Justification for a Proof of Concept (POC) on Cursor AI

Objective: To select an AI coding assistant that can understand our existing AI service architecture and automate the creation of new services from this template. The primary goal is to reduce developer effort by having the AI handle the repetitive but necessary modifications across multiple files.

Our Architecture: We use a standardized, template-based MLOps project structure. A typical service includes src, config, scripts, a Dockerfile, and various API and data handling files. When creating a new service, 80–90% of this codebase remains the same, with changes required only in 2–3 key files to accommodate a new model or data source.

Core Requirement: The ideal tool must be able to perform targeted, multi-file modifications based on high-level commands, with a full understanding of the entire repository's context.

Comparative Analysis: Cursor vs. Alternatives

1. Cursor: The AI-Native Code Editor

Cursor is built from the ground up as an editor for AI-driven development. It is a fork of VS Code, so the user interface is already familiar to our developers.

How it addresses our requirement: A developer opens our service template in Cursor. To create a new service, they would use the integrated chat and provide a high-level prompt, explicitly referencing files from our architecture.

Strengths for Our Architecture:
- Unmatched Codebase Context: Cursor's ability to instantly reference any file or folder with the @ symbol is its killer feature.
- Seamless Multi-File Edits: Deep integration enables planning and execution across multiple files in one operation.
- Agentic Workflow: Cursor can autonomously identify files, write code, and apply changes.

Potential Weaknesses:
- Requires a New Editor: Developers must switch from their current editor to Cursor.

2. Qodo / CodeGPT: The IDE Extension

These tools enhance an existing IDE (like VS Code) with codebase awareness, operating primarily through an extension panel.

Strengths: No workflow disruption, strong repository analysis.
Weaknesses: Less integrated experience, context management requires more manual effort.

3. Gemini Code Assist: The Large-Context Assistant

Also an IDE extension, Gemini's key advantage is its extremely large context window, allowing it to process a massive amount of code at once.

Strengths: Massive context, ecosystem advantage for Google Cloud users.
Weaknesses: Less specialized for refactoring; more focused on generation than structured modifications.

Summary and Recommendation for POC

Evaluation Criteria	Cursor	Qodo / CodeGPT	Gemini Code Assist
Automation of Our Template	Excellent	Good	Good

Multi-File Editing	Excellent (Core Feature)	Good	Good
Codebase Context Precision	Excellent (@ referencing)	Good (Indexing-based)	Very Good (Large window)
Developer Effort	Lowest (High-level commands)	Low	Low
Workflow Change	High (New Editor)	None (Extension)	None (Extension)

Final Justification:

For the specific goal of automating the creation of new AI services from our established template, Cursor is the strongest candidate for a Proof of Concept.
While other tools are highly capable, Cursor's entire design philosophy is built around the exact task we need to perform: making intelligent, context-aware changes across an entire repository. Its ability to perform 'codebase surgery' with high-level commands is superior to the competition and directly addresses our primary pain point. The requirement to switch editors is a valid concern, but the potential productivity gains from its deeply integrated, multi-file editing capabilities justify a POC to measure the trade-off.