Questions for Prediction Market:

- Will the Name of the Feature *(e.g., “Proactive optimisitation” or “Anywhere in IDE access”)* be available in JetBrains IDE in next 6 months?

- Will the Name of the Feature *(e.g., “Proactive optimisitation” or “Anywhere in IDE access”)* be available in any AI tools for coding in next 6 months?

Examples:

- **Proactive automation of repetitive tasks feature be available in JetBrains IDE in next 6 months?**
  The proactive automation of repetitive tasks feature involves the AI autonomously identifying and executing routine coding activities that do not require unique human insight. This includes tasks such as formatting code, generating boilerplate code, running routine tests, and managing code documentation.

- **Will the in-IDE daily ecosystem summarizer feature be available in JetBrains IDE in next 6 months?**
  The In-IDE daily ecosystem summarizer is a tool or function designed to provide concise, actionable summaries of key activities, metrics, and significant events within a given ecosystem on a daily basis. The summarizer aims to aggregate and synthesize relevant information to help users stay informed and make informed decisions without needing to sift through large volumes of data.

Potential Features for Prediction Market:

1. **in-IDE daily ecosystem summarizer**

A Daily Ecosystem Summarizer is a tool or function designed to provide concise, actionable summaries of key activities, metrics, and significant events within a given ecosystem on a daily basis. The summarizer aims to aggregate and synthesize relevant information to help users stay informed and make informed decisions without needing to sift through large volumes of data.

2. **Proactive automation of repetitive tasks**

Proactive automation of repetitive tasks for a coding assistant involves the AI autonomously identifying and executing routine coding activities that do not require unique human insight. This includes tasks such as formatting code, generating boilerplate code, running routine tests, and managing code documentation.

3. **Project-wide context awareness**

Project-wide context awareness for an AI assistant in coding ensures that the AI understands the entire scope, structure, and history of a project, leading to more accurate and relevant code suggestions, bug fixes, and optimizations. This comprehensive understanding enhances productivity by providing context-sensitive assistance, improving code quality, reducing errors, and facilitating seamless collaboration among team members.

4. **Proactive documentation**

Proactive documentation for an AI coding assistant involves the AI automatically generating and suggesting relevant documentation as developers write or modify code. By understanding the code's context, purpose, and structure, the AI provides timely documentation, such as comments, function descriptions, and usage examples. This ensures that the codebase remains well-documented without extra manual effort, improving code readability, maintainability, and easing collaboration among team members.

5. **Proactive optimization**

Proactive optimization for an AI coding assistant involves the AI automatically identifying potential improvements in the code for performance, efficiency, and best practices. By continuously analyzing the codebase, the AI offers real-time suggestions for optimizing algorithms, reducing resource consumption, and enhancing code quality. This proactive approach helps developers maintain high performance and efficient code, reducing technical debt and improving overall application performance without waiting for manual reviews or refactoring sessions.

6. **Multi-file generation**

Multi-file generation for an AI coding assistant involves the AI autonomously creating and managing multiple interconnected files within a project. By understanding the project structure and dependencies, the AI generates all necessary files such as configuration files, modules, components, and documentation. This capability ensures coherence, reduces manual setup effort, and accelerates the development process, allowing developers to focus on higher-level coding tasks and functionality.

7. **Sketch2Code**

Sketch2Code for an AI coding assistant involves the AI autonomously converting sketch design files into production-ready code. By analyzing the visual elements, layouts, and interactions specified in the Figma design, the AI generates corresponding HTML, CSS, and JavaScript, or other relevant code.

8. **Annotating AI-code**

Annotating AI-generated code involves the AI automatically marking or tagging the sections of code it has produced. By clearly indicating which parts of the codebase were generated by the AI, this feature ensures transparency and helps developers quickly identify and review AI contributions. This capability enhances code accountability, makes debugging easier, and

fosters better collaboration by clearly distinguishing between human-written and AI-generated code.

9. AI real-time code refactoring

AI-Driven Real-Time Code Refactoring utilizes AI to anticipate code refactoring opportunities as developers write and even as they pause to think about potential solutions. By continuously analyzing the code structure and developer's patterns, the AI dynamically suggests refactoring strategies such as modularization, method extraction, and code optimization

10. Brain-computer interface integrated coding

Coding with BCI for an AI coding assistant involves the AI assisting in code generation and development through direct brain-computer interface inputs provided by any BCI technology. By translating neural signals into actionable coding commands and suggestions, the AI enables developers to interact with their coding environment more intuitively and efficiently. It involves both direct control of IDE with BCI-derived commands and brain-based adjustments of the workflow.

11. Business-context awareness

Business-context awareness for an AI coding assistant allows the AI to tailor its suggestions and solutions based on the specific business goals, industry regulations, and organizational workflows. This targeted assistance ensures that the generated code aligns with the company's strategic objectives, compliance requirements, and operational practices, thereby enhancing relevance, accelerating development processes, and supporting better business outcomes.

12. Proactive debugging

Proactive debugging for an AI coding assistant involves the AI automatically detecting and diagnosing potential bugs and issues in real-time as developers write code. By continuously analyzing the code for common errors, logical inconsistencies, and potential runtime issues, the AI provides immediate feedback and suggests fixes. This proactive approach helps developers catch and resolve bugs early in the development process, reducing debugging time, improving code reliability, and ensuring smoother and more efficient development cycles.

---

13. Project set-up assistance

Project set-up assistance for an AI coding assistant refers to the AI's capability to help developers initiate new projects by automating and streamlining the initial configuration and setup processes. This feature simplifies starting new projects and ensures consistency and adherence to best practices from the outset.

14. Emotion-aware debugging

The AI assistant can detect the emotional state of the developer through sensors or wearables and adjust its responses accordingly. For example, if it detects frustration, it might offer simplified explanations or take over more complex tasks temporarily to alleviate stress.

15. Full conversational project management and coding

An AI that can manage an entire software development project from start to finish through natural language conversations. It would handle everything from initial planning, task assignment, coding, to deployment, all guided by conversational input from team members.

16. Dynamic diagram generation

Automatically generate diagrams from textual descriptions or code comments. For example, a developer could type "draw network diagram" with specific details about the network layout, and the AI would generate a corresponding diagram.

17. Inline AI chat interface

Provide an inline chat interface where developers can type queries or commands and get immediate code suggestions or assistance. This should allow for immediate review and editing within the same screen.

18. Multi-Layered abstraction and inspection

Provide multiple levels of info abstraction for AI suggestions and actions, from high-level summaries to detailed views.

19. Integrated code review assistance

Assist with code reviews by using AI to understand the context of the review, related comments, and suggestions.

20. Visual refactoring assistant

Create visualizations to map out refactorings, showing both high-level and detailed relationships between code elements. This feature would help developers understand the impact of their changes on the entire codebase, including functions, signatures, and parameters within files.

21. Layered code understanding and visualization

Provide an AutoCAD or Visio-like interface that allows developers to see different layers of the code structure. This interface would help in understanding dependencies, circular references, and other complex relationships in a linear and structured manner.

22. Assisted prompt engineering

Develop an AI-assisted prompt engineering tool that helps refine questions based on iterative feedback. The AI would provide guidance on improving the clarity and completeness of the queries to get better responses.

23. Integrated environment recommendations

An AI tool that provides comprehensive recommendations based on the entire code environment. This could include specific fixes for deployment issues, such as those related to Docker, by examining code, configurations, and runtime environments.

24. Iterative code impact analysis

Provide detailed analysis and strategies for breaking circular dependencies and understanding their implications. The tool would offer multiple strategies and show their projected impact on the codebase.

25. Holistic project context visualization

Offer both visual and textual summaries of the entire project context to help developers understand the broad picture before diving into specific tasks. This helps prevent inadvertently causing new issues while solving existing problems.

26. Solution logic evaluation

Develop an AI feature that evaluates the overall logic and solution approach, not just style or syntax. This would provide a higher level of understanding and verification, suggesting better solutions when necessary.

27. Intelligent import and conversion suggestions

The IDE would offer intelligent prompts when it detects new source files or external libraries, suggesting potential actions to integrate them with existing project settings. This would include converting Maven projects to Gradle or vice versa and updating project files accordingly.

28. Proactive documentation-guided coding

This feature would prompt developers to write out their intended functionality in the form of documentation or comments first. The AI would then use this information to generate corresponding code, improving accuracy and reducing the time taken to manually write out routine segments of code.

29. Task-specific AI proposals

This feature would allow the AI to offer provisional steps or code for specific tasks before the developer moves on to another task. This ensures that the developer's workflow is not interrupted by having to revisit completed tasks.

30. Fallback suggestions when stuck

When the AI encounters a question it cannot answer, it should suggest alternative approaches or provide links to online resources like Stack Overflow for potential solutions.

31. AI-Driven repository merging assistant

This feature would assist in merging separate repositories into a Monorepo by automating the git subtree process. The assistant would ask clarifying questions about which branch and subdirectory to use, making the process seamless within the IDE.

32. Intelligent import and conversion suggestions

The IDE would offer intelligent prompts when it detects new source files or external libraries, suggesting potential actions to integrate them with existing project settings. This would include converting Maven projects to Gradle or vice versa and updating project files accordingly.

33. Step-by-step project integration

This feature would let developers integrate one repository at a time into a Monorepo, ensuring that each step is validated and complete before proceeding to the next. The AI assists at each step, verifying compatibility and updating necessary project files.

34. Proactive code dependency checker

An AI feature that proactively checks for code dependencies and potential issues before finalizing any proposed code changes. This includes prompting users to download necessary libraries and highlight dependencies that might introduce errors once integrated.

35. Pre-execution error predictor

This feature would simulate a pre-execution of the proposed changes in a large codebase, identifying potential errors and dependencies across multiple files. It provides warnings and suggestions to mitigate any issues before the code is executed or committed.

36. Speech-to-code interaction engine

A speech-to-code interaction feature allowing developers to communicate with the IDE using voice commands. This can speed up coding tasks, especially when formulating complex queries or writing algorithms, reducing the need to type detailed instructions.

37. Integrated performance deviation monitor

An AI tool that continuously monitors and learns the usual performance patterns of your system. It identifies and reports any deviations from these patterns, whether improvements or regressions, allowing you to understand the performance impact of recent changes in the code.

38. Non-intrusive AI notifications

Designed to be a non-intrusive AI assistant that can be called upon when needed. Notifications can be minimized or expanded based on the user's preference, ensuring that space on the screen is conserved, and interruptions are minimized.