models means that outputs often need human refinement, especially for domain-specific applications. However, the development of the solution itself can be accomplished with fewer iterations due to the established platform foundation.

The custom Sonnet v2 solution, while requiring more development iterations initially, can be optimized to reduce the number of HIL iterations needed during actual use. By fine-tuning the model specifically for the target domain and implementing purpose-built prompt engineering, the custom solution can produce more accurate first-pass results. This potentially reduces the need for human intervention in the workflow, improving overall efficiency once the system is fully implemented. The trade-off is clear: more development iterations upfront for fewer operational iterations later, versus faster development but potentially more human refinement needed during ongoing use.

4.6 Prompting Flexibility

Prompting flexibility is a critical factor in achieving accurate and useful outputs from any LLM-based solution. GitHub Copilot's standard implementation offers limited prompting flexibility, with predefined patterns that work well for general coding tasks but may be insufficient for specialized domains. Through extensions, this flexibility can be enhanced, allowing for more sophisticated prompt engineering and context management. However, these extensions still operate within the constraints of the GitHub Copilot architecture.

A custom Sonnet v2 solution offers maximum prompting flexibility, allowing for the implementation of complex prompt engineering techniques tailored to specific use cases. This includes the ability to develop domain-specific prompt templates, implement sophisticated context management, and create centralized prompt repositories with version control. The custom approach enables techniques like few-shot learning, chain-of-thought prompting, and retrieval-augmented generation to be implemented exactly as needed. This level of prompting flexibility directly contributes to the superior accuracy potential of the custom solution, as prompts can be precisely engineered for the specific tasks at hand rather than adapted to fit within a predefined system.

**Comparative Analysis**

5.1. GitHub Copilot

**5.1.1. Positive Points**

* **Rapid Development Timeline**: GitHub Copilot's existing infrastructure allows for faster implementation of solutions, reducing time-to-market.
* **Seamless IDE Integration**: Already integrated with popular development environments, providing a familiar workflow for developers.
* **Extensibility**: GitHub Copilot extensions can be developed to enhance functionality with specialized prompts and context.
* **Continuous Updates**: Benefits from regular updates and improvements from GitHub without additional maintenance effort.
* **Established Ecosystem**: Access to a mature ecosystem with existing documentation, community support, and best practices.
* **Lower Development Costs**: Reduced initial development costs as the base platform is already established.
* **Centralized Prompt Management**: GitHub Copilot extensions can support centralized prompt repositories and management systems.
* **Scalability**: Built on infrastructure designed to handle enterprise-scale deployments.

**5.1.2. Limitations**

* **Base Application Limitations**: The standard GitHub Copilot implementation without customization produces suboptimal results for specialized tasks.
* **Less Control Over Core Model**: Limited ability to modify the underlying model architecture compared to a custom solution.
* **Dependency on GitHub's Infrastructure**: Subject to GitHub's service availability and potential policy changes.
* **Subscription Costs**: Requires ongoing GitHub Copilot subscription fees for all developers using the solution.
* **Learning Curve for Extension Development**: Team needs to become familiar with GitHub Copilot's extension development framework.
* **Less Specialized for Domain-Specific Tasks**: Without customization, may not be optimized for highly specialized industry requirements.
* **Pricing Model Based on Seats**: Cost scales with number of users rather than with actual usage or compute resources.

**5.1.3. Performance on Comparison Criteria**

* **Accuracy**: Medium-high when enhanced with custom extensions and specialized prompts; base application accuracy is insufficient without customization.
* **Development Time**: Low-medium; significantly faster than building a custom solution from scratch.
* **Maintenance Requirements**: Low; GitHub handles core maintenance, team only maintains the extension.
* **Customizability**: Medium; highly customizable through extensions but with limitations on modifying the core model.
* **Integration Capabilities**: High; designed to work within existing development workflows and tools.
* **Cost Structure**: Lower upfront development costs but guaranteed ongoing subscription fees per user regardless of actual usage.
* **Scalability**: High; built on infrastructure designed for enterprise-scale use.
* **Security**: Medium-high; benefits from GitHub's security practices but introduces third-party dependencies.

5.2. Custom Software Package using Sonnet v2

**5.2.1. Positive Points**

* **Superior Model Accuracy**: a customized solution will allow us to use the latest and greatest models giving far better results compared to relying on GitHub Copilot's underlying models.
* **Complete Customization Control**: Full ability to tailor every aspect of the solution to specific requirements.
* **Proprietary Solution**: Ownership of the entire technology stack without dependency on third-party platforms.
* **Domain-Specific Optimization**: Can be precisely tuned for the exact use cases and domain knowledge required.
* **Flexible Deployment Options**: Can be deployed on-premises, in private clouds, or as a managed service based on requirements.
* **No External Licensing Dependencies**: Not subject to changes in third-party services, pricing, or policies.
* **Custom Integration Capabilities**: Can be designed to integrate with any existing systems without platform limitations.
* **Usage-Based Cost Structure**: AWS costs scale directly with actual usage rather than per-seat licensing.
* **Specialized Prompt Engineering**: Purpose-built prompt management system optimized for the specific use case.

**5.2.2. Limitations**

* **Significant Development Time**: Building from scratch requires substantially more time for development, testing, and deployment.
* **Higher Initial Investment**: Requires greater upfront resources for development, infrastructure, and specialized expertise.
* **Maintenance Responsibility**: All maintenance, updates, and improvements must be handled internally.
* **Infrastructure Requirements**: Needs dedicated infrastructure for model hosting and serving, increasing operational complexity.
* **Specialized Expertise Required**: Demands team members with deep expertise in AI model development and deployment.
* **AWS Costs**: While potentially more cost-effective for certain usage patterns, still incurs ongoing AWS infrastructure and API costs.
* **Longer Time-to-Market**: Extended development timeline delays realization of business benefits.

**5.2.3. Performance on Comparison Criteria**

* **Accuracy**: High; Sonnet v2 provides superior baseline performance for specialized tasks.
* **Development Time**: High; requires building all components from scratch.
* **Maintenance Requirements**: High; full responsibility for all aspects of the solution.
* **Customizability**: Very high; complete control over all aspects of the solution.
* **Integration Capabilities**: High but requires custom development for each integration point.
* **Cost Structure**: Higher upfront costs but potentially more predictable and controllable long-term costs that scale with actual usage rather than user count.
* **Scalability**: Medium-high; can be designed for scalability but requires dedicated engineering effort.
* **Security**: High; can implement custom security measures tailored to specific requirements.

**6. Comparative Analysis Summary**

6.1 Strengths and Weaknesses

**GitHub Copilot Extension:**

* **Key Strengths**: Rapid development timeline, seamless IDE integration, lower initial development costs, and minimal maintenance requirements. The extension approach significantly improves accuracy over the base application while leveraging existing infrastructure.
* **Primary Weaknesses**: Limited control over the underlying model, ongoing per-seat subscription costs regardless of usage, and constraints on customization depth. Requires more human-in-the-loop iterations for specialized tasks.

**Custom Sonnet v2 Solution:**

* **Key Strengths**: Superior model accuracy, complete control over all aspects of the solution, usage-based cost structure, and maximum flexibility in prompt engineering and model selection. Optimized specifically for Angular upgrade workflows.
* **Primary Weaknesses**: Significantly longer development timeline, higher initial investment, greater maintenance responsibility, and more complex infrastructure requirements. Requires specialized AI expertise to develop and maintain.

6.2 Cost-Benefit Analysis

**GitHub Copilot Extension:**

* **Costs**: Lower upfront development investment but ongoing GitHub Copilot license fees for each developer. Predictable subscription model with potential for cost increases as team size grows.
* **Benefits**: Faster time-to-market, leveraging existing developer familiarity with GitHub tools, reduced maintenance burden, and continuous platform improvements from GitHub.
* **ROI Timeline**: Shorter path to initial value realization but potentially higher long-term costs for large teams.

**Custom Sonnet v2 Solution:**

* **Costs**: Substantial initial development investment, infrastructure setup costs, and ongoing AWS usage fees. Requires dedicated resources for maintenance and updates.
* **Benefits**: Higher accuracy reducing manual correction time, complete control over future development, usage-based cost scaling, and potential for deeper integration with BMO-specific systems.
* **ROI Timeline**: Longer path to initial value realization but potentially more cost-effective long-term for high-volume usage scenarios.

**7. Implementation Considerations**

7.1 Integration with Existing Systems

**GitHub Copilot Extension:**

* Seamlessly integrates with existing GitHub repositories and development workflows
* Minimal disruption to current IDE setups as Copilot is already compatible with popular environments like VS Code, Visual Studio, and JetBrains IDEs
* Can leverage existing CI/CD pipelines with minor modifications
* May require configuration to work with BMO's specific security protocols and network constraints
* Limited ability to integrate deeply with proprietary internal tools unless specific extension points are developed

**Custom Sonnet v2 Solution:**

* Requires more extensive integration planning but offers greater flexibility for connecting with BMO's internal systems
* Can be designed from the ground up to interface with existing Angular codebases, testing frameworks, and deployment pipelines
* Potential for deeper integration with BMO-specific tools, security frameworks, and governance processes
* Ability to implement custom APIs for seamless connection with other internal systems
* May require additional infrastructure setup and network configuration to operate within BMO's environment

7.2 Training and Adoption

**GitHub Copilot Extension:**

* Lower training barrier for developers already familiar with GitHub Copilot
* Incremental learning curve focused only on Angular-specific extension features
* Can leverage existing GitHub Copilot documentation and supplement with extension-specific materials
* Potential resistance may be limited as the base technology is already established in the industry
* Adoption can be phased, starting with early adopters before organization-wide rollout

**Custom Sonnet v2 Solution:**

* Requires more comprehensive training program for the entirely new system
* Need for detailed documentation covering all aspects of the custom solution
* Higher initial learning curve but potentially more intuitive for Angular-specific tasks once learned
* May face more significant adoption challenges as a completely new tool in the workflow
* Opportunity to design training specifically for BMO's Angular upgrade scenarios and development practices

7.3 Scalability and Future-Proofing

**GitHub Copilot Extension:**

* Inherits GitHub Copilot's enterprise-grade scalability
* Future Angular versions would require extension updates, but the base platform continues to evolve independently
* Limited control over the long-term roadmap of the underlying technology
* Dependent on GitHub's continued support and development of the Copilot platform
* May face challenges if BMO's requirements diverge significantly from GitHub's development direction

**Custom Sonnet v2 Solution:**

* Can be architected specifically for BMO's anticipated scaling needs
* Complete control over the upgrade path to support future Angular versions
* Ability to incorporate new AI models as they become available without platform constraints
* More adaptable to evolving organizational requirements and changing technical landscapes
* Higher initial investment creates a foundation that can evolve with BMO's specific needs
* Requires dedicated resources for ongoing development to maintain future compatibility