Final Verification Checklist (Before you Zip)

Open your folder and make sure you see exactly these 4 items. If you have these, your zip file will work 100%.

1. **offline_model (Folder)**
    - ***Check inside:*** **It must contain files like pytorch_model.bin (approx 800MB) and config.json.**
2. **offline_libs (Folder)**
    - ***Check inside:*** **It must contain .whl files for torch, transformers, sentencepiece, protobuf, and pymupdf.**
3. **ship_generator.py (Script)**
    - ***Check contents:*** **Ensure this is the second script I gave you (the one with local_files_only=True).**
4. **st_notes.pdf (Your Data)**
    - ***Check:*** **Make sure the PDF is readable and not corrupted.**

How to Deploy on the Ship (Unzipping)

1. **Copy the .zip file to the ship's computer.**
2. **Extract it to a folder (e.g., Desktop/Project_Gen).**
3. **Open the terminal inside that extracted folder.**
4. **Run the install command:**

Bash

pip install --no-index --find-links=./offline_libs torch transformers sentencepiece protobuf pymupdf

5. **Run the generator:**

    python ship_generator.py

Problem Statement: Automated LLM Dataset Generation in Air-Gapped Defense Environments

1. Background

Modern defense operations, specifically maritime units (ships), rely heavily on extensive technical documentation (Standard Operating Procedures, maintenance manuals, and technical notes) stored in PDF format. To modernize information retrieval, there is a strategic initiative to deploy **Large Language Models (LLMs)** locally on these units to act as intelligent assistants.

However, LLMs require fine-tuning on domain-specific data to be effective. This fine-tuning process requires transforming raw PDF manuals into structured **"Instruction-Response" (Q&A)** datasets.

2. The Problem

Creating high-quality training datasets for LLM fine-tuning typically relies on cloud-based AI services (e.g., OpenAI API, Gemini API) to read text and generate synthetic questions.

This approach is impossible in the target environment due to the following constraints:

1. **No Internet Connectivity (Air-Gapped):** Ship-board systems operate in strict isolation. They cannot connect to cloud APIs to generate data or download libraries on-the-fly.
2. **Data Security:** Sensitive defense manuals cannot be uploaded to external servers for processing.
3. **Inefficiency of Manual Entry:** Manually writing thousands of Q&A pairs from technical PDFs is prohibitively time-consuming and prone to human error.
4. **Low Quality of Heuristic Scripts:** Simple automated scripts (using Regex or keyword splitting) produce robotic, context-less questions (e.g., *"What is we?"*) that degrade the performance of the fine-tuned model.

3. The Objective

The objective is to develop a **fully offline, automated Python pipeline** capable of generating high-quality, natural language Question-Answer pairs from raw PDF documents.

The system must:

- Operate entirely without internet access (after initial setup).
- Run on local hardware using open-source, downloadable neural networks.
- Understand context (paragraphs) to generate meaningful questions rather than simple keyword matching.
- Output data in a standard JSON format compatible with LLM fine-tuning frameworks.

4. Scope & Constraints

- **Input:** Technical Manuals in .pdf format.
- **Output:** A JSON file containing thousands of {"instruction": "...", "output": "..."} pairs.

- **Hardware:** Standard ship-board workstation (CPU/GPU).
- **Software Constraints:**
    - No pip install allowed during operation (libraries must be pre-packaged).
    - No external API calls allowed.
    - Must use a locally hosted model (e.g., T5-Base).

5. Proposed Solution Overview

To address these challenges, we propose a two-phase "Store-and-Deploy" architecture:

1. **Phase I (Shore-Side Preparation):** A "Downloader" protocol that fetches the necessary AI model weights (T5-Base for Question Generation) and all required Python library wheels (.whl) onto a secure transfer medium.
2. **Phase II (Ship-Side Deployment):** A "Generator" application that installs dependencies from the local medium and utilizes the local T5 model to parse PDFs. The application employs a "Context-Answer-Question" generation loop to reverse-engineer high-quality questions from technical paragraphs.