Step we going to do:

Detailed Technical Workflow for Our AI Co-Pilot

Phase 1: Building the Core Engine (Days 1-5)

1. **Setup the Workshop (Day 1 - Today's Task):**
    - **Action:** Install the fundamental tools: Python and the VS Code editor.
    - **Goal:** Create the clean and professional environment where we will build everything.
2. **Establish AI Communication (Day 2):**
    - **Action:** We will use `pip` (Python's package installer) in the VS Code terminal to install a library to connect to a powerful LLM (e.g., `google-generativeai`). We'll then write a simple 10-line Python script to send a test question and print the AI's answer.
    - **Goal:** Confirm we have a working "phone line" to the AI's brain.
3. **Build the ABB Knowledge Base (Days 3, 4, 5):**
    - **Action:** This is where we handle the PDFs. We will download official ABB product manuals (e.g., for the AC500 PLC) and programming examples.
    - **Action:** We will install a Python library specifically for reading PDFs, such as `PyPDF2`.
    - **Action:** We will write a Python script that will:
        - Open each ABB PDF document.
        - Read the document page by page.
        - Extract all the text.
        - Save the extracted information into clean, simple `.txt` files.
    - **Goal:** Convert the valuable information locked inside ABB's PDF manuals into a structured text format that our AI can read and learn from. This is the heart of our RAG system.

Phase 2: Building the Application (Days 6-8)

1. **Create the Co-Pilot Interface (Day 6):**
    - **Action:** We will install the **Streamlit** library (`pip install streamlit`).
    - **Action:** We will write a new Python script that uses Streamlit's simple commands to create a web-based user interface. It will have a text box for the user to type their command and a display area for the AI-generated code. We will style it to look like an official ABB tool.
    - **Goal:** Build the user-friendly "dashboard" that an engineer will interact with.
2. **Implement the Safety Check (Day 7):**
    - **Action:** We will download and set up the open-source **MATIEC** compiler, which is a tool that can check for syntax errors in IEC 61131-3 Structured Text.
    - **Action:** We will write a Python function that takes the AI's generated code, saves it to a temporary file, and then runs the MATIEC tool on that file. Our function will read the output from MATIEC to see if the code is "Correct" or has "Errors".
    - **Goal:** Create our automated verification loop, the key feature that makes our project reliable and professional.

Regarding Advanced Interfaces like Modbus

You asked a brilliant question about creating an interface like Modbus.

- **What it's for:** Protocols like Modbus and OPC UA are used for **live communication** with real PLCs. Python libraries like

  `pymodbus` allow you to write code that can, for example, read a live sensor value from a physical ABB PLC or command it to turn on a motor.
- **Our Hackathon Scope:** For our 11-day project, our mission is to perfect the **AI-powered code generation and verification**. This is the core challenge.
- **The "Future Work" Vision:** Integrating a live Modbus connection is a fantastic "next step" for the project. In your final presentation, you can confidently state: *"The next phase of this Co-Pilot is to connect it directly to a physical or*

simulated ABB PLC using protocols like Modbus or OPC UA, enabling hardware-in-the-loop testing of the AI-generated logic." This shows the judges you have a complete vision for the product.