AI chat into a "prompt engineer" prelude before unleashing the coding beast. In current IDEs like Cursor or Zed, the chat often blurs lines: It might auto-generate code mid-convo if it detects intent (e.g., "Fix this bug" triggers edits), or stick to chit-chat for clarifications, with some offering mode toggles (e.g., Cursor's "Apply" button or separate chat/code tabs). But your "boot loader" mode flips that inefficiency on its head: A dedicated, metric-driven warmup phase where the AI interrogates and refines until the prompt is bulletproof, minimizing hallucinations, rewrites, and token waste downstream. It'd make the IDE feel proactive and precise, especially for complex builds like AI agents or full apps.

Let's refine and expand this into our ForgeIDE vision—we've already got building blocks like context maps, previews, and metrics dashboards that slot in perfectly. I'll call this feature **PromptForge Mode**: A boot loader chat that scaffolds the ideal coding prompt through iterative engagement, only transitioning when readiness metrics hit green.

**Core Concept: PromptForge Mode Workflow**

* **Activation**: Trigger via a button ("Enter PromptForge") or auto-detect (e.g., if user says "Build an agent for X"). It's a modal chat overlay, separate from the main IDE chat to avoid confusion.
* **Build-Up Process**: The AI acts as a "requirements engineer," starting broad and drilling down. It builds context via questions, summaries, and user confirmations, tracking internal metrics (e.g., clarity, completeness) until they're satisfied.
  + **Engagement Style**: Conversational but structured—e.g., "Tell me more about the app's core flow," or "What edge cases should we handle?" It references project files, past sessions (via saved contexts), or even scans codebases for awareness.
  + **Metrics-Driven Gatekeeping**: Behind the scenes, the AI evaluates against a checklist (customizable by user). Only when all are 80%+ does it compile the "perfect prompt" and hand off to the coding agent.
* **Transition**: Once ready, it previews the final prompt (editable), then switches to coding mode—generating diffs/edits without full rewrites (tying into our earlier precision edit ideas).
* **Benefits Over Current IDEs**: No more vague "it codes when it thinks it should"—this enforces rigor upfront, reducing the "retarded" memory lapses or overzealous rewrites you hate. For example, if a UI panel worked before, it pulls from saved states to reference it explicitly in the prompt.

**Key Metrics for Readiness**

The AI would use a dynamic set of fulfillment metrics, displayed in a mini-dashboard within the chat (integrating our ForgeMetrics panel). These ensure the prompt covers all angles for high-quality output. Here's a table of examples:

| **Metric** | **Description** | **How It's Fulfilled** | **Threshold for "Ready"** |
| --- | --- | --- | --- |
| **Intent Clarity** | Measures how well the user's goal is defined (e.g., "Build a chat agent" vs. "Agent with RAG, tool calls, and error handling"). | AI asks clarifying questions; scores via semantic analysis (e.g., keyword coverage). | 90%—No ambiguities left. |
| **Context Completeness** | Awareness of project state (files, deps, past panels). | Pulls from context map/history; queries user on gaps (e.g., "Reference this old UI panel?"). | 85%—All relevant elements referenced. |
| **Scope Feasibility** | Checks if request fits API/cloud limits (e.g., token caps for long chains). | Cross-references with user's setup via metrics dashboard; suggests scaling if low. | 75%—Flags overkill early. |
| **Edge Case Coverage** | Identifies potential pitfalls (e.g., hallucinations in agents). | Prompts user for scenarios; auto-suggests based on common patterns. | 80%—At least 3-5 cases outlined. |
| **Prompt Optimization** | Ensures the final prompt is concise yet detailed (e.g., under 2k tokens). | AI iteratively refines drafts, measuring length/structure. | 95%—Balanced for efficiency. |
| **User Confidence** | Subjective user buy-in. | Ends with "On a 1-10, how ready is this?"; loops if low. | 80%—User confirms via quick poll. |

* **Visualization**: A progress radial chart in the chat sidebar fills as metrics climb (e.g., green arcs per category). If one lags (e.g., "Context at 50%—Need more on DB integration?"), the AI pivots questions there.
* **Fallbacks**: If metrics stall, offer "Bail to Manual Prompt" or "Import Template" (grabbing from repos as we discussed).

**Implementation Ties to Our IDE Features**

This slots right into what we've built:

* **Context Map Integration**: The boot loader feeds into the map—nodes for user responses, edges for refinements. Previews show how the prompt will evolve the graph without full rewrites.
* **Preview/Approval Panel**: Before handoff, preview the compiled prompt with diff highlights (e.g., "Added edge case from your last answer").
* **Save/Reload**: Snapshot the boot loader session—reload a half-built prompt later, remembering that "working UI panel" exactly.
* **Metrics Dashboard**: The global ForgeMetrics panel tracks overall "Prompt Quality Avg" across sessions, suggesting tweaks like "Upgrade API for better feasibility scoring."
* **Agent Builder Synergy**: For app/agent builds, this mode preps prompts for multi-agent swarms—e.g., questioning "How should agents collaborate?" to hit autonomy metrics.

In ForgeIDE, this would make AI chats feel like a co-pilot briefing before takeoff: Thorough, metric-backed, and user-centric. No more random code dumps or forgotten states—it builds trust and efficiency. If we added voice mode (like Grok 3's), it'd feel even more natural for Q&A. What do you think—tweak the metrics, or imagine how it'd handle a specific example like recovering a UI panel?