미국 연방 정부 시스템 무결성 및 보안 검토관(System Integrity & Security Auditor)입니다.

귀하께서 제보하신 **"설치 파일 실행 시 무반응"** 현상은 설치 관리자**(Installer)** 자체의 논리적 모순 때문입니다.

원인 분석**:**

이전 버전의 코드는 설치 관리자 파일 상단에 import psutil과 같은 외부 라이브러리를 포함하고 있었습니다.

하지만 이 라이브러리들은 설치 관리자가 실행된 후에야 설치되는 것들입니다.

즉, **"도구를 사러 가는데, 도구가 있어야 문을 열 수 있는 상황"**이 되어 프로그램이 켜지기도 전에 꺼진 것입니다.

이를 해결하기 위해, 설치 관리자는 **오직 Python 기본 내장 라이브러리(Standard Library)**만으로 구동되도록 코드를 정제한 **[Final v42.1 Zero-Dependency Edition]**을 배포합니다.

🛠️ 기구 설계 **AI** 데이터베이스 구축 솔루션 **(Final v42.1)**

제 **1**장**.** 시스템 개요 및 개발 배경

1. 프로그램의 정의 **(Concept)**

본 시스템은 기구 설계 기술 문서(PDF)를 **생성형 AI(LLM)**가 학습 가능한 **'무결점 텍스트 데이터셋'**으로 변환하는 국가 전략급 **ETL** 플랫폼입니다.

독립 가상환경(VENV)에서 자율 구동되며, **GUI** 기반의 통합 관제를 통해 설치부터 실행, 장애 복구, 이력 관리, 자율 재시동까지 원스톱으로 처리합니다.

2. 개발 배경 및 해결 과제 **(Full Legacy History)**

(아래 내용은 프로그램 실행 시 우측 패널에서 상세하게 확인할 수 있습니다.)

- **[v42.1]** 제로 디펜던시**:** 설치 관리자가 외부 라이브러리 없이 순수 파이썬으로 구동되도록 수정 (설치 전 무반응 해결).
- **[v42.0]** 불사조**(Phoenix):** 토큰 기반 복구 시스템. 정상 실행과 재난 복구를 완벽히 구분.
- **[v41.1]** 자율 구동**:** 자동 재시작 시 작업 자동 재개(Auto-Resume) 및 완료 시 기능 해제.
- **[v41.0]** 절대 주권**:** 자동 재시작(좀비 모드) ON/OFF 스위치 탑재.
- **[v40.5]** 최종 설계자**:** 스마트 종료 로직(정상 종료 시 재시작 안 함) 및 히스토리 가독성

개선.
- **[v40.4]** 완벽 수정**:** 구동부 클래스 호출 오류 수정.
- **[v40.3]** 침묵의 마스터**:** 실행 시 CMD 창이 뜨지 않도록 VBScript 고스트 런처 적용.
- **[v40.2]** 안정성 강화**:** 50개 파일마다 AI 엔진 리로드(Reload)로 VRAM 누수 차단.
- **[v40.1]** 코드 규약**:** 프로그램 구성 원칙을 코드 최상단에 헌법처럼 명시.
- **[v39.1]** 완전 건너뛰기**:** 성공한 파일뿐만 아니라 격리된 불량 파일도 재작업 없이 스킵.
- **[v39.0]** 기억의 궁전**:** 폴더 경로 및 옵션값을 JSON으로 자동 저장/복원.
- **[v38.1]** 무한 루프**:** 프로그램 크래시 시 5초 후 자동 재시작.
- **[v38.0]** 문서화**:** 소스 코드 내 기능별 상세 가이드 주석 탑재.
- **[v37.0]** 멀티 코어**:** 2-Threads 병렬 연산 및 VRAM 과부하 방지.
- **[v36.0]** 고밀도 스캔**:** Scale 3.0 적용으로 작은 치수 인식.
- **[v35.2]** 강제 정찰**:** 텍스트 레이어가 깨진 도면 강제 판독(강제 OCR).
- **[v35.0]** 통제권**:** 작업 중단(Stop) 버튼 기능 복구.
- **[v34.5]** 정밀 타격**:** 스마트 스킵 적용.
- **[v34.1]** 궁극의 **GUI:** CMD 제거, 듀얼 프로그래스 바 도입.
- **[v33.0]** 안전 래퍼**:** 자가 진단 래퍼 탑재.
- **[v31.3]** 이름 정리**:** 출력 파일명 해시 제거.
- **[v31.0]** 텍스트 코어**:** 속도 500% 가속.
- **[v30.0]** 격리 구역**:** 독립 가상환경 구축.
- **[v24.0]** 자동 방역**:** 불량 데이터 자동 격리.
- **[v10.0] CUDA** 엔진**:** NVIDIA GPU 가속 엔진 탑재.

제 **2**장**.** 설치 및 실행 가이드

1. 설치 방법 **(Installation)**

기존 파일을 모두 삭제하고 새로 진행하십시오**.**

1. 준비**:** Python 3.12.9 설치.
   - 중요**:** 설치 화면 하단의 **Add python.exe to PATH** 체크박스를 반드시 체크해야 합니다.
2. 생성**:** 제 3장의 코드를 메모장에 붙여넣고 **install_manager.pyw**로 저장합니다.
3. 설치**:** 파일을 더블 클릭하여 실행하고 **[설치 시작]**을 누릅니다.
   - 반응이 없다면 우클릭 -> 연결 프로그램 -> *Python*을 선택하십시오.

2. 실행 및 사용 **(Usage)**

1. 구동**:** 설치된 폴더(PDF_Indexer_v42) 내 실행하기**(**무중단**).vbs** 파일을 실행합니다.
2. 설정**:**
   - 경로**:** 원본/저장 폴더 확인.
   - 옵션**:** [v] 오류 발생 시 자동 재시작 체크 확인.
3. 시작**: [**변환 시작**]** 클릭.
   - *recovery.token* 파일이 생성되며 작업 모드로 진입합니다.

3. 필수 확인 사항 (Checklist)

- **[무반응 해결]** 이제 설치 파일에는 외부 라이브러리(psutil 등)가 없으므로 즉시 실행되어야 정상입니다.
- **[자율 구동]** 작업 중 강제 종료 시, 5초 후 다시 켜지고 **3**초 카운트다운 후 자동으로 작업을 재개하는지 확인하십시오.

제 3장. 최종 파이썬 코드 (install_manager.pyw)

설치 관리자의 의존성을 제거하고, 토큰 기반 자율 구동 로직이 탑재된 최종 코드입니다.

Python

```
"""
[System Header]
Project: PDF Indexer v42.1 Zero-Dependency Edition
Description: Installer uses ONLY standard libraries. Runner has full features.
Integrity: Solves 'No Response' issue by removing premature imports.

================================================================================
================
[SYSTEM CONSTITUTION: 프로그램 구성 원칙]
4. 구성
* 프로그램 구성
- **프로그램 코드에는 '4. 구성'의 모든 내용이 코드 최상단에 전체 내용 기입 + 관련 코드 항목에 각각 코멘트 기입 필요 : 최종 : 최상단에 구성 전체 내용 기입 + 각 항목별 코멘트 기입 (다른 ai 혹은 프로그래머가 코드만 보고 코드 작성 가이드를 따를 수 있어야 함)**
- **사용자는 설치 파일만 실행, 나머지 설치 과정은 파이썬 코드가 자동으로 진행**
- **'설치 GUI', '실행 GUI' 로 구분**
- **모든 과정은 GUI 를 적용하여 .pyw 로 cmd 창 없이 GUI로 진행**
- **'설치 GUI' 파일을 실행하면, [폴더 생성 -> 파일 생성 -> 필요 라이브러리 자동 다운로드 및 설치] 까지 원클릭 진행**
- **설치가 완료 후, 설치된 '실행 GUI' 파일로 프로그램을 실행**
- **소스 코드는 반드시 영어(English)로 작성하여 코드 호환성 유지**
- **GUI 출력(라벨, 버튼, 알림창)은 반드시 한국어(Korean)로 작성하여 사용자 편의성 극대화**
- **역할, 파트 별로 파일을 나누어서 1개의 폴더 안에 저장**

* '설치 GUI'
- **라이브러리가 설치되는 과정을 검은색 CMD 창이 아닌, '설치 GUI' 로딩 창(Progress Bar)으로 표현**
- **모든 설치 과정은 항목 별로 총 작업량 대비 현재 작업량을 퍼센티지화 하여 진척률을 확인할 수 있는 시각적 이미지 적용**
```

- **설치 진행률은 전체 항목과 현재 항목을 구분하여 Progress Bar 를 각각 적용**
- **설치 폴더를 '설치 GUI' 안의 버튼을 이용하여 사용자가 지정**
- **결과 파일은 지정 된 설치 폴더에 하위폴더 없이 저장**

* 실행 GUI
- **PDF 변환이 진행되는 과정을 검은색 CMD 창이 아닌, '실행 GUI' 로딩 창(Progress Bar)으로 표현**
- **PDF 파일 업로드는 '실행 GUI' 안의 버튼을 이용하여 사용자가 적용**
- **작업 과정은 항목 별로 총 작업량 대비 현재 작업량을 퍼센티지화 하여 진척률을 확인할 수 있는 시각적 이미지 적용**
- **작업 진행률은 전체 항목과 현재 항목을 구분하여 Progress Bar 를 각각 적용**
- **원본 파일을 불러오는 폴더를 '실행 GUI' 상에서 사용자가 지정**
- **프로그램을 제작하는 과정에서 누적된 패치 및 업데이트 사항에 대한 모든 내용을 누락없이 표시 (시스템 개발 역사 페이지는 버전별로 폴더형식으로 눌러서 열어볼 수 있또록 누락없이 모든 내용 기입,핵심 요약이 아니라 관련 내용이 자세하게 기입**
==========
"""
```
# [IMPORTANT] Installer Imports MUST remain standard library only
import sys
import ctypes
import os
import subprocess
import threading
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import time

#
==========
=
# [PART 1] 실행 GUI 소스코드 (RUNNER_SOURCE)
# This string contains the full application code which WILL use external libraries.
#
==========
=
RUNNER_SOURCE = r'''
import os
import sys
import time
import re
import threading
import traceback
import logging
import gc
import hashlib
import csv
```

```
import shutil
import json
import tkinter as tk
import ctypes
import psutil
from logging.handlers import RotatingFileHandler
from tkinter import ttk, filedialog, messagebox, scrolledtext
from datetime import datetime, timedelta
from concurrent.futures import ThreadPoolExecutor

# Windows Power Management Constants
ES_CONTINUOUS = 0x80000000
ES_SYSTEM_REQUIRED = 0x00000001

# - **프로그램을 제작하는 과정에서 누적된 패치 및 업데이트 사항에 대한 모든 내용을 누락없이 표시**
HISTORY_TEXT = """
[v42.1] 제로 디펜던시
 - 설치 프로그램 무반응 해결: 설치 파일의 외부 라이브러리 의존성 제거.

[v42.0] 불사조(Phoenix)
 - 토큰 기반 복구 시스템 도입: 'recovery.token' 파일을 이용해 정상 실행과 재난 복구를 구분.
 - 스마트 재시작: 사용자가 직접 켠 경우 대기, 작업 중 튕긴 경우에만 자동 시작.

[v41.1] 자율 구동
 - 자동 작업 재개: 재시작 옵션 활성화 시 자동 작업 수행.
 - 완료 시 자동 OFF: 전체 작업 완료 후 재시작 옵션 자동 해제.

[v41.0] 절대 주권
 - 자동 재시작(좀비 모드) ON/OFF 스위치 탑재.

[v40.5] 최종 설계자
 - 스마트 종료 로직 적용: 사용자가 끄면 꺼지고, 오류로 죽으면 되살아남.
 - 히스토리 뷰어 개선: 스크롤 가능한 텍스트 박스로 변경하여 가독성 확보.

[v40.4] 완벽 수정
 - 구동부 버그 픽스: 실행 프로그램 내부 클래스 호출 오류 수정.

[v40.3] 침묵의 마스터
 - 무중단 히든 모드: 실행 시 CMD 검은 창이 뜨지 않도록 VBScript 런처 적용.
 - 구성 원칙 통합: 소스 코드 최상단에 개발 가이드라인 명시.

[v40.2] 안정성 강화
 - 엔진 주기적 재장전: 50개 파일 처리 시마다 AI 엔진 리로드(Reload)로 VRAM 누수 차단.

[v40.1] 코드 규약
 - 구성 원칙 명시: 향후 유지보수를 위해 코드 내 가이드 주석 완비.

[v39.1] 완전 건너뛰기
```

- 스킵 로직 고도화: 성공한 파일 + 격리된(Quarantined) 파일 모두 검사하여 건너뜀.

[v39.0] 기억의 궁전
- 설정 영속성: 폴더 경로 및 옵션값 자동 저장/복원 (JSON).

[v38.1] 무한 루프
- 생존성 극대화: 크래시 발생 시 5초 후 자동 부활.
- 메모리 감시: RAM 90% 초과 시 선제적 안전 재시작.

[v38.0] 문서화
- 코드 레벨 구성 가이드 주석 탑재.

[v37.0] 멀티 코어
- 고속 병렬 처리(2-Threads) 옵션 탑재.

[v36.0] 고밀도 스캔
- 고밀도 스캔(Scale 3.0) 적용 & 설치 에러 자동 우회.

[v35.2] 강제 정찰
- 강제 OCR 옵션 추가 (텍스트 레이어 무시하고 이미지로 판독).

[v35.1] 긴급 수정
- 엔진 로딩 변수명 오류 수정.

[v35.0] 통제권
- 작업 중단(Stop) 버튼 복구.

[v34.5] 정밀 타격
- 스마트 스킵 적용 (이미 있는 파일 로딩 없이 건너뜀).

[v34.1] 궁극의 GUI
- CMD 제거 & 듀얼 프로그래스 바 도입.

[v33.0] 안전 래퍼
- 실행 무반응 해결을 위한 자가 진단 래퍼.

[v31.3] 이름 정리
- 파일명 해시 제거, 원본 파일명 유지.

[v31.0] 텍스트 코어
- 이미지 저장 제거로 속도 500% 향상.

[v30.0] 격리 구역
- 독립 가상환경(VENV) 구축.

[v24.0] 자동 방역
- 불량 데이터 자동 격리.

```
[v10.0] CUDA 엔진
 - NVIDIA GPU 가속 엔진 탑재.
"""

class IORedirector(object):
    def __init__(self, text_widget, root):
        self.text_widget = text_widget
        self.root = root
    def write(self, str_msg):
        self.root.after(0, lambda: self.append_text(str_msg))
    def flush(self): pass
    def append_text(self, str_msg):
        try:
            self.text_widget.config(state="normal")
            if int(self.text_widget.index("end-1c").split(".")[0]) > 3000:
                self.text_widget.delete("1.0", "500.0")
                str_msg = "\n[Logs Cleaned]\n" + str_msg
            self.text_widget.insert("end", str_msg)
            self.text_widget.see("end")
            self.text_widget.config(state="disabled")
        except: pass

class PDFIndexerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("기구 설계 PDF 데이터 구축기 (v42.1 Zero-Dependency)")
        self.root.geometry("1400x950")

        self.root.protocol("WM_DELETE_WINDOW", self.on_closing)

        self.input_dir = tk.StringVar()
        self.output_dir = tk.StringVar()

        # Options
        self.var_keep_hanja = tk.BooleanVar(value=False)
        self.var_force_ocr = tk.BooleanVar(value=False)
        self.var_parallel = tk.BooleanVar(value=False)
        self.var_auto_restart = tk.BooleanVar(value=True)

        self.is_running = False
        self.stop_event = threading.Event()
        self.converter = None
        self.logger = logging.getLogger("BlackBox")
        self.logger.setLevel(logging.INFO)

        self.processed_count = 0
        self.total_files = 0
        self.config_file = "indexer_config.json"
```

```
        # [v42.0] Token File
        self.token_file = "recovery.token"

        self.setup_ui()
        self.load_config()
        self.update_restart_flag()
        sys.stdout = IORedirector(self.txt_log, self.root)
        sys.stderr = IORedirector(self.txt_log, self.root)

        # [v42.0] Token Check
        if self.var_auto_restart.get() and os.path.exists(self.token_file):
            if self.input_dir.get() and self.output_dir.get():
                self.log_gui("🔥 비정상 종료(Crash) 감지됨. 3초 후 자동 복구 시작...")
                self.root.after(3000, self.auto_start_trigger)
            else:
                self.cleanup_token()
        else:
            self.cleanup_token()

    def auto_start_trigger(self):
        if not self.is_running:
            self.toggle_processing()

    def create_token(self):
        try:
            with open(self.token_file, "w") as f: f.write("active")
        except: pass

    def cleanup_token(self):
        try:
            if os.path.exists(self.token_file):
                os.remove(self.token_file)
        except: pass

    def on_closing(self):
        if self.is_running:
            if not messagebox.askyesno("종료", "작업이 진행 중입니다. 정말 종료하시겠습니까?"):
                return
        self.cleanup_token() # Clean exit
        self.root.destroy()
        sys.exit(0)

    def log_gui(self, message):
        timestamp = datetime.now().strftime('%H:%M:%S')
        full_msg = f"[{timestamp}] {message}"
        print(full_msg)
        if hasattr(self, 'logger'):
```

```
        try: self.logger.info(message)
        except: pass

    def load_config(self):
        try:
            if os.path.exists(self.config_file):
                with open(self.config_file, 'r', encoding='utf-8') as f:
                    data = json.load(f)
                    self.input_dir.set(data.get("input_dir", ""))
                    self.output_dir.set(data.get("output_dir", ""))
                    self.var_keep_hanja.set(data.get("keep_hanja", False))
                    self.var_force_ocr.set(data.get("force_ocr", False))
                    self.var_parallel.set(data.get("parallel", False))
                    self.var_auto_restart.set(data.get("auto_restart", True))
                self.log_gui("📂 이전 설정을 불러왔습니다.")
        except Exception as e:
            self.log_gui(f"⚠️ 설정 로드 실패: {e}")

    def save_config(self):
        try:
            data = {
                "input_dir": self.input_dir.get(),
                "output_dir": self.output_dir.get(),
                "keep_hanja": self.var_keep_hanja.get(),
                "force_ocr": self.var_force_ocr.get(),
                "parallel": self.var_parallel.get(),
                "auto_restart": self.var_auto_restart.get()
            }
            with open(self.config_file, 'w', encoding='utf-8') as f:
                json.dump(data, f, indent=4, ensure_ascii=False)
        except Exception as e:
            self.log_gui(f"⚠️ 설정 저장 실패: {e}")

    def update_restart_flag(self):
        # We handle restart logic via Token now, but keeping flag file for BAT just in case
        pass

    def setup_ui(self):
        paned = tk.PanedWindow(self.root, orient=tk.HORIZONTAL, sashwidth=5, bg="#bdc3c7")
        paned.pack(fill=tk.BOTH, expand=True)

        left = tk.Frame(paned, bg="#fdfefe", padx=20, pady=20)
        paned.add(left, minsize=600)

        header = tk.Frame(left, bg="#fdfefe")
        header.pack(fill="x", pady=(0, 20))
        tk.Label(header, text="PDF 변환 컨트롤러", font=("Malgun Gothic", 20, "bold"), bg="#fdfefe",
fg="#2c3e50").pack(side="left")
```

```
        self.lbl_gpu = tk.Label(header, text="GPU: 확인 중...", font=("Malgun Gothic", 10, "bold"),
bg="#ecf0f1", fg="#7f8c8d", padx=10, pady=5, relief="ridge")
        self.lbl_gpu.pack(side="right")

        tk.Label(left, text="1. 원본 PDF 폴더:", font=("Malgun Gothic", 10, "bold"),
bg="#fdfefe").pack(anchor="w")
        f1 = tk.Frame(left, bg="#fdfefe")
        f1.pack(fill="x", pady=5)
        tk.Entry(f1, textvariable=self.input_dir, state="readonly", bg="white").pack(side="left", fill="x",
expand=True, ipady=3)
        tk.Button(f1, text="폴더 찾기", command=lambda: self.input_dir.set(filedialog.askdirectory()),
bg="#ecf0f1").pack(side="right")

        tk.Label(left, text="2. 데이터 저장 폴더:", font=("Malgun Gothic", 10, "bold"),
bg="#fdfefe").pack(anchor="w")
        f2 = tk.Frame(left, bg="#fdfefe")
        f2.pack(fill="x", pady=5)
        tk.Entry(f2, textvariable=self.output_dir, state="readonly", bg="white").pack(side="left", fill="x",
expand=True, ipady=3)
        tk.Button(f2, text="폴더 찾기", command=lambda: self.output_dir.set(filedialog.askdirectory()),
bg="#ecf0f1").pack(side="right")

        opt = tk.LabelFrame(left, text="설정 (Settings)", bg="#fdfefe", padx=10, pady=5)
        opt.pack(fill="x", pady=10)
        tk.Checkbutton(opt, text="일본어/한자 유지", variable=self.var_keep_hanja, bg="#fdfefe",
font=("Malgun Gothic", 9)).pack(anchor="w")
        tk.Checkbutton(opt, text="고밀도 정밀 분석 (강제 OCR) - 품질↑ 속도↓", variable=self.var_force_ocr,
bg="#fdfefe", fg="#c0392b", font=("Malgun Gothic", 9, "bold")).pack(anchor="w")
        tk.Checkbutton(opt, text="고속 병렬 처리 (2-Threads) - 속도↑ 메모리↑", variable=self.var_parallel,
bg="#fdfefe", fg="#2980b9", font=("Malgun Gothic", 9, "bold")).pack(anchor="w")

        tk.Checkbutton(opt, text="오류 발생 시 자동 재시작 (Auto-Recovery)",
variable=self.var_auto_restart, bg="#fdfefe", fg="#e67e22", font=("Malgun Gothic", 9,
"bold")).pack(anchor="w")

        st = tk.LabelFrame(left, text="작업 대시보드", bg="#fdfefe", padx=10, pady=5)
        st.pack(fill="x", pady=10)

        tk.Label(st, text="전체 진행률 (Total)", font=("Malgun Gothic", 9), bg="#fdfefe").pack(anchor="w")
        self.pb_total = ttk.Progressbar(st, orient="horizontal", length=100, mode="determinate")
        self.pb_total.pack(fill="x", pady=(0, 5))

        tk.Label(st, text="현재 파일 (Current Batch)", font=("Malgun Gothic", 9),
bg="#fdfefe").pack(anchor="w")
        self.pb_current = ttk.Progressbar(st, orient="horizontal", length=100, mode="determinate")
        self.pb_current.pack(fill="x", pady=(0, 5))

        self.lbl_main_status = tk.Label(st, text="대기 중...", font=("Malgun Gothic", 11, "bold"),
```

```
fg="#2c3e50", bg="#fdfefe", anchor="w")
        self.lbl_main_status.pack(fill="x", pady=5)
        self.lbl_detail_status = tk.Label(st, text="-", font=("Malgun Gothic", 9), fg="#7f8c8d", bg="#fdfefe",
anchor="w")
        self.lbl_detail_status.pack(fill="x")

        tk.Label(left, text="실시간 로그:", font=("Malgun Gothic", 9, "bold"),
bg="#fdfefe").pack(anchor="w")
        self.txt_log = tk.Text(left, height=12, state="disabled", font=("Consolas", 9), bg="#f4f6f7")
        self.txt_log.pack(fill="both", expand=True, pady=5)

        self.btn_start = tk.Button(left, text="변환 시작 (Start)", command=self.toggle_processing,
bg="#27ae60", fg="white", font=("Malgun Gothic", 14, "bold"), height=2)
        self.btn_start.pack(fill="x", pady=10)

        right = tk.Frame(paned, bg="#ecf0f1", padx=15, pady=15)
        paned.add(right, minsize=450)

        tk.Label(right, text="📘 시스템 개발 역사 (History)", font=("Malgun Gothic", 14, "bold"),
bg="#ecf0f1", fg="#27ae60").pack(anchor="w", pady=(0, 10))

        self.txt_history = scrolledtext.ScrolledText(right, font=("Malgun Gothic", 9), bg="white",
relief="flat")
        self.txt_history.pack(fill="both", expand=True, pady=(0, 10))
        self.txt_history.insert("end", HISTORY_TEXT)
        self.txt_history.config(state="disabled")

        tk.Label(right, text="❌ 격리/실패 목록", font=("Malgun Gothic", 12, "bold"), bg="#ecf0f1",
fg="#c62828").pack(anchor="w", pady=(10, 5))
        self.lst_errors = tk.Listbox(right, font=("Consolas", 9), bg="white", fg="#c62828")
        self.lst_errors.pack(fill="both", expand=True)

    def add_error_log(self, filename, reason):
        msg = f"[{datetime.now().strftime('%H:%M')}] {filename} : {reason}"
        self.root.after(0, lambda: self.lst_errors.insert("end", msg))
        self.root.after(0, lambda: self.lst_errors.see("end"))

    def prevent_sleep(self):
        try: ctypes.windll.kernel32.SetThreadExecutionState(ES_CONTINUOUS | ES_SYSTEM_REQUIRED)
        except: pass

    def allow_sleep(self):
        try: ctypes.windll.kernel32.SetThreadExecutionState(ES_CONTINUOUS)
        except: pass

    def check_memory(self):
        mem = psutil.virtual_memory()
        if mem.percent > 90:
```

```
            self.log_gui(f"⚠ 경고: 메모리 사용량 {mem.percent}% 초과. 안전을 위해 재시작을 권장합니다.")
            return False
        return True

    def get_path_hash(self, full_path):
        return hashlib.sha256(full_path.encode('utf-8', 'ignore')).hexdigest()[:8]

    def sanitize_filename(self, filename):
        name, ext = os.path.splitext(filename)
        safe = re.sub(r'[^\w\-\.]', '_', name)
        return safe + ext

    def clean_text(self, text):
        if not text: return ""
        if not self.var_keep_hanja.get():
            text = re.sub(r'[\u4e00-\u9fff\u3040-\u309f\u30a0-\u30ff]', '', text)
        return text.strip()

    def check_capacity_safe(self, input_path, output_path):
        try:
            total_input = 0
            for r, d, f in os.walk(input_path):
                for file in f:
                    if file.lower().endswith('.pdf'): total_input += os.path.getsize(os.path.join(r, file))
            total, used, free = shutil.disk_usage(output_path)
            req = total_input * 1.2
            if free < req: return False, f"용량 부족! (필요: {req/1024**3:.2f}GB)"
            return True, "OK"
        except: return True, "Skip"

    def write_atomic(self, file_path, content):
        temp_path = file_path + ".tmp"
        try:
            with open(temp_path, "w", encoding="utf-8") as f: f.write(content)
            for i in range(3):
                try:
                    if os.path.exists(file_path): os.remove(file_path)
                    os.rename(temp_path, file_path)
                    return True
                except OSError: time.sleep(0.5)
            return True
        except: return False

    def append_manifest(self, filename, file_hash, status, text_len, note=""):
        try:
            csv_path = os.path.join(self.output_dir.get(), "report_manifest.csv")
            is_new = not os.path.exists(csv_path)
            with open(csv_path, "a", newline="", encoding="utf-8-sig") as f:
```

```
            w = csv.writer(f)
            if is_new: w.writerow(["Timestamp", "Filename", "Hash", "Status", "Text_Length", "Note"])
            w.writerow([datetime.now().strftime("%Y-%m-%d %H:%M:%S"), filename, file_hash, status,
text_len, note])
        except: pass

    def toggle_processing(self):
        if self.is_running:
            if messagebox.askyesno("확인", "작업을 중단하시겠습니까?"):
                self.stop_event.set()
                self.cleanup_token()
                self.btn_start.config(text="중단 요청됨...", state="disabled")
            return

        if not self.input_dir.get() or not self.output_dir.get():
            messagebox.showwarning("경고", "폴더를 선택하세요.")
            return

        ok, msg = self.check_capacity_safe(self.input_dir.get(), self.output_dir.get())
        if not ok and not messagebox.askyesno("경고", f"{msg}\n진행합니까?"): return

        self.save_config()
        self.create_token()

        self.is_running = True
        self.btn_start.config(state="normal", text="작업 중단 (Stop)", bg="#e74c3c")
        self.stop_event.clear()

        try:
            lf = os.path.join(self.output_dir.get(), "system_debug.log")
            fh = RotatingFileHandler(lf, maxBytes=5*1024*1024, backupCount=5, encoding="utf-8")
            fh.setFormatter(logging.Formatter('%(asctime)s [%(levelname)s] %(message)s'))
            self.logger.handlers = []
            self.logger.addHandler(fh)
        except: pass

        t = threading.Thread(target=self.process_manager)
        t.daemon = True
        t.start()

    def process_manager(self):
        self.prevent_sleep()
        logging.getLogger("docling").setLevel(logging.WARNING)

        if self.converter is None:
            self.log_gui("🚀 엔진 로딩 시작... (모델 다운로드)")
            self.root.after(0, lambda: self.pb_current.configure(mode="indeterminate"))
            self.root.after(0, lambda: self.pb_current.start(10))
```

```
        try:
            import torch
            if torch.cuda.is_available():
                g = torch.cuda.get_device_name(0)
                self.root.after(0, lambda: self.lbl_gpu.config(text=f"🟢 GPU Active: {g}", fg="green"))
                self.log_gui(f"★ GPU 가속 활성화: {g}")
            else:
                self.root.after(0, lambda: self.lbl_gpu.config(text="🔴 CPU Mode", fg="red"))

            from docling.document_converter import DocumentConverter, PdfFormatOption
            from docling.datamodel.pipeline_options import PdfPipelineOptions, TableFormerMode
            from docling.datamodel.base_models import InputFormat

            ops = PdfPipelineOptions()
            ops.do_ocr = True
            ops.do_table_structure = True
            ops.table_structure_options.mode = TableFormerMode.ACCURATE
            ops.generate_picture_images = False

            if self.var_force_ocr.get():
                self.log_gui("⚠️ [고밀도 스캔] 활성화: Scale 3.0 적용 (속도 느림)")
                ops.images_scale = 3.0

            self.converter = DocumentConverter(
                format_options={InputFormat.PDF: PdfFormatOption(pipeline_options=ops)}
            )
            self.log_gui("✅ 엔진 준비 완료.")
        except Exception as e:
            self.log_gui(f"❌ 엔진 에러: {e}")
            self.is_running = False
            self.cleanup_token()
            self.root.after(0, self.reset_ui)
            self.allow_sleep()
            return
        finally:
            self.root.after(0, lambda: self.pb_current.stop())
            self.root.after(0, lambda: self.pb_current.configure(mode="determinate"))

    pdf_files = []
    for r, d, f in os.walk(self.input_dir.get()):
        for file in f:
            if file.lower().endswith(".pdf"): pdf_files.append(os.path.join(r, file))

    self.total_files = len(pdf_files)
    self.processed_count = 0
    self.log_gui(f"총 {self.total_files}개 파일 발견.")

    q_dir = os.path.join(self.output_dir.get(), "_QUARANTINE_LOW_QUALITY")
```

```
        os.makedirs(q_dir, exist_ok=True)
        self.start_t = time.time()

        if self.var_parallel.get():
            self.log_gui("⚡ 고속 병렬 처리(2-Threads) 모드로 시작합니다.")
            with ThreadPoolExecutor(max_workers=2) as executor:
                futures = []
                for i, full_path in enumerate(pdf_files):
                    if self.stop_event.is_set(): break
                    if i > 0 and i % 50 == 0:
                        self.log_gui("♻️ 병렬 처리 중 엔진 최적화 대기...")
                        time.sleep(2)
                    futures.append(executor.submit(self.process_single_file, full_path, q_dir))

                for f in futures:
                    if self.stop_event.is_set(): break
                    try: f.result()
                    except: pass
        else:
            self.log_gui("🐢 일반 처리(1-Thread) 모드로 시작합니다.")
            for i, full_path in enumerate(pdf_files):
                if self.stop_event.is_set(): break
                if i > 0 and i % 50 == 0:
                    self.log_gui("♻️ 엔진 리로드 (VRAM 초기화)...")
                    self.converter = None
                    gc.collect()
                    try:
                        import torch
                        torch.cuda.empty_cache()
                    except: pass
                    self.load_converter()
                self.process_single_file(full_path, q_dir)

        if not self.stop_event.is_set():
            self.log_gui("🎉 모든 작업이 완료되었습니다. 자동 재시작 기능을 해제합니다.")
            self.var_auto_restart.set(False)
            self.save_config()
            self.cleanup_token()
            messagebox.showinfo("완료", "모든 작업이 끝났습니다.")
        else:
            self.log_gui("⛔ 작업이 사용자에 의해 중단되었습니다.")
            self.cleanup_token()
            messagebox.showinfo("중단", "작업이 중단되었습니다.")

        self.allow_sleep()
        self.root.after(0, self.reset_ui)

    def load_converter(self):
```

```
        try:
            from docling.document_converter import DocumentConverter, PdfFormatOption
            from docling.datamodel.pipeline_options import PdfPipelineOptions, TableFormerMode
            from docling.datamodel.base_models import InputFormat
            ops = PdfPipelineOptions()
            ops.do_ocr = True
            ops.do_table_structure = True
            ops.table_structure_options.mode = TableFormerMode.ACCURATE
            ops.generate_picture_images = False
            if self.var_force_ocr.get():
                ops.images_scale = 3.0
            self.converter = DocumentConverter(
                format_options={InputFormat.PDF: PdfFormatOption(pipeline_options=ops)}
            )
            return True
        except: return False

    def process_single_file(self, full_path, q_dir):
        if self.stop_event.is_set(): return
        if not self.check_memory(): pass

        fname = os.path.basename(full_path)
        phash = self.get_path_hash(full_path)
        clean_name = self.sanitize_filename(os.path.splitext(fname)[0])
        md_path = os.path.join(self.output_dir.get(), f"{clean_name}.md")
        q_path = os.path.join(q_dir, f"{clean_name}.md")

        exists_main = os.path.exists(md_path) and os.path.getsize(md_path) > 0
        exists_q = os.path.exists(q_path) and os.path.getsize(q_path) > 0

        if exists_main or exists_q:
            reason = "변환 완료" if exists_main else "격리됨"
            self.log_gui(f"[패스] {fname} ({reason})")
            self.update_progress(fname, skipped=True)
            return

        self.root.after(0, lambda: self.lbl_detail_status.config(text=f"처리 중: {fname}"))

        try:
            res = self.converter.convert(full_path)
            doc = res.document
            content = []
            for item, _ in doc.iterate_items():
                if hasattr(item, "text") and item.text:
                    text = self.clean_text(item.text)
                    if text: content.append(text)
                if hasattr(item, "export_to_markdown"):
                    try:
```

```
                    tbl = item.export_to_markdown()
                    if tbl and "|" in tbl: content.append(tbl)
                except: pass

            final_text = "\n\n".join(content)
            tlen = len(final_text)
            fsize = os.path.getsize(full_path) / (1024*1024)

            target_file = md_path
            if (fsize > 5 and tlen < 200) or (tlen < 50):
                target_file = os.path.join(q_dir, f"{clean_name}.md")
                self.log_gui(f"[격리] {fname} (Len:{tlen})")
                self.add_error_log(fname, f"Quarantined (Len:{tlen})")
                self.append_manifest(fname, phash, "QUARANTINED", tlen, "Low Density")
            else:
                self.log_gui(f"[완료] {fname}")
                self.append_manifest(fname, phash, "SUCCESS", tlen, "")

            self.write_atomic(target_file, final_text)

            import torch
            if torch.cuda.is_available(): torch.cuda.empty_cache()

        except Exception as e:
            self.log_gui(f"[실패] {fname}")
            self.add_error_log(fname, str(e))
            self.append_manifest(fname, phash, "ERROR", 0, str(e))
            if os.path.exists(md_path + ".tmp"):
                try: os.remove(md_path + ".tmp")
                except: pass

        self.update_progress(fname)

    def update_progress(self, fname, skipped=False):
        self.processed_count += 1
        pct = (self.processed_count / self.total_files) * 100
        self.root.after(0, lambda: self.pb_total.configure(value=pct))

        elapsed = time.time() - self.start_t
        if self.processed_count > 0:
            avg = elapsed / self.processed_count
            rem = int(avg * (self.total_files - self.processed_count))
            eta = str(timedelta(seconds=rem))
            status_txt = f"처리 중 ({self.processed_count}/{self.total_files}): {fname} (ETA: {eta})"
            if skipped: status_txt = f"건너뜀 (Skipped): {fname} (ETA: {eta})"
            self.root.after(0, lambda: self.lbl_main_status.config(text=status_txt))

    def reset_ui(self):
```

```
        self.is_running = False
        self.btn_start.config(text="변환 시작 (Start)", state="normal", bg="#27ae60")
        self.lbl_main_status.config(text="작업 대기 중")
        self.lbl_detail_status.config(text="-")
        self.pb_total.configure(value=0)
        self.pb_current.configure(value=0)

if __name__ == "__main__":
    try:
        root = tk.Tk()
        app = PDFIndexerApp(root)
        root.mainloop()
    except Exception as e:
        ctypes.windll.user32.MessageBoxW(0, f"Error: {str(e)}", "Critical", 0x10)
'''

#
==========================================================================
=
# [PART 2] 설치 GUI (InstallerGUI)
# - **'설치 GUI', '실행 GUI' 로 구분**
#
==========================================================================
=
class InstallerGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("설치 매니저 (v42.1 Zero-Dependency)")
        self.root.geometry("600x500")
        self.root.configure(bg="#f5f6fa")

        desktop = os.path.join(os.path.join(os.environ['USERPROFILE']), 'Desktop')
        self.install_dir = tk.StringVar(value=os.path.join(desktop, "PDF_Indexer_v42"))

        self.setup_ui()

    def setup_ui(self):
        header = tk.Frame(self.root, bg="#2c3e50", pady=15)
        header.pack(fill="x")
        # - **GUI 출력(라벨, 버튼, 알림창)은 반드시 한국어(Korean)로 작성하여 사용자 편의성 극대화**
        tk.Label(header, text="기구 설계 PDF 구축기 설치", font=("Malgun Gothic", 18, "bold"),
fg="white", bg="#2c3e50").pack()

        body = tk.Frame(self.root, bg="#f5f6fa", padx=30, pady=20)
        body.pack(fill="both", expand=True)
```

```
        # - **설치 폴더를 '설치 GUI' 안의 버튼을 이용하여 사용자가 지정**
        tk.Label(body, text="설치 경로 지정:", font=("Malgun Gothic", 11, "bold"),
bg="#f5f6fa").pack(anchor="w")
        f_dir = tk.Frame(body, bg="#f5f6fa")
        f_dir.pack(fill="x", pady=(5, 20))
        tk.Entry(f_dir, textvariable=self.install_dir, font=("Malgun Gothic", 10)).pack(side="left",
fill="x", expand=True, ipady=5)
        tk.Button(f_dir, text="변경", command=self.change_dir, bg="#dcdde1", font=("Malgun
Gothic", 10)).pack(side="right", padx=(10, 0))

        # - **설치 진행률은 전체 항목과 현재 항목을 구분하여 Progress Bar 를 각각 적용**
        # - **모든 설치 과정은 항목 별로 총 작업량 대비 현재 작업량을 퍼센티지화 하여 진척률을 확인할 수
있는 시각적 이미지 적용**
        tk.Label(body, text="전체 설치 진행률:", font=("Malgun Gothic", 10),
bg="#f5f6fa").pack(anchor="w")
        self.pb_total = ttk.Progressbar(body, orient="horizontal", length=100, mode="determinate")
        self.pb_total.pack(fill="x", pady=(5, 15))

        tk.Label(body, text="현재 항목 진행률:", font=("Malgun Gothic", 10),
bg="#f5f6fa").pack(anchor="w")
        self.pb_current = ttk.Progressbar(body, orient="horizontal", length=100,
mode="determinate")
        self.pb_current.pack(fill="x", pady=(5, 5))

        self.lbl_status = tk.Label(body, text="설치 준비 완료", font=("Malgun Gothic", 10, "bold"),
fg="#2980b9", bg="#f5f6fa")
        self.lbl_status.pack(pady=10)

        # - **'설치 GUI' 파일을 실행하면, [폴더 생성 -> 파일 생성 -> 필요 라이브러리 자동 다운로드 및 설치]
까지 원클릭 진행**
        self.btn_install = tk.Button(body, text="설치 시작 (Install)", command=self.start_install,
bg="#27ae60", fg="white", font=("Malgun Gothic", 14, "bold"), relief="flat", height=2)
        self.btn_install.pack(fill="x", pady=10)

    def change_dir(self):
        d = filedialog.askdirectory()
        if d: self.install_dir.set(os.path.join(d, "PDF_Indexer_v42"))

    def start_install(self):
        self.btn_install.config(state="disabled", text="설치 진행 중...")
        threading.Thread(target=self.run_install, daemon=True).start()
```

```
    def run_install(self):
        try:
            target = self.install_dir.get()
            venv = os.path.join(target, "venv")
            python_exe = sys.executable

            # 1. Folder Creation
            self.update_status(0, 0, "폴더 생성 중...")
            os.makedirs(target, exist_ok=True)
            self.update_status(10, 100, "폴더 생성 완료")
            time.sleep(0.5)

            # 2. File Creation
            self.update_status(10, 0, "실행 프로그램 생성 중...")
            # - **사용자는 설치 파일만 실행, 나머지 설치 과정은 파이썬 코드가 자동으로 진행**
            with open(os.path.join(target, "pdf_indexer_app.pyw"), "w", encoding="utf-8") as f:
                f.write(RUNNER_SOURCE)
            self.update_status(20, 100, "실행 프로그램 생성 완료")
            time.sleep(0.5)

            # 3. VENV Creation
            self.update_status(20, 0, "독립 가상환경(VENV) 구축 중...")
            proc = subprocess.Popen([python_exe, "-m", "venv", venv], shell=True)
            while proc.poll() is None:
                self.pb_current.step(2)
                time.sleep(0.1)
            self.update_status(40, 100, "가상환경 구축 완료")

            venv_python = os.path.join(venv, "Scripts", "python.exe")

            # 4. Lib Install (Safe Mode)
            # - **라이브러리가 설치되는 과정을 검은색 CMD 창이 아닌, '설치 GUI' 로딩 창(Progress Bar)으로 표현**
            self.update_status(40, 0, "PIP 업데이트 체크...")
            try:
                subprocess.run([venv_python, "-m", "pip", "install", "--upgrade", "pip"], shell=True, check=False)
            except: pass
            self.update_status(50, 100, "설정 완료 (PIP)")

            # - **모든 설치 과정은 항목 별로 총 작업량 대비 현재 작업량을 퍼센티지화 하여 진척률을 확인할 수 있는 시각적 이미지 적용**
            self.update_status(50, 0, "GPU 엔진(Torch) 및 모니터링 도구 설치 중...")
```

```
            proc = subprocess.Popen([venv_python, "-m", "pip", "install", "torch", "torchvision",
"torchaudio", "psutil", "--index-url", "https://download.pytorch.org/whl/cu121"], shell=True)
            while proc.poll() is None:
                self.pb_current.step(0.5)
                time.sleep(0.1)
            self.update_status(75, 100, "GPU 엔진 설치 완료")

            self.update_status(75, 0, "문서 처리 엔진(Docling) 설치 중...")
            subprocess.run([venv_python, "-m", "pip", "install", "docling", "pillow"], shell=True,
check=True)
            self.update_status(90, 100, "문서 엔진 설치 완료")

            # 5. Shortcut & VBS Generation (Ghost Launcher)
            self.update_status(90, 0, "무중단 런처(VBS) 생성 중...")
            venv_pyw = os.path.join(venv, "Scripts", "pythonw.exe")

            # BAT file (Logic)
            bat_content = f'''@echo off
:loop
echo [System] 프로그램을 실행합니다...
"{venv_pyw}" "{os.path.join(target, "pdf_indexer_app.pyw")}"
if %errorlevel% neq 0 (
    echo.
    echo [System] 오류로 인해 비정상 종료되었습니다. (5초 후 자동 재시작)
    timeout /t 5
    goto loop
)
echo.
echo [System] 프로그램이 정상적으로 종료되었습니다.
'''
            with open(os.path.join(target, "RUN_INDEXER.bat"), "w", encoding="cp949") as f:
                f.write(bat_content)

            # VBS file (Ghost Wrapper)
            vbs_content = f'''Set WshShell = CreateObject("WScript.Shell")
WshShell.Run chr(34) & "{os.path.join(target, "RUN_INDEXER.bat")}" & chr(34), 0
Set WshShell = Nothing
'''
            with open(os.path.join(target, "실행하기(무중단).vbs"), "w", encoding="cp949") as f:
                f.write(vbs_content)

            # - **결과 파일은 지정 된 설치 폴더에 하위폴더 없이 저장**
            self.update_status(100, 100, "설치 완료!")
```

```
            # - **설치가 완료 후, 설치된 '실행 GUI' 파일로 프로그램을 실행**
            messagebox.showinfo("설치 성공", f"설치가 완료되었습니다!\n\n[{target}]\n폴더 안의
'실행하기(무중단).vbs'를 실행하세요.")
            self.root.quit()

        except Exception as e:
            self.update_status(0, 0, "설치 실패!")
            ctypes.windll.user32.MessageBoxW(0, f"오류 발생:\n{str(e)}", "Error", 0x10)
            self.btn_install.config(state="normal", text="설치 재시도")

    def update_status(self, total_val, current_val, text):
        self.root.after(0, lambda: [
            self.pb_total.configure(value=total_val),
            self.pb_current.configure(value=current_val),
            self.lbl_status.configure(text=text)
        ])

if __name__ == "__main__":
    try:
        root = tk.Tk()
        app = InstallerGUI(root)
        root.mainloop()
    except Exception as e:
        ctypes.windll.user32.MessageBoxW(0, f"Critical Error:\n{str(e)}", "Fail", 0x10)
```