import re

import csv

import os

import random

from datetime import datetime

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.common.keys import Keys

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.common.action\_chains import ActionChains

from docx import Document

import time

from selenium.common.exceptions import (

    NoSuchElementException,

    ElementClickInterceptedException,

    TimeoutException,

    StaleElementReferenceException

)

*# Map key skills to job titles*

SKILL\_TO\_TITLES = {

    "Deep Learning": ["AI Engineer", "ML Engineer"],

    "TensorFlow": ["ML Engineer", "AI Engineer"],

    "Keras": ["AI Engineer"],

    "Pandas": ["Data Analyst", "Data Scientist"],

    "NumPy": ["Data Analyst", "ML Engineer"],

    "NLP": ["NLP Engineer", "AI Researcher"],

    "Django": ["Python Developer", "Backend Developer"],

    "Python": ["Python Developer", "Data Scientist"],

    "Machine Learning": ["ML Engineer", "Data Scientist"],

    "Artificial Intelligence": ["AI Engineer"]

}

FALLBACK\_TERMS = ["AI", "Machine Learning", "Data Science", "Software Engineer", "Developer"]

*# Extract skills from resume*

def extract\_skills(resume\_text, skills\_list):

    return [skill for skill in skills\_list if re.search(r'\b' + re.escape(skill) + r'\b', resume\_text, re.IGNORECASE)]

*# Read resume from DOCX*

def read\_resume\_skills(docx\_path):

    doc = Document(docx\_path)

    return "\n".join(para.text for para in doc.paragraphs)

*# Infer job titles from extracted skills*

def infer\_job\_titles(skills):

    titles = set()

    for skill in skills:

        if skill in SKILL\_TO\_TITLES:

            titles.update(SKILL\_TO\_TITLES[skill])

    return list(titles)

*# WebDriver setup with improved anti-detection*

def setup\_webdriver():

    options = webdriver.ChromeOptions()

    options.add\_argument("--incognito")

    options.add\_argument('--disable-blink-features=AutomationControlled')

    options.add\_argument('--disable-gpu')

    options.add\_argument('--window-size=1920,1080')

    options.add\_argument("--start-maximized")

    options.add\_argument('--user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36')

    options.add\_experimental\_option("excludeSwitches", ["enable-automation"])

    options.add\_experimental\_option('useAutomationExtension', False)

    driver = webdriver.Chrome(options=options)

*# Execute CDP commands for anti-bot evasion*

    driver.execute\_cdp\_cmd('Network.setUserAgentOverride', {"userAgent": 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36'})

    driver.execute\_script("Object.defineProperty(navigator, 'webdriver', {get: () => undefined})")

    return driver

*# Upload file with improved handling*

def upload\_file(driver, file\_path):

    try:

*# Try multiple possible file upload selectors*

        selectors = [

            "//input[@type='file']",

            "//input[contains(@accept, 'pdf') or contains(@accept, 'docx')]",

            "//button[contains(@aria-label, 'upload') or contains(@aria-label, 'Upload')]",

            "//label[contains(@for, 'file') or contains(@for, 'upload')]"

        ]

        for selector in selectors:

            try:

                elements = driver.find\_elements(By.XPATH, selector)

                for element in elements:

                    try:

                        if element.tag\_name == 'input' and element.get\_attribute('type') == 'file':

                            element.send\_keys(file\_path)

                            print(f"✓ File uploaded using {selector}")

                            time.sleep(2)

                            return True

                        elif element.is\_displayed() and element.is\_enabled():

*# If it's a button, try clicking it first to reveal the file input*

                            element.click()

                            time.sleep(1)

*# Then look for the actual file input that may have appeared*

                            file\_inputs = driver.find\_elements(By.XPATH, "//input[@type='file']")

                            if file\_inputs:

                                file\_inputs[0].send\_keys(file\_path)

                                print(f"✓ File uploaded after clicking {selector}")

                                time.sleep(2)

                                return True

                    except:

                        continue

            except:

                continue

        print("⚠️ Could not find a suitable file upload element")

        return False

    except Exception as e:

        print(f"❌ Upload error: {str(e)[:100]}")

        return False

*# Add human-like randomized delays*

def human\_delay(min\_seconds=0.5, max\_seconds=2.5):

    delay = random.uniform(min\_seconds, max\_seconds)

    time.sleep(delay)

*# Human-like typing*

def human\_type(element, text):

    for char in text:

        element.send\_keys(char)

        time.sleep(random.uniform(0.05, 0.15))

*# Auto-fill fields with improved detection*

def autofill\_fields(driver):

    try:

        field\_mapping = {

            "phone": "9004177451",

            "mobile": "9004177451",

            "city": "Mumbai",

            "location": "Mumbai",

            "postal": "500001",

            "zip": "500001",

            "website": "linkedin.com/in/yourprofile",

            "salary": "negotiable",

            "years": "2"

        }

        input\_elements = driver.find\_elements(By.TAG\_NAME, "input")

        for field in input\_elements:

            try:

                name = (field.get\_attribute("name") or "").lower()

                placeholder = (field.get\_attribute("placeholder") or "").lower()

                aria\_label = (field.get\_attribute("aria-label") or "").lower()

*# Skip if it's a file input or hidden*

                if field.get\_attribute("type") in ["file", "hidden"] or not field.is\_displayed():

                    continue

                field\_identifiers = [name, placeholder, aria\_label]

                for identifier in field\_identifiers:

                    for key, value in field\_mapping.items():

                        if key in identifier and field.is\_enabled() and field.is\_displayed():

*# Clear the field first*

                            field.clear()

*# Type like a human*

                            human\_type(field, value)

                            print(f"✓ Filled '{key}' field with '{value}'")

                            human\_delay(0.3, 1.0)

                            break

            except:

                continue

*# Handle dropdowns and select elements*

        dropdowns = driver.find\_elements(By.TAG\_NAME, "select")

        for dropdown in dropdowns:

            try:

                if dropdown.is\_displayed() and dropdown.is\_enabled():

                    dropdown.click()

                    human\_delay()

                    options = dropdown.find\_elements(By.TAG\_NAME, "option")

                    if options:

*# Select a non-default option (usually not the first one)*

                        if len(options) > 1:

                            options[1].click()

                        else:

                            options[0].click()

                    human\_delay()

            except:

                continue

*# Handle radio buttons for common questions*

        radio\_groups = driver.find\_elements(By.XPATH, "//input[@type='radio']")

        processed\_names = set()

        for radio in radio\_groups:

            try:

                name = radio.get\_attribute("name")

                if name and name not in processed\_names and radio.is\_displayed():

                    processed\_names.add(name)

                    radio.click()

                    human\_delay()

            except:

                continue

*# Handle checkboxes - often for terms agreement*

        checkboxes = driver.find\_elements(By.XPATH, "//input[@type='checkbox']")

        for checkbox in checkboxes:

            try:

                if checkbox.is\_displayed() and not checkbox.is\_selected():

*# For terms and agreements, usually we want to check these*

                    try:

                        label\_text = checkbox.find\_element(By.XPATH, "./ancestor::label").text.lower()

                    except:

                        label\_text = ""

                    if any(term in label\_text for term in ["agree", "terms", "consent", "privacy"]):

                        checkbox.click()

                        human\_delay()

            except:

                continue

    except Exception as e:

        print(f"⚠️ Autofill issue: {str(e)[:100]}")

*# Improved application attempt function*

def attempt\_application(driver, resume\_path, cover\_letter\_path):

    try:

*# Wait for the apply button*

        apply\_button\_selectors = [

            "button.jobs-apply-button",

            "button[data-control-name='jobdetails\_topcard\_inapply']",

            "button[aria-label\*='Apply']",

            "button[aria-label\*='apply']",

            "a[data-control-name='jobdetails\_topcard\_inapply']",

            "//button[contains(text(), 'Apply')]",

            "//button[contains(@class, 'jobs-apply')]",

            "//a[contains(@class, 'jobs-apply')]"

        ]

        apply\_button = None

        for selector in apply\_button\_selectors:

            try:

                if selector.startswith("//"):

                    elements = driver.find\_elements(By.XPATH, selector)

                else:

                    elements = driver.find\_elements(By.CSS\_SELECTOR, selector)

                for element in elements:

                    if element.is\_displayed() and element.is\_enabled():

                        apply\_button = element

                        break

                if apply\_button:

                    break

            except:

                continue

        if not apply\_button:

            print("⚠️ Could not find apply button with initial selectors")

*# Try scrolling down to make sure the button is in view*

            driver.execute\_script("window.scrollTo(0, document.body.scrollHeight/3);")

            time.sleep(1)

*# Try one more time with broader selectors*

            try:

                buttons = driver.find\_elements(By.TAG\_NAME, "button")

                for button in buttons:

                    try:

                        if button.is\_displayed() and button.is\_enabled():

                            text = button.text.lower()

                            if "apply" in text:

                                apply\_button = button

                                print("✓ Found apply button using text search")

                                break

                    except:

                        continue

            except:

                pass

        if not apply\_button:

            return "Failed: Apply button not found"

*# Check if we've already applied*

        try:

            button\_text = apply\_button.text.lower()

            if "applied" in button\_text or "application submitted" in button\_text:

                return "Already applied"

        except:

            pass

*# Click the apply button*

        try:

            apply\_button.click()

            print("✓ Clicked apply button")

        except ElementClickInterceptedException:

            try:

                driver.execute\_script("arguments[0].click();", apply\_button)

                print("✓ Script-clicked apply button")

            except:

*# Try another approach - scroll to button first*

                driver.execute\_script("arguments[0].scrollIntoView(true);", apply\_button)

                time.sleep(1)

                apply\_button.click()

                print("✓ Clicked apply button after scrolling")

        human\_delay(1.5, 3.0)

*# Check for the application form*

        try:

            WebDriverWait(driver, 10).until(

                EC.presence\_of\_element\_located((By.CSS\_SELECTOR, "form"))

            )

            print("✓ Application form detected")

        except TimeoutException:

            print("⚠️ Application form not found, checking for alternative elements")

*# Look for common elements in LinkedIn's application flow*

            form\_indicators = [

                "//div[contains(@class, 'jobs-easy-apply')]",

                "//div[contains(@aria-label, 'application')]",

                "//h3[contains(text(), 'application')]"

            ]

            form\_found = False

            for selector in form\_indicators:

                try:

                    if driver.find\_elements(By.XPATH, selector):

                        form\_found = True

                        print("✓ Alternative application form detected")

                        break

                except:

                    continue

            if not form\_found:

                print("⚠️ No form elements found - may be an external application")

*# Upload resume*

        if not upload\_file(driver, resume\_path):

*# If upload fails, check if resume is already uploaded/pre-filled*

            print("⚠️ Resume upload failed or not needed")

        else:

            print("✓ Resume uploaded")

        human\_delay()

*# Upload cover letter if there's a specific section for it*

        cover\_letter\_indicators = [

            "//label[contains(text(), 'Cover') or contains(text(), 'cover')]",

            "//span[contains(text(), 'Cover') or contains(text(), 'cover')]",

            "//h3[contains(text(), 'Cover') or contains(text(), 'cover')]",

            "//div[contains(text(), 'Cover') or contains(text(), 'cover')]"

        ]

        for indicator in cover\_letter\_indicators:

            try:

                if driver.find\_elements(By.XPATH, indicator):

                    upload\_file(driver, cover\_letter\_path)

                    print("✓ Cover letter uploaded")

                    break

            except:

                continue

*# Fill any form fields*

        autofill\_fields(driver)

*# Handle multi-step application process*

        attempt\_count = 0

        max\_attempts = 15  *# Increased limit for multi-step applications*

        while attempt\_count < max\_attempts:

            attempt\_count += 1

            human\_delay(1.0, 2.0)

*# Look for next/continue/submit buttons in priority order*

            button\_selectors = [

                "//button[contains(text(), 'Submit')]",

                "//button[contains(text(), 'submit')]",

                "//button[contains(text(), 'Continue')]",

                "//button[contains(text(), 'continue')]",

                "//button[contains(text(), 'Next')]",

                "//button[contains(text(), 'next')]",

                "//button[contains(@aria-label, 'Submit')]",

                "//button[contains(@aria-label, 'Continue')]",

                "//button[contains(@aria-label, 'Next')]",

                "//button[@type='submit']",

                "//footer//button"  *# LinkedIn often has buttons in the footer*

            ]

            next\_button = None

            for selector in button\_selectors:

                try:

                    buttons = driver.find\_elements(By.XPATH, selector)

                    for button in buttons:

                        try:

                            if button.is\_displayed() and button.is\_enabled():

                                button\_text = button.text.lower()

                                if any(term in button\_text for term in ["submit", "continue", "next", "apply", "save", "send"]):

                                    next\_button = button

                                    break

                        except:

                            continue

                    if next\_button:

                        break

                except:

                    continue

            if not next\_button:

*# Try one more approach - find any button in the form*

                try:

                    form\_buttons = driver.find\_elements(By.XPATH, "//form//button | //footer//button")

                    for button in form\_buttons:

                        if button.is\_displayed() and button.is\_enabled():

                            next\_button = button

                            print("✓ Found button in form/footer")

                            break

                except:

                    pass

            if not next\_button:

*# If we've gone through at least a few steps, it might be complete*

                if attempt\_count > 2:

                    print("✓ No more buttons found after multiple steps - likely completed")

                    return "Applied - multiple steps completed"

                else:

                    print("⚠️ No buttons found to continue - may be stuck or external")

                    break

*# Before clicking next, fill any new fields that might have appeared*

            autofill\_fields(driver)

*# Scroll the button into view before clicking*

            try:

                driver.execute\_script("arguments[0].scrollIntoView({block: 'center'});", next\_button)

                human\_delay(0.5, 1.0)

            except:

                pass

*# Click the button*

            try:

                button\_text = next\_button.text

                next\_button.click()

                print(f"✓ Clicked button: '{button\_text}'")

            except:

                try:

                    driver.execute\_script("arguments[0].click();", next\_button)

                    print(f"✓ Script-clicked button: '{next\_button.text}'")

                except:

                    print("⚠️ Could not click button, trying one more approach")

                    try:

*# Try using ActionChains*

                        actions = ActionChains(driver)

                        actions.move\_to\_element(next\_button).click().perform()

                        print(f"✓ Action-clicked button: '{next\_button.text}'")

                    except:

                        print("⚠️ All click methods failed, possibly stuck or completed")

                        break

*# Handle any pop-ups that might appear*

            try:

                dialogs = driver.find\_elements(By.CSS\_SELECTOR, "div[role='dialog']")

                for dialog in dialogs:

                    if dialog.is\_displayed():

                        buttons = dialog.find\_elements(By.TAG\_NAME, "button")

                        for button in buttons:

                            if any(term in button.text.lower() for term in ["submit", "continue", "next", "yes", "confirm", "ok"]):

                                button.click()

                                print("✓ Handled dialog popup")

                                human\_delay()

                                break

            except:

                pass

*# Give the page a moment to load next step*

            human\_delay(1.0, 2.0)

*# Check if we've finished successfully*

        success\_indicators = [

            "//span[contains(text(), 'Application submitted') or contains(text(), 'Applied')]",

            "//div[contains(text(), 'Application submitted') or contains(text(), 'Applied')]",

            "//h2[contains(text(), 'Application submitted') or contains(text(), 'Applied')]",

            "//p[contains(text(), 'Application submitted') or contains(text(), 'Applied')]",

            "//div[contains(text(), 'Your application was sent') or contains(text(), 'Application sent')]",

            "//div[contains(@class, 'artdeco-inline-feedback--success')]"

        ]

        for indicator in success\_indicators:

            try:

                if driver.find\_elements(By.XPATH, indicator):

                    return "Applied successfully!"

            except:

                continue

*# If we've gone through the process and don't see error messages, assume success*

        error\_indicators = [

            "//div[contains(text(), 'error')]",

            "//div[contains(@class, 'error')]",

            "//div[contains(text(), 'failed')]"

        ]

        for indicator in error\_indicators:

            try:

                if driver.find\_elements(By.XPATH, indicator):

                    return "Application may have failed"

            except:

                continue

        return "Likely applied - process completed"

    except Exception as e:

        print(f"❌ Application error: {str(e)}")

        return f"Failed: {str(e)[:100]}"

*# Search and apply logic with improved job filtering*

def process\_jobs(driver, title\_term, resume\_path, cover\_letter\_path):

    results = []

    try:

*# Go directly to LinkedIn jobs page*

        print(f"🌐 Navigating to LinkedIn Jobs page")

        driver.get("https://www.linkedin.com/jobs/")

        time.sleep(5)  *# Give it time to fully load*

*# Wait for the search input - try different possible selectors*

        search\_input = None

        search\_selectors = [

            "//input[contains(@class, 'jobs-search-box\_\_text-input')]",

            "//input[contains(@id, 'jobs-search-box')]",

            "//input[contains(@placeholder, 'Search by title')]",

            "//input[contains(@placeholder, 'Search jobs')]",

            "//form//input[@type='text']"

        ]

        for selector in search\_selectors:

            try:

                elements = driver.find\_elements(By.XPATH, selector)

                for element in elements:

                    if element.is\_displayed() and element.is\_enabled():

                        search\_input = element

                        break

                if search\_input:

                    break

            except:

                continue

        if not search\_input:

            print("❌ Could not find search input - make sure you're on the jobs page")

*# Try one more approach - find any visible input*

            try:

                inputs = driver.find\_elements(By.TAG\_NAME, "input")

                for input\_elem in inputs:

                    if input\_elem.is\_displayed() and input\_elem.get\_attribute("type") in ["text", "search"]:

                        search\_input = input\_elem

                        print("✓ Found alternative search input")

                        break

            except:

                pass

        if not search\_input:

            print("❌ Failed to find any search input - please navigate to jobs page manually")

*# Ask user to navigate and search manually*

            driver.get("https://www.linkedin.com/jobs/")

            input("Please navigate to the LinkedIn jobs page and search for jobs, then press Enter...")

            print("✓ Continuing with manual search")

        else:

*# Clear existing input and type search term*

            search\_input.clear()

            human\_type(search\_input, title\_term)

            human\_delay()

            search\_input.send\_keys(Keys.RETURN)

            print(f"🔍 Searching for: {title\_term}")

*# Wait for search results to load*

            human\_delay(5.0, 8.0)

*# Try to apply Easy Apply filter if available*

            try:

*# Click "All filters" button*

                filter\_buttons = driver.find\_elements(By.XPATH, "//button[contains(text(), 'All filters') or contains(@aria-label, 'Filter') or contains(text(), 'Filters')]")

                if filter\_buttons:

                    for button in filter\_buttons:

                        if button.is\_displayed():

                            button.click()

                            print("✓ Clicked filters button")

                            human\_delay(1.0, 2.0)

                            break

*# Look for Easy Apply checkbox*

                easy\_apply\_checkboxes = driver.find\_elements(By.XPATH, "//label[contains(text(), 'Easy Apply') or contains(text(), 'easy apply')]")

                if easy\_apply\_checkboxes:

                    for checkbox in easy\_apply\_checkboxes:

                        if checkbox.is\_displayed():

                            checkbox.click()

                            print("✓ Clicked Easy Apply filter")

                            human\_delay(1.0, 2.0)

                            break

*# Click apply/show results button*

                apply\_buttons = driver.find\_elements(By.XPATH, "//button[contains(text(), 'Show') or contains(text(), 'Apply') or contains(text(), 'Done')]")

                if apply\_buttons:

                    for button in apply\_buttons:

                        if button.is\_displayed() and button.is\_enabled():

                            button.click()

                            print("✓ Applied filters")

                            human\_delay(2.0, 3.0)

                            break

            except Exception as e:

                print(f"⚠️ Error applying filters: {str(e)[:100]}")

*# Give the page time to load results*

        print("⏳ Waiting for job listings to load...")

        time.sleep(5)

*# Try multiple approaches to find job cards*

        job\_cards = []

*# First attempt with traditional list items*

        try:

            cards = driver.find\_elements(By.CSS\_SELECTOR, "ul.jobs-search\_\_results-list li")

            if cards:

                job\_cards = cards

                print(f"✓ Found {len(cards)} job cards using method 1")

        except:

            pass

*# Second attempt with job card containers*

        if not job\_cards:

            try:

                cards = driver.find\_elements(By.XPATH, "//div[contains(@class, 'job-card-container')]")

                if cards:

                    job\_cards = cards

                    print(f"✓ Found {len(cards)} job cards using method 2")

            except:

                pass

*# Third attempt with broader search*

        if not job\_cards:

            try:

                cards = driver.find\_elements(By.XPATH, "//div[contains(@class, 'job-card') or contains(@class, 'job-search-card')]")

                if cards:

                    job\_cards = cards

                    print(f"✓ Found {len(cards)} job cards using method 3")

            except:

                pass

*# Fourth attempt - find links that might be job listings*

        if not job\_cards:

            try:

                cards = driver.find\_elements(By.XPATH, "//a[contains(@href, '/jobs/view/') or contains(@href, '/jobs/collections/')]")

                if cards:

                    job\_cards = cards

                    print(f"✓ Found {len(cards)} job cards using method 4")

            except:

                pass

*# If still no cards found, try scrolling down*

        if not job\_cards:

            print("⚠️ No job listings found, trying to scroll down...")

            for \_ in range(3):

                driver.execute\_script("window.scrollBy(0, 500)")

                time.sleep(2)

                try:

                    cards = driver.find\_elements(By.XPATH, "//div[contains(@class, 'job-card') or contains(@class, 'job-search-card') or contains(@class, 'job-result-card')]")

                    if cards:

                        job\_cards = cards

                        print(f"✓ Found {len(cards)} job cards after scrolling")

                        break

                except:

                    pass

        if not job\_cards:

            print("❌ No job listings found - LinkedIn may have changed its layout or there's an issue with the search")

*# Take a screenshot to help debug*

            try:

                driver.save\_screenshot(f"debug\_no\_jobs\_{datetime.now().strftime('%H%M%S')}.png")

                print("✓ Saved debug screenshot")

            except:

                pass

            return results

        print(f"📋 Found {len(job\_cards)} job listings")

        applied\_count = 0

        max\_applications = 8  *# Increased limit per search*

        for i, card in enumerate(job\_cards[:15]):  *# Process up to 15 listings*

            if applied\_count >= max\_applications:

                print(f"⚙️ Reached application limit ({max\_applications}) for this search")

                break

            try:

*# Extract job info before clicking*

                job\_title = ""

                company = ""

                job\_link = ""

*# Try different approaches to get job details*

                try:

                    job\_title = card.find\_element(By.CSS\_SELECTOR, "h3").text

                except:

                    try:

                        job\_title = card.find\_element(By.XPATH, ".//h3 | .//a[contains(@class, 'job-title')]").text

                    except:

                        job\_title = "Unknown Title"

                try:

                    company = card.find\_element(By.CSS\_SELECTOR, "h4").text

                except:

                    try:

                        company = card.find\_element(By.XPATH, ".//h4 | .//a[contains(@class, 'company')]").text

                    except:

                        company = "Unknown Company"

                try:

                    job\_link = card.find\_element(By.CSS\_SELECTOR, "a").get\_attribute("href")

                except:

                    try:

                        job\_link = card.get\_attribute("href")

                    except:

*# If we can't get the link, we can't process this job*

                        print(f"⚠️ Could not get link for job {i+1}")

                        continue

*# Skip if it contains sponsored or promoted*

                if "BUDGET\_EXHAUSTED\_JOB" in job\_link or any(term in job\_link for term in ["promoted", "sponsored"]):

                    results.append([job\_link, "Skipped - Promoted/Closed"])

                    print(f"⏭️ Skipping promoted/sponsored job {i+1}")

                    continue

                print(f"\n🔶 Processing job {i+1}: {job\_title} at {company}")

*# First, open in a new tab to avoid losing search results*

                driver.execute\_script(f"window.open('{job\_link}', '\_blank');")

                human\_delay()

*# Switch to the new tab*

                driver.switch\_to.window(driver.window\_handles[1])

                human\_delay(3.0, 5.0)  *# Give more time for job details page to load*

*# Attempt to apply*

                status = attempt\_application(driver, resume\_path, cover\_letter\_path)

                if "Applied" in status:

                    applied\_count += 1

                results.append([job\_link, f"{status} - {job\_title} at {company}"])

                print(f"📊 Status: {status}")

*# Close tab and switch back to results*

                driver.close()

                driver.switch\_to.window(driver.window\_handles[0])

                human\_delay(1.0, 2.0)

*# Add random delay between applications to appear more human-like*

                if i < len(job\_cards) - 1:

                    delay = random.uniform(2, 5)

                    print(f"⏱️ Waiting {delay:.1f}s before next application...")

                    time.sleep(delay)

            except Exception as e:

                print(f"❌ Error processing job: {str(e)[:100]}")

                job\_info = f"{job\_title} at {company}" if 'job\_title' in locals() and 'company' in locals() else "Unknown job"

                results.append([job\_link if 'job\_link' in locals() else "Unknown", f"Error: {str(e)[:100]} - {job\_info}"])

*# Make sure we're back on the results page*

                try:

                    if len(driver.window\_handles) > 1:

                        driver.close()

                        driver.switch\_to.window(driver.window\_handles[0])

                except:

                    pass

        return results

    except Exception as e:

        print(f"❌ Error in search process: {str(e)}")

        results.append(["Error", f"Search process error: {str(e)[:100]}"])

        return results

*# Function to check for CAPTCHA and ask for manual intervention*

def check\_for\_captcha(driver):

    captcha\_indicators = [

        "//div[contains(text(), 'captcha')]",

        "//iframe[contains(@src, 'captcha')]",

        "//div[contains(@class, 'captcha')]",

        "//h1[contains(text(), 'Security Verification')]",

        "//div[contains(text(), 'unusual activity')]",

        "//div[contains(text(), 'verification')]"

    ]

    for indicator in captcha\_indicators:

        try:

            if driver.find\_elements(By.XPATH, indicator):

                print("\n⚠️ CAPTCHA or security verification detected!")

                input("Please solve the CAPTCHA manually and press Enter when done...")

                print("✓ Continuing with job applications\n")

                return True

        except:

            continue

    return False

*# Main function with improved flow and reporting*

def main():

    print("\n" + "="\*60)

    print("🤖 Advanced LinkedIn Job Application Bot")

    print("="\*60 + "\n")

*# File paths*

    resume\_docx = "Hitesh\_CV (1) (2) (1).docx"

    resume\_pdf = os.path.abspath("Hitesh\_Resume.pdf")

    cover\_letter\_pdf = os.path.abspath("Hitesh\_Cover\_Letter.pdf")

*# Check files exist*

    for file\_path in [resume\_docx, resume\_pdf, cover\_letter\_pdf]:

        if not os.path.exists(file\_path):

            print(f"❌ ERROR: File not found: {file\_path}")

            print("Please ensure all files are in the correct location.")

            input("\nPress Enter to exit...")

            return

*# Process resume to extract skills*

    print("📄 Analyzing resume for skills...")

    SKILL\_POOL = list(SKILL\_TO\_TITLES.keys())

    resume\_text = read\_resume\_skills(resume\_docx)

    found\_skills = extract\_skills(resume\_text, SKILL\_POOL)

    print(f"✓ Found skills: {', '.join(found\_skills)}")

    inferred\_titles = infer\_job\_titles(found\_skills)

    if not inferred\_titles:

        inferred\_titles = ["Software Engineer", "Python Developer"]  *# Default if no skills matched*

    print(f"🎯 Target job titles: {', '.join(inferred\_titles)}")

*# Setup driver and login*

    print("\n🌐 Setting up browser...")

    driver = setup\_webdriver()

    driver.get("https://www.linkedin.com/login")

    print("⚠️ Please log in to LinkedIn manually when the browser opens")

    input("Press Enter once you've logged in...")

*# Check if login was successful*

    if "feed" not in driver.current\_url and "mynetwork" not in driver.current\_url:

        print("⚠️ Please navigate to LinkedIn homepage/feed after login")

        input("Press Enter once you're on the LinkedIn homepage...")

    print("\n🚀 Starting job applications...")

    all\_results = []

    timestamp = datetime.now().strftime("%Y-%m-%d\_%H-%M")

*# Process each job title with better sequencing and error handling*

    for i, title in enumerate(inferred\_titles):

        print(f"\n🔍 Searching for {title} positions...")

*# Check for CAPTCHA before each search*

        check\_for\_captcha(driver)

*# For different search terms*

        for search\_variant in [f"{title} Remote", title, f"{title} Entry Level"]:

            print(f"\n🔍 Trying search term: {search\_variant}")

            results = process\_jobs(driver, search\_variant, resume\_pdf, cover\_letter\_pdf)

            all\_results.extend(results)

*# If we got results, no need to try other variants*

            if results:

                print(f"✓ Found jobs with search term: {search\_variant}")

                break

*# Add randomized delay between searches*

            delay = random.uniform(5, 10)

            print(f"⏱️ Taking a short break ({delay:.1f}s) between searches...")

            time.sleep(delay)

*# After each job title search, take a longer break to avoid rate limiting*

        if i < len(inferred\_titles) - 1:

            delay = random.uniform(15, 25)

            print(f"⏱️ Taking a longer break ({delay:.1f}s) to avoid detection...")

            time.sleep(delay)

*# If no results, try fallback terms*

    if not all\_results:

        print("\n⚠️ No jobs found with skill-based titles. Trying fallback terms...")

        for term in FALLBACK\_TERMS:

            print(f"\n🔍 Trying fallback search: {term}")

*# Check for CAPTCHA before each search*

            check\_for\_captcha(driver)

            results = process\_jobs(driver, term, resume\_pdf, cover\_letter\_pdf)

            all\_results.extend(results)

            if results:

                print(f"✓ Found jobs with fallback term: {term}")

                break

            human\_delay(10.0, 15.0)

*# Export results and summary*

    filename = f"application\_results\_{timestamp}.csv"

    with open(filename, "w", newline="", encoding="utf-8") as f:

        writer = csv.writer(f)

        writer.writerow(["Job Link", "Status"])

        writer.writerows(all\_results)

*# Generate summary*

    applied\_count = sum(1 for result in all\_results if "Applied" in result[1])

    error\_count = sum(1 for result in all\_results if "Error" in result[1] or "Failed" in result[1])

    skipped\_count = sum(1 for result in all\_results if "Skipped" in result[1])

    print("\n" + "="\*60)

    print(f"📊 APPLICATION SUMMARY")

    print("="\*60)

    print(f"✅ Successfully applied: {applied\_count}")

    print(f"⚠️ Errors/Failures: {error\_count}")

    print(f"⏭️ Skipped jobs: {skipped\_count}")

    print(f"📝 Total jobs processed: {len(all\_results)}")

    print(f"📋 Results exported to: {filename}")

    print("="\*60)

    input("\n🏁 Process complete! Press Enter to close the browser...")

    driver.quit()

if \_\_name\_\_ == "\_\_main\_\_":

    try:

        main()

    except Exception as e:

        print(f"\n❌ CRITICAL ERROR: {str(e)}")

        print("The program encountered an unexpected error.")

        input("Press Enter to exit...")

### PATCHED BLOCK START ###  
  
# Log and Screenshot Patch in attempt\_application()  
driver.save\_screenshot("step1\_apply\_page.png")  
print("🧪 Screenshot taken: step1\_apply\_page.png")  
  
print("🔍 Searching for apply button...")  
apply\_button\_found = False  
apply\_button\_selectors = [  
 "button.jobs-apply-button",  
 "//button[contains(text(), 'Apply')]",  
 "//button[contains(@class, 'jobs-apply')]"  
]  
  
for selector in apply\_button\_selectors:  
 elements = driver.find\_elements(By.XPATH, selector) if selector.startswith("//") else driver.find\_elements(By.CSS\_SELECTOR, selector)  
 for element in elements:  
 if element.is\_displayed() and element.is\_enabled():  
 element.click()  
 print("✅ Apply button clicked.")  
 driver.save\_screenshot("step2\_clicked\_apply.png")  
 apply\_button\_found = True  
 break  
 if apply\_button\_found:  
 break  
  
if not apply\_button\_found:  
 print("⚠️ Apply button not found. Continuing anyway.")  
 driver.save\_screenshot("step2\_apply\_not\_found.png")  
  
# Forced upload and autofill (even if Apply button skipped)  
upload\_file(driver, resume\_path)  
upload\_file(driver, cover\_letter\_path)  
autofill\_fields(driver)  
  
try:  
 next\_btn = driver.find\_element(By.XPATH, "//button[contains(text(), 'Next') or contains(text(), 'Submit')]")  
 if next\_btn.is\_displayed():  
 next\_btn.click()  
 driver.save\_screenshot("step3\_submit\_clicked.png")  
 print("🚀 Submit button clicked.")  
 else:  
 print("⚠️ Submit button found but not clickable.")  
except Exception as ex:  
 print(f"⚠️ Submit step skipped: {str(ex)}")  
 driver.save\_screenshot("step3\_no\_submit.png")  
  
### PATCHED BLOCK END ###