Confirming Understanding of Previously Answered Questions:

1. **Umicom GTK as a Submodule:** Yes – We will treat *Umicom GTK* as a separate repository in the Umicom Foundation GitHub account and include it in our main project as a **Git submodule**. You want step-by-step instructions on creating this new repository under the Umicom Foundation, cloning it locally, and then adding it as a submodule to the main Umicom project.

2. **Umicom LLVM Targets:** The initial compilation target for our custom LLVM is **x86 on Windows**. Later on, we will expand support to **Linux** (likely x86_64 on Linux) and to **RISC-V** (an open standard architecture). We even have plans for our own operating system (the **Umicom OS** repository) in the future. But for now, **Windows (PowerShell environment)** is our primary development platform, targeting x86 architecture first.

3. **Weekly Artifact Publication:** Yes – We will produce a weekly build artifact (starting with **Windows**). Specifically, we plan to bundle the compiled GTK and LLVM components into a **zip file** as a release each week. This zip will contain all necessary binaries for the IDE (e.g. compiled GTK libraries, LLVM tools, etc.) for easy distribution and testing on Windows. As we progress, each weekly release will be packaged and made available (likely via GitHub Releases).

4. **IDE Layout & Features:** The IDE's user interface will follow a familiar layout **similar to popular IDEs** like Visual Studio Code, Eclipse, NetBeans, or IntelliJ. That means:

5. A **splash screen** at startup featuring the Umicom logo (you've provided the logo asset already).
6. A standard **editor layout** with panels for file explorer/project tree, code editor, and so on – so it feels familiar to developers.
7. Additional **panels/windows** for:
   Console Output (runtime logs, program output),
   Build/Compile Messages (showing build progress and errors),
   Chat/AI Assistant Window where the user can interact with integrated Large Language Models (LLMs).

8. **LLM Integration:** We will integrate both **local LLMs** (from the list of open models we discussed in our documentation) and external ones via API (such as OpenAI's models, using the API keys you have). This means the IDE will have a chat or assistant panel that can interface with a selection of AI models – either running locally or via online API – to assist the developer (for example, code suggestions, explanations, etc.). We'll design the UI to allow choosing between these AI backends.

9. **Compiler Picker (Default Compiler):** The default compiler integrated into our environment will be **GCC (GNU Compiler Collection)**. We choose GCC because it supports multiple languages (C, C++, etc.) out-of-the-box and is robust for our needs. We will include other compilers (such as TinyCC, Clang/LLVM's own clang, etc.) as **optional components**:

10. Those additional compilers can be included as Git submodules (just like other dependencies) but can be toggled on/off in the IDE settings or build configuration.
11. We will **fork** the repositories of these tools/libraries (GCC, TinyCC, etc.) under our own GitHub organization when needed, and add them to our project as submodules. All these will be built **from source** as part of our build process (no pre-built binaries checked into source control).

12. For development, we'll work with source code submodules, but for end-users we will provide compiled binaries in the release packages. For example, the official releases on GitHub can

include the compiled executables for GCC (and any other tools we include) so that users can download a ready-to-run IDE bundle with all compilers and libraries included.

I **apologize for any repetition or confusion** earlier. I have now carefully recorded all the information you provided and will proceed with the deep research tasks using these confirmed answers as the starting point.

Let me know if everything above is correct and complete. Once you're satisfied, I will continue with the detailed research and planning as requested.
