Exportiert aus HAWKI am: 2.9.2025 von Bousfiha, Tanae

Zusammenfassung *(automatisiert erstellt)*

Das Problem liegt höchstwahrscheinlich an einem Mismatch zwischen den generierten Dateinamen im Skript (z.B. result_input.csv) und den in command.json spezifizierten Pfaden (z.B. result.csv). Der Container schreibt die Resultate zwar korrekt, aber diese werden von XNAT nicht erkannt, da die Dateinamen nicht übereinstimmen. Außerdem ist es wichtig, in der Analyse-Skriptdatei dynamisch eindeutige Dateinamen basierend auf der Eingabedatei zu verwenden, z.B. result_input.csv, und die `path`-Angabe in command.json mit Wildcards wie `'*.csv'` zu aktualisieren. Debugausgaben im Container helfen bei der Diagnose.

Systemprompt

Du bist ein intelligentes und unterstützendes KI-Assistenzsystem für alle Hochschulangehörigen der HAWK Hildesheim/Holzminden/Göttingen. Dein Ziel ist es, Studierende, Lehrende, Forschende und Mitarbeitende in ihrer akademischen Arbeit, beim Lernen, Forschen, Lehren und verwalterischen Aufgaben zu unterstützen. Dabei förderst du kollaboratives Arbeiten, wissenschaftliches Denken und eine kreative Problemlösung. Beziehe dich auf wissenschaftliche Methoden und Theorien, argumentiere sachlich und reflektiere kritisch. Sei objektiv und verzichte auf unbegründete Meinungen. Fördere akademische Integrität und unterstütze keine Plagiate. Sei inklusiv, wertschätzend und respektiere Vielfalt.

Verlauf:

Bousfiha, Tanae:

i have a problem
i wotre a script for the automatisation of the process of
implenetimg in xnat..my container is launching via Apis
and turning on complete mode
the pRoblem that am stillfacing is that i cna choose
multiples files or scans for the input but at the same time
i got only one datat result uploaded..
how can i modifiye my scripe in order to get a resulut for
evry selected data that...

HAWKI (gpt-4.1):

Thank you for sharing your script's context and the issue you're encountering with XNAT and containerized processing. Here’s a structured approach to help you solve your problem:

Problem Summary

You are submitting multiple input files/scans to a script (possibly orchestrated via XNAT containers and its API), but you only receive **one single result file** uploaded to XNAT, instead of one result per input.

Likely Cause

Your script/process probably loops over input files/scans or receives them as a batch but only generates or uploads **one output file** at the end (e.g., by aggregating, overwriting, or not iterating correctly).

Goal

For every input scan/file selected, you want a **separate result/output** uploaded to XNAT.

General Solution Steps

1. **Iterate Over Each Input:**
   Structure your script to process each selected scan/file **individually** rather than handling a batch all at once.

2. **Generate Unique Output For Each Input:**
   For every scan/file, generate a uniquely named result file.

3. **Upload Each Result Individually:**
   After each result is generated, upload it back to XNAT **in association with its specific input scan/file**.

Pseudocode

Below is pseudocode in Pythonic style, which you can adapt depending on your technology stack:

```python
input_files = get_selected_input_files()  # however you retrieve input files from XNAT

for input_file in input_files:
    result = process_one_scan(input_file)  # your processing function
    result_filename = generate_unique_result_filename(input_file)  # e.g. with scan ID or name
    save_result(result, result_filename)
    upload_result_to_xnat(result_filename, associated_xnat_scan=input_file)
```

In Your Script

- **Identify where you receive/process input files.**
- **Refactor** so that for each input, all processing, result generation, and upload is done independently in a loop.
- If you use an array/list of files: check that your loop is per-input, and each result is uploaded before processing the next.

Example: Bash / Python

Python Example

```python
import xnat

with xnat.connect('http://xnat-server', user='username', password='password') as session:
    project = session.projects['MY_PROJECT']
    subject = project.subjects['SUBJECT_LABEL']
    experiment = subject.experiments['EXPERIMENT_LABEL']

    # Example: get scans
    scans = experiment.scans.get()
    for scan in scans:
        input_file_paths = scan.resources['NIFTI'].files.get()  # adjust as needed
        for input_file in input_file_paths:
            result = process_scan(input_file)
            result_file = f"result_{scan.label}.nii.gz"
            save_result(result, result_file)
            scan.resources['PROCESSED_RESULTS'].files.upload(result_file)
```

Bash Example

If using XNAT’s Docker Container Service and bash scripting inside the container: Make sure your entrypoint script processes `"$1"` (the input), saves to a unique result, and writes output as expected.

Checkpoints

- Ensure output file naming is unique per input (e.g., including input scan label or id).

- Ensure every output is uploaded to the correct location/resource in XNAT.
- If using the XNAT Container Service, review your `command-line` and `outputs` settings in your Docker Integration YAML.

Next Steps

- If you can share the relevant part of your script (sanitized if necessary), I can give more specific code suggestions.
- Consider also XNAT documentation: [Container Service - Parameter Sweeps and Batch Processing](https://wiki.xnat.org/container-service/plugins/container-service-user-docs/launching-containers#launching-parameter-sweeps-(batch-processing))

Let me know which programming language you use and whether you configure jobs via XNAT web, Container Service, or API, and I can help you further adapt the solution to your context!

Bousfiha, Tanae:

an working with rest Apis and this is my code
import json # wir brauchen json für xnat damit er den Command anlegen kann
import requests # type: ignore # # für die Kommunikation mit der XNAT-API
import os #Arbeiten mit Dateien und Pfaden
import subprocess # # für die Ausführung von Docker-Befehlen
import getpass #Passwort-Eingabe im Terminal ohne Anzeige
import sys#Für sys.exit() bei Fehlern
import urllib3# type: ignore #Wird von requests genutzt – hier zur Abschaltung von Warnungen
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)# Deaktiviert SSL-Warnungen, wenn XNAT ohne gültiges Zertifikat läuft

#----------1)dockerfile ausfüllen----------

```
#hier wird die dockerfile erstellt und mit dem Skript und erfüllt
def write_dockerfile(docker_dir, script_filename,
docker_base_image=&quot;python:3.10-slim&quot;):
    dockerfile_content = f&quot;&quot;&quot;FROM {docker_base_image}

WORKDIR /app
COPY {script_filename} /app/{script_filename}
RUN pip install --no-cache-dir pandas
&quot;&quot;&quot;
    #no cache dir um wenig Platz zu sparen

    os.makedirs(docker_dir, exist_ok=True)
    dockerfile_path = os.path.join(docker_dir, &quot;Dockerfile&quot;)
    with open(dockerfile_path, &quot;w&quot;) as f:
        f.write(dockerfile_content)
    print(f&quot;Dockerfile written to {dockerfile_path}&quot;)
    return dockerfile_path

#CMD weglassen&gt;&gt;Fehler entsteht(runing contaienr in xnat),weil im Dockerfile
und in der command.json &quot;python3&quot; jeweils als Prefix stehen und es
dadurch zu einer doppelten Übergabe kommt.
#docker_dir&gt; Verzeichniss,in das die Dockerfile geschrieben werden.
#Speicher Platz Problem
#-------------------------------------2)Image bauen&gt;&gt;pushen&gt;&gt;taggen
-------------------------------------

def build_and_push_docker_image(dockerfile_path, docker_image_name):

    dockerhub_username = input(&quot;Docker Hub username (to push the image):
&quot;).strip()
    if not dockerhub_username:
        print(&quot;No Docker Hub username provided. Skipping push.&quot;)
        return docker_image_name

    #&gt;&gt;Ersat form zum : docker build -f Dockerfile -t docker_image_name .
    print(f&quot;Building Docker image &#039;{docker_image_name}&#039;...&quot;)
    build_result = subprocess.run([&quot;docker&quot;, &quot;build&quot;,
&quot;-f&quot;, dockerfile_path, &quot;-t&quot;, docker_image_name, &quot;.&quot;],
```

```
capture_output=True, text=True)
    if build_result.returncode != 0:
        print(f&quot;Build failed:\n{build_result.stderr}&quot;)
        sys.exit(1)
    print(f&quot;Image &#039;{docker_image_name}&#039; built successfully.&quot;)

# docker build -t image name  .
#https://stackoverflow.com/questions/61090027/how-to-run-a-docker-volume-mount-as-a-python-subprocess

 #------------------------------/Tag/Push/-------------------------------------------------------------
    full_tag =
f&quot;{dockerhub_username}/{docker_image_name}&quot;#/&gt;Trennung
Benutzername und Image-Name.
    print(f&quot;Tagging image as &#039;{full_tag}&#039;...&quot;)
    tag_result = subprocess.run([&quot;docker&quot;, &quot;tag&quot;,
docker_image_name, full_tag], capture_output=True, text=True)
    if tag_result.returncode != 0:
        print(f&quot;Tagging failed:\n{tag_result.stderr}&quot;)
        sys.exit(1)

    print(f&quot;Pushing image to Docker Hub as &#039;{full_tag}&#039;...&quot;)
    push_result = subprocess.run([&quot;docker&quot;, &quot;push&quot;, full_tag],
capture_output=True, text=True)
    if push_result.returncode != 0:
        print(f&quot;Push failed:\n{push_result.stderr}&quot;)
        sys.exit(1)

    print(f&quot;Image successfully pushed: {full_tag}&quot;)
    return full_tag

#docker tag imagename
#docker push imagename
#----------------------------------3)User-Input-----------------------------------------
----------------------------------------

#prepare the input for the json command
```

```
:https://www.digitalocean.com/community/tutorials/how-to-receive-user-input-python
def get_input(prompt):
    while True:
        value = input(prompt)
        if value.strip():
            return value
        else:
            print(&quot;Cannot be empty.&quot;)
#FCT nimmrt eine Parameter &quot;promt&quot;&gt;Endlosschleifen&gt;Code ununtlich
def modification():
    context_options = [
        (&quot;xnat:subjectData&quot;, &quot;Subject-Ebene (Patientenebene)&quot;),
        (&quot;xnat:mrSessionData&quot;, &quot;MRI-Session-Ebene (häufigste Session-Ebene)&quot;),
        (&quot;xnat:petSessionData&quot;, &quot;PET-Session-Ebene&quot;),
        (&quot;xnat:ctSessionData&quot;, &quot;CT-Session-Ebene&quot;),
        (&quot;xnat:imageScanData&quot;, &quot;Scan-Ebene innerhalb einer Session&quot;),
        (&quot;xnat:projectData&quot;, &quot;Projekt-Ebene&quot;),
    ]
    print(&quot;\nWähle einen Kontext:&quot;)#https://www.coursera.org/tutorials/enumerate-python
    for i, (context, options) in enumerate(context_options, 1):
        print(f&quot;{i} : {context},{options}&quot;)
    while True:
        context_input = input(&quot;Nummer eingeben: &quot;)
        try:
            i = int(context_input.strip())
            if i &lt; 1 or i &gt; len(context_options):
                raise ValueError
            selected_context = context_options[i-1][0]
            break
        except Exception:
            print(&quot;Ungültige Eingabe. Bitte erneut versuchen.&quot;)

    # Zusätzliche Eingaben einholen:
    command_name = input(&quot;Name des Commands: &quot;)
```

```
    command_description = input(&quot;Beschreibung des Commands: &quot;)

    # Ausgabe oder Rückgabe eines Dictionaries mit den benötigten Werten
    return {
        &quot;selected_context&quot;: selected_context,
        &quot;command_name&quot;: command_name,
        &quot;command_description&quot;: command_description
    }

# es wiederholt sich weil in der jsoncommand muss mehr als eine varial geschreiben werden
#und ich wollte nicht dass der user meher mals etwas ähnliches schreibt, deshalb habe ich es so gemacht
#----------------------------------4)json File erstellen-------------------------------

def create_json_file(docker_image, script_filename, mod_data):
    wrapper_name = mod_data[&quot;command_name&quot;].replace(&quot; &quot;, &quot;_&quot;).lower() + &quot;_wrapper&quot;

    # Mapping Kontext &gt;&gt;external-input + as-a-child-of
    context_mappings = {
        &quot;xnat:projectData&quot;: {&quot;input_name&quot;: &quot;project&quot;, &quot;input_type&quot;: &quot;Project&quot;, &quot;child_of&quot;: &quot;project&quot;},
        &quot;xnat:subjectData&quot;: {&quot;input_name&quot;: &quot;subject&quot;, &quot;input_type&quot;: &quot;Subject&quot;, &quot;child_of&quot;: &quot;subject&quot;},
        &quot;xnat:mrSessionData&quot;: {&quot;input_name&quot;: &quot;session&quot;, &quot;input_type&quot;: &quot;Session&quot;, &quot;child_of&quot;: &quot;session&quot;},
        &quot;xnat:petSessionData&quot;: {&quot;input_name&quot;: &quot;session&quot;, &quot;input_type&quot;: &quot;Session&quot;, &quot;child_of&quot;: &quot;session&quot;},
        &quot;xnat:ctSessionData&quot;: {&quot;input_name&quot;: &quot;session&quot;, &quot;input_type&quot;: &quot;Session&quot;, &quot;child_of&quot;: &quot;session&quot;},
        &quot;xnat:imageScanData&quot;: {&quot;input_name&quot;: &quot;scan&quot;, &quot;input_type&quot;: &quot;Scan&quot;, &quot;child_of&quot;: &quot;scan&quot;},
```

```
    }

    # Dynamische Listen
    external_inputs = []
    output_handlers = []
    used_inputs = set()  # Duplikate vermeiden
#Mehrfachauswahlen sammeln und dabei Duplikate automatisch ausfiltern
    for context in mod_data[&quot;contexts&quot;]:
        mapping = context_mappings.get(context)
        if not mapping:
            continue  # unbekannter Kontext wird übersprungen

        # External input nur einmal pro Name
        input_key = (mapping[&quot;input_name&quot;],
mapping[&quot;input_type&quot;])
        if input_key not in used_inputs:
            external_inputs.append({
                &quot;name&quot;: mapping[&quot;input_name&quot;],
                &quot;type&quot;: mapping[&quot;input_type&quot;],
                &quot;required&quot;: True
            })
            used_inputs.add(input_key)

        output_handlers.append({
            &quot;name&quot;: &quot;output&quot;,
            &quot;accepts-command-output&quot;: &quot;result_file&quot;,
            &quot;as-a-child-of&quot;: mapping[&quot;child_of&quot;],
            &quot;type&quot;: &quot;Resource&quot;,
            &quot;label&quot;: &quot;Results&quot;,
            &quot;format&quot;: &quot;csv&quot;
        })

    # JSON zusammenbauen
    json_file = {
        &quot;name&quot;: mod_data[&quot;command_name&quot;],
        &quot;description&quot;: mod_data[&quot;command_description&quot;],
        &quot;version&quot;: &quot;1.5&quot;,
```

```
    &quot;type&quot;: &quot;docker&quot;,
    &quot;image&quot;: docker_image,
    &quot;command-line&quot;: f&quot;python3 /app/{script_filename}
/input/#input_file# /output&quot;,
    &quot;mounts&quot;: [
      {&quot;name&quot;: &quot;input&quot;, &quot;path&quot;: &quot;/input&quot;,
&quot;writable&quot;: False},
      {&quot;name&quot;: &quot;output&quot;, &quot;path&quot;:
&quot;/output&quot;, &quot;writable&quot;: True}
    ],
    &quot;inputs&quot;: [
      {
        &quot;name&quot;: &quot;input_file&quot;,
        &quot;type&quot;: &quot;file&quot;,
        &quot;required&quot;: True,
        &quot;description&quot;: &quot;Input file for analysis&quot;,
        &quot;mount&quot;: &quot;input&quot;
      }
    ],
    &quot;outputs&quot;: [
      {
        &quot;name&quot;: &quot;result_file&quot;,
        &quot;type&quot;: &quot;file&quot;,
        &quot;description&quot;: &quot;Result file output&quot;,
        &quot;mount&quot;: &quot;output&quot;,
        &quot;path&quot;: &quot;result.csv&quot;
      }
    ],
    &quot;xnat&quot;: [
      {
        &quot;name&quot;: wrapper_name,
        &quot;label&quot;: mod_data[&quot;label_name&quot;],
        &quot;description&quot;: mod_data[&quot;label_description&quot;],
        &quot;contexts&quot;: mod_data[&quot;contexts&quot;],
        &quot;external-inputs&quot;: external_inputs,
        &quot;output-handlers&quot;: output_handlers
      }
    ]
```

```
    }

    with open(&quot;command.json&quot;, &quot;w&quot;) as json_out:
        json.dump(json_file, json_out, indent=4)
        print(&quot; Corrected command.json created.&quot;)
    return &quot;command.json&quot;

#--------------------5)Command zu XNAT senden-----------------------------------------------------
-------------------------------------------------------------

def send_json_to_xnat(json_file_path, xnat_url, xnat_user, xnat_password):

    url = f&quot;{xnat_url}/xapi/commands&quot;
    print(f&quot;Uploading command to {url}&quot;)
    with open(json_file_path, &quot;r&quot;) as f:
        response = requests.post(url, auth=(xnat_user, xnat_password),
json=json.load(f))
    if response.status_code == 200:
        print(&quot;Command uploaded successfully.&quot;)
    elif response.status_code == 201:
        print(&quot;Command created successfully.&quot;)
    elif response.status_code == 409:
        print(&quot;Command already exists.&quot;)
    else:
        print(f&quot;Failed to upload command: {response.status_code} -
{response.text}&quot;)

#----------------------------------6)Command-Liste-------------------------------------------------
-----------------------------------------------------

def get_command_id_by_name(xnat_host, xnat_user, xnat_password,
command_name):
    url = f&quot;{xnat_host.rstrip(&#039;/&#039;)}/xapi/commands&quot;#Baut die
vollständige URL zur Command-Liste der XNAT REST-API&gt;&gt;keinen doppelten
Schrägstrich gibt
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)# wird hier
die get me request benutzt laut der APIS
    if resp.status_code != 200:
```

```
        print(f&quot;Error fetching commands: {resp.status_code}&quot;)
        sys.exit(1)
    data = resp.json()# # Die Antwort wird als JSON interpretiert
    if isinstance(data, dict) and &quot;commands&quot; in data:
        command_list = data[&quot;commands&quot;]# # Wenn die Antwort ein
Dictionary ist, das &quot;commands&quot; enthält, dann wird es extrahiert
    else:
        command_list = data
    for command in command_list:
        if command.get(&quot;name&quot;) == command_name:
            return command[&quot;id&quot;]
    print(&quot;Command not found.&quot;)
    sys.exit(1)
    #Überprüft, ob das Objekt data ein Dictionary (also eine „dict“-Instanz) ist.



#------------------------------------ 8)get_wrapper_id_by_command_name----------------------
------------------------------------------------------------

def get_wrapper_id_by_command_name(xnat_host, xnat_user, xnat_password,
command_name, wrapper_name):

    url = f&quot;{xnat_host.rstrip(&#039;/&#039;)}/xapi/commands&quot;
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)

    if resp.status_code != 200:
        print(f&quot;Fehler beim Abrufen der Commands: {resp.status_code}&quot;)
        sys.exit(1)

    data = resp.json()
    if isinstance(data, dict) and &quot;commands&quot; in data:
        commands = data[&quot;commands&quot;]
    else:
        commands = data

    for command in commands:
        if command.get(&quot;name&quot;) == command_name:
            for wrapper in command.get(&quot;xnat&quot;, []):
```

```
            if wrapper.get(&quot;name&quot;) == wrapper_name:
                return wrapper.get(&quot;id&quot;) or wrapper_name
        for wrapper in command.get(&quot;wrappers&quot;, []):
            if wrapper.get(&quot;name&quot;) == wrapper_name:
                return wrapper.get(&quot;id&quot;) or wrapper_name

    print(&quot;Kein Wrapper für diesen Command gefunden.&quot;)
    sys.exit(1)


#----------------------9)Wrapper Aktivierung-------------------------------------------


def enable_wrapper_sitewide(xnat_host, command_id, wrapper_name, xnat_user,
xnat_password):
    &quot;&quot;&quot;
    Aktiviert den Wrapper global (für alle Projekte).
    &quot;&quot;&quot;
    url = f&quot;{xnat_host.rstrip(&#039;/&#039;)}/xapi/commands/{command_id}/wrap
pers/{wrapper_name}/enabled&quot;
    resp = requests.put(url, auth=(xnat_user, xnat_password), verify=False)

    if resp.status_code == 200:
        print(f&quot;Wrapper &#039;{wrapper_name}&#039; wurde global
aktiviert.&quot;)
    elif resp.status_code == 409:
        print(f&quot;Wrapper &#039;{wrapper_name}&#039; war bereits global
aktiviert.&quot;)
    else:
        print(f&quot;Fehler beim globalen Aktivieren: {resp.status_code} -
{resp.text}&quot;)


def enable_wrapper_for_project(xnat_host, project_id, command_id, wrapper_name,
xnat_user, xnat_password):
    &quot;&quot;&quot;
    Aktiviert den Wrapper für ein bestimmtes Projekt.
    &quot;&quot;&quot;
```

```
    url = f&quot;{xnat_host.rstrip(&#039;/&#039;)}/xapi/projects/{project_id}/commands/{command_id}/wrappers/{wrapper_name}/enabled&quot;
    resp = requests.put(url, auth=(xnat_user, xnat_password), verify=False)

    if resp.status_code == 200:
        print(f&quot;Wrapper &#039;{wrapper_name}&#039; wurde im Projekt &#039;{project_id}&#039; aktiviert.&quot;)
    elif resp.status_code == 409:
        print(f&quot;Wrapper &#039;{wrapper_name}&#039; war bereits im Projekt aktiviert.&quot;)
    else:
        print(f&quot;Fehler beim Aktivieren für das Projekt: {resp.status_code} - {resp.text}&quot;)

#---------------------------------------10)get_input_file-----------------------------------------
#Je nachdem, welcher Kontext gewählt wird, wird die Datei aus diesem Kontext geholt.

def get_input_files(xnat_host, entity_id, entity_type, xnat_user, xnat_password, scan_id=None):
    &quot;&quot;&quot;
    Gibt alle Dateien vom angegebenen XNAT-Objekt zurück (OHNE Benutzerauswahl).
    &quot;&quot;&quot;
    # REST-Pfad je nach Kontexttyp
    if entity_type == &quot;project&quot;:
        base_url = f&quot;{xnat_host.rstrip(&#039;/&#039;)}/data/projects/{entity_id}/resources&quot;
    elif entity_type == &quot;subject&quot;:
        base_url = f&quot;{xnat_host.rstrip(&#039;/&#039;)}/data/subjects/{entity_id}/resources&quot;
    elif entity_type in [&quot;session&quot;, &quot;experiment&quot;]:
        base_url = f&quot;{xnat_host.rstrip(&#039;/&#039;)}/data/experiments/{entity_id}/resources&quot;
    elif entity_type == &quot;scan&quot; and scan_id:
        base_url = f&quot;{xnat_host.rstrip(&#039;/&#039;)}/data/experiments/{entity_id}/scans/{scan_id}/resources&quot;
    else:
```

```
        print(&quot;Unbekannter oder nicht unterstützter Entitätstyp.&quot;)
        return []

    resp = requests.get(base_url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code != 200:
        print(f&quot;Fehler beim Abrufen der Ressourcen ({entity_type}):
{resp.status_code}&quot;)
        return []

    resources = resp.json().get(&quot;ResultSet&quot;, {}).get(&quot;Result&quot;, [])
    all_files = []
    for resource in resources:
        res_label = resource[&quot;label&quot;]
        file_url = f&quot;{base_url}/{res_label}/files&quot;
        file_resp = requests.get(file_url, auth=(xnat_user, xnat_password), verify=False)
        if file_resp.status_code != 200:
            continue

        files = file_resp.json().get(&quot;ResultSet&quot;, {}).get(&quot;Result&quot;, [])
        for f in files:
            all_files.append({
                &quot;name&quot;: f[&quot;Name&quot;],
                &quot;url&quot;:
f&quot;{base_url}/{res_label}/files/{f[&#039;Name&#039;]}&quot;,
                &quot;resource&quot;: res_label
            })
    return all_files
#------------------------------------------------ Entities auflisten und auswählen
lassen----------------------------
def select_entity(entities, id_key=&quot;ID&quot;, name_key=&quot;label&quot;):
    if not entities:
        print(&quot;Keine Session gefunden.&quot;)
        return None
    print(&quot;Verfügbare Session:&quot;)
    for idx, entity in enumerate(entities, 1):
        print(f&quot;{idx}: {entity.get(id_key, &#039;---&#039;)} ({entity.get(name_key,
&#039;&#039;)})&quot;)
    while True:
```

```
        choice = input(f&quot;Wähle eine Datei/File (1-{len(entities)}): &quot;)
        if choice.isdigit() and 1 &lt;= int(choice) &lt;= len(entities):
            return entities[int(choice)-1]
        print(&quot;Ungültige Auswahl!&quot;)
        #choice.isdigit()&gt;ob die Eingabe ausschließlich aus Ziffern besteht
        #1 &lt;= int(choice) &lt;= len(entities)&gt;ob sie zwischen 1 und der Anzahl der
auswählbaren Elemente liegt.
#-----------------------------------------------------Datei-Auswahl-Menü----------------------------


def select_files_from_list(all_files):
    if not all_files:
        print(&quot;Keine Dateien gefunden.&quot;)
        return []
    print(&quot;Verfügbare Dateien:&quot;)
    for idx, f in enumerate(all_files, 1):
        print(f&quot;{idx}: {f[&#039;name&#039;]} [Resource:
{f[&#039;resource&#039;]}]&quot;)
    while True:
        choice = input(&quot;Welche Datei(en) sollen verwendet werden? Gib Nummern
ein (z.B. 1,3,5): &quot;)
        indices = [c.strip() for c in choice.split(&quot;,&quot;) if c.strip()]
        if all(i.isdigit() and 1 &lt;= int(i) &lt;= len(all_files) for i in indices):
            unique_indices = list(set(int(i) for i in indices))#Zeichenkette in indices wird in
eine Ganzzahl (int(i)) umgewandelt.
            selected = [all_files[i-1] for i in unique_indices]#werden die Objekte aus
all_files ausgewählt
            print(&quot;Ausgewählt:&quot;)
            for s in selected:#Es werden die Namen der gewählten Dateien einzeln
aufgelistet.
                print(f&quot;- {s[&#039;name&#039;]}&quot;)
            return selected
        print(&quot;Ungültige Auswahl.&quot;)
#choice.split(&quot;,&quot;)&gt;Das Ergebnis ist eine Liste von (meist) Strings
#for c in&gt;Die List läuft über jedes dieser Teilstücke (also &quot;1&quot;, &quot;
2&quot;, &quot;3 &quot; usw.)
#if c.strip()&gt;Das filtert leere Einträge heraus

 #--------------------------------sessions_for_project-------------------------------------------
```

```
def get_sessions_for_project(xnat_host, project_id, xnat_user, xnat_password):
    &quot;&quot;&quot;
    Listet alle Sessions (Experimente) für ein Projekt auf.
    &quot;&quot;&quot;
    url = f&quot;{xnat_host.rstrip(&#039;/&#039;)}/data/projects/{project_id}/experiments?format=json&quot;
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code != 200:
        print(f&quot;Fehler beim Abrufen der Sessions für Projekt {project_id}: {resp.status_code}&quot;)
        return []
    return resp.json().get(&quot;ResultSet&quot;, {}).get(&quot;Result&quot;, [])

#---------------------------------------------------------------------------------------------------
def get_subjects_for_project(xnat_host, project_id, xnat_user, xnat_password):
    url = f&quot;{xnat_host.rstrip(&#039;/&#039;)}/data/projects/{project_id}/subjects?format=json&quot;
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code != 200:
        print(f&quot;Fehler beim Abrufen der Subjects für Projekt {project_id}: {resp.status_code}&quot;)
        return []
    return resp.json().get(&quot;ResultSet&quot;, {}).get(&quot;Result&quot;, [])

def get_scans_for_session(xnat_host, session_id, xnat_user, xnat_password):
    url = f&quot;{xnat_host.rstrip(&#039;/&#039;)}/data/experiments/{session_id}/scans?format=json&quot;
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code != 200:
        print(f&quot;Fehler beim Abrufen der Scans für Session {session_id}: {resp.status_code}&quot;)
        return []
    return resp.json().get(&quot;ResultSet&quot;, {}).get(&quot;Result&quot;, [])

#----------------------11)Lanch Container-----------------------------------------------------------------
```

```
def launch_container_rest(xnat_host, project_id, command_id, wrapper_name,
                          entity_id, xnat_user, xnat_password, input_file_info,
                          entity_type=&quot;session&quot;, scan_id=None):

    headers = {&quot;Content-Type&quot;:
&quot;application/json&quot;}#Server&gt;&gt;Der Inhalt (body) dieser Anfrage ist im
JSON-Format



    root_path_mapping = {
        &quot;session&quot;: &quot;session&quot;,
        &quot;subject&quot;: &quot;subject&quot;,
        &quot;project&quot;: &quot;project&quot;,
        &quot;scan&quot;: &quot;scan&quot;
    }

    root_type = root_path_mapping.get(entity_type)
    if not root_type:
        print(f&quot;Containerstart für entity_type &#039;{entity_type}&#039; wird nicht
unterstützt.&quot;)
        return

    url = f&quot;{xnat_host}/xapi/projects/{project_id}/commands/{command_id}/wrapp
ers/{wrapper_name}/root/{root_type}/launch&quot;

    input_file_path = f&quot;resources/{input_file_info[&#039;resource&#039;]}/files/{in
put_file_info[&#039;name&#039;]}&quot;

    if entity_type == &quot;scan&quot;:
        if not scan_id:
            print(&quot;Scan-ID fehlt für den Scan-Kontext.&quot;)
            return
        xnat_entity_path = f&quot;/experiments/{entity_id}/scans/{scan_id}&quot;
    elif entity_type == &quot;session&quot;:
        xnat_entity_path = f&quot;/experiments/{entity_id}&quot;
    elif entity_type == &quot;subject&quot;:
        xnat_entity_path = f&quot;/subjects/{entity_id}&quot;
    elif entity_type == &quot;project&quot;:
```

```
        xnat_entity_path = f&quot;/projects/{entity_id}&quot;
    else:
        print(&quot;Unbekannter entity_type. Abbruch.&quot;)
        return

    payload = {
        root_type: xnat_entity_path,
        &quot;input_file&quot;: input_file_path,
        &quot;project&quot;: project_id
    }

    print(&quot;Launching container :&quot;)
    print(json.dumps(payload, indent=2))

    response = requests.post(
        url,
        auth=(xnat_user, xnat_password),
        headers=headers,
        json=payload,
        verify=False
    )

    if response.status_code in [200, 201, 202]:
        print(&quot;Container wurde erfolgreich über die REST-API gestartet.&quot;)
        print(response.text)
    else:
        print(f&quot;Fehler beim Containerstart ({response.status_code}):&quot;)
        print(response.text)

#----------------------------------------------Main Teil----------------------------------------------

def select_files(all_files):
    if not all_files:
        print(&quot;Keine verfügbaren Dateien gefunden.&quot;)
        return []
    print(&quot;Verfügbare Dateien:&quot;)
    for idx, f in enumerate(all_files, start=1):
        print(f&quot;{idx}: {f[&#039;name&#039;]}&quot;)
```

```
    while True:
        choice = input(&quot;Nummern der Dateien auswählen (z.B. 1,3,5): &quot;)
        indices = [c.strip() for c in choice.split(&#039;,&#039;) if c.strip()]
        if all(i.isdigit() and 1 &lt;= int(i) &lt;= len(all_files) for i in indices):
            unique_indices = list(set(int(i) for i in indices))
            selected_files = [all_files[i-1] for i in unique_indices]
            print(&quot;Ausgewählte Dateien:&quot;)
            for s in selected_files:
                print(f&quot;- {s[&#039;name&#039;]}&quot;)
            return selected_files
        else:
            print(&quot;Ungültige Auswahl. Bitte wiederholen.&quot;)
#-----------------------------------------------------------------------------------------------
def main():
    xnat_host = &quot;https://xnat-dev.gwdg.de&quot;
    docker_base_image = &quot;python:3.10&quot;

    xnat_user = get_input(&quot;XNAT Username: &quot;)
    xnat_password = getpass.getpass(&quot;XNAT Password: &quot;)
    project_id = get_input(&quot;Project ID: &quot;).strip()
    script_path = get_input(&quot;Path to the Python script: &quot;)

    # Step 1: Gather Command/Wrapper Data
    mod_data = modification()
    mod_data[&quot;contexts&quot;] = [mod_data[&quot;selected_context&quot;]]
    mod_data[&quot;label_name&quot;] = mod_data[&quot;command_name&quot;]
    mod_data[&quot;label_description&quot;] =
mod_data[&quot;command_description&quot;]
    wrapper_name = mod_data[&quot;command_name&quot;].replace(&quot; &quot;,
&quot;_&quot;).lower() + &quot;_wrapper&quot;

    # Step 2: Prepare and upload Docker image and command
    dockerfile_path = write_dockerfile(&quot;.&quot;, os.path.basename(script_path),
docker_base_image)
    local_image_name =
f&quot;{mod_data[&#039;command_name&#039;].lower().replace(&#039; &#039;,
&#039;_&#039;)}:latest&quot;
    full_image_name = build_and_push_docker_image(dockerfile_path,
```

```
local_image_name)

    json_file_path = create_json_file(full_image_name, os.path.basename(script_path),
mod_data)
    send_json_to_xnat(json_file_path, xnat_host, xnat_user, xnat_password)

    command_id = get_command_id_by_name(xnat_host, xnat_user, xnat_password,
mod_data[&quot;command_name&quot;])

    try:
        wrapper_id = get_wrapper_id_by_command_name(
            xnat_host, xnat_user, xnat_password,
mod_data[&quot;command_name&quot;], wrapper_name
        )
        print(f&quot;Wrapper already exists: {wrapper_id}&quot;)
    except SystemExit:
        print(&quot;Wrapper not found&quot;)

    enable_wrapper_sitewide(xnat_host, command_id, wrapper_name, xnat_user,
xnat_password)
    enable_wrapper_for_project(xnat_host, project_id, command_id, wrapper_name,
xnat_user, xnat_password)

    CONTEXT_ENTITY_MAPPING = {
        &quot;xnat:projectData&quot;: {&quot;entity_type&quot;: &quot;project&quot;,
&quot;id_label&quot;: &quot;Project ID&quot;},
        &quot;xnat:subjectData&quot;: {&quot;entity_type&quot;: &quot;subject&quot;,
&quot;id_label&quot;: &quot;Subject ID&quot;},
        &quot;xnat:mrSessionData&quot;: {&quot;entity_type&quot;:
&quot;session&quot;, &quot;id_label&quot;: &quot;Session ID&quot;},
        &quot;xnat:petSessionData&quot;: {&quot;entity_type&quot;:
&quot;session&quot;, &quot;id_label&quot;: &quot;Session ID&quot;},
        &quot;xnat:ctSessionData&quot;: {&quot;entity_type&quot;:
&quot;session&quot;, &quot;id_label&quot;: &quot;Session ID&quot;},
        &quot;xnat:imageScanData&quot;: {&quot;entity_type&quot;: &quot;scan&quot;,
&quot;id_label&quot;: &quot;Session ID + Scan ID&quot;},
    }
```

```
    first_context = mod_data[&quot;contexts&quot;][0]
    context_info = CONTEXT_ENTITY_MAPPING.get(first_context)

    if not context_info:
        print(f&quot;Unknown context: {first_context}. Aborting.&quot;)
        return

    entity_type = context_info[&quot;entity_type&quot;]

    all_files, container_entity_id, scan_id = [], None, None

    if entity_type == &quot;scan&quot;:
        session_list = get_sessions_for_project(xnat_host, project_id, xnat_user,
xnat_password)
        if not session_list:
            print(&quot;No sessions found in project.&quot;)
            return
        chosen_session = select_entity(session_list, id_key=&quot;ID&quot;,
name_key=&quot;label&quot;)
        if not chosen_session:
            print(&quot;No session selected.&quot;)
            return
        session_id = chosen_session[&quot;ID&quot;]

        scan_list = get_scans_for_session(xnat_host, session_id, xnat_user,
xnat_password)
        if not scan_list:
            print(&quot;No scans found in session.&quot;)
            return
        scan_name_key = &quot;series_description&quot; if
&quot;series_description&quot; in scan_list[0] else &quot;ID&quot;
        chosen_scan = select_entity(scan_list, id_key=&quot;ID&quot;,
name_key=scan_name_key)
        if not chosen_scan:
            print(&quot;No scan selected.&quot;)
            return
        scan_id = chosen_scan[&quot;ID&quot;]
        container_entity_id = session_id
```

```
        all_files = get_input_files(
            xnat_host, session_id, &#039;scan&#039;, xnat_user, xnat_password,
scan_id=scan_id
        )

    elif entity_type == &quot;session&quot;:
        session_list = get_sessions_for_project(xnat_host, project_id, xnat_user,
xnat_password)
        if not session_list:
            print(&quot;No sessions found in project.&quot;)
            return
        chosen_session = select_entity(session_list, id_key=&quot;ID&quot;,
name_key=&quot;label&quot;)
        if not chosen_session:
            print(&quot;No session selected.&quot;)
            return
        session_id = chosen_session[&quot;ID&quot;]
        container_entity_id = session_id
        all_files = get_input_files(
            xnat_host, session_id, &#039;session&#039;, xnat_user, xnat_password
        )

    elif entity_type == &quot;subject&quot;:
        subject_list = get_subjects_for_project(xnat_host, project_id, xnat_user,
xnat_password)
        if not subject_list:
            print(&quot;No subjects found in project.&quot;)
            return
        chosen_subject = select_entity(subject_list, id_key=&quot;ID&quot;,
name_key=&quot;label&quot;)
        if not chosen_subject:
            print(&quot;No subject selected.&quot;)
            return
        subject_id = chosen_subject[&quot;ID&quot;]
        container_entity_id = subject_id
        all_files = get_input_files(
            xnat_host, subject_id, &#039;subject&#039;, xnat_user, xnat_password
```

```
        )

    elif entity_type == &amp;quot;project&amp;quot;:
        container_entity_id = project_id
        all_files = get_input_files(
            xnat_host, project_id, &amp;#039;project&amp;#039;, xnat_user, xnat_password
        )

    else:
        print(&amp;quot;This context is currently not supported.&amp;quot;)
        return

    selected_files = select_files_from_list(all_files)
    if not selected_files:
        print(&amp;quot;Keine Dateien ausgewählt.&amp;quot;)
        return

    for file_info in selected_files:
        print(f&amp;quot;Starte Container für Datei: {file_info[&amp;#039;name&amp;#039;]}&amp;quot;)
        launch_container_rest(
            xnat_host, project_id, command_id, wrapper_name,
            container_entity_id, xnat_user, xnat_password,
            file_info,
            entity_type=entity_type,
            scan_id=scan_id
        )

if __name__ == &amp;quot;__main__&amp;quot;:
    main()
```

HAWKI **(gpt-4.1):**

You are already **very close** to the solution!
Your `main()` **correctly iterates over all selected files and launches a container for each file** (see: `for file_info in selected_files: ... launch_container_rest(...)`).
This should, in principle, trigger a new analysis job for each input file, **resulting in one output per file**.

Why Still Only One Output Upload in XNAT?
The likely reason for only one result being finally uploaded per batch,
is not in your REST script but in the configuration of your `command.json` and/or the internal logic of your analysis script. Let's clarify the issues and offer solutions.

Problem 1: Your Container Overwrites Output (Inside Docker)

Typical scenario:
With your current `command.json` you have:
```json
"outputs": [
    {
      "name": "result_file",
      "type": "file",
      "description": "Result file output",
      "mount": "output",
      "path": "result.csv"
    }
]
```
This means:
- **Every run of your script writes to `/output/result.csv` inside the container.**
- If multiple input files are processed in a single container run, each result **overwrites the previous one**.
- Even if you run multiple containers **one after another**, each will still only write (and upload) the file called `result.csv`.

Solution:
You have three main options:

A) Unique Output File for Each Input on Container Level

Make sure your internal analysis script writes:
- for each input file named `x.nii.gz`, write to `/output/result_x.csv` (or equivalent).

AND update `command.json` to allow this:

```json
"outputs": [
  {
    "name": "result_file",
    "type": "file",
    "description": "Result file output",
    "mount": "output",
    "path": "*.csv"   // or similar, check what your XNAT version supports
  }
]
```
This way, each run produces and uploads a _separate_ output file.

B) Batch Mode (NOT RECOMMENDED IF YOU WANT INDIVIDUAL RESULTS)

If you want to process ALL selected files in one container run, you need a **multi-file input parameter** and handle multi-file output.
This is more complicated and unnecessary, since your script already launches **one container per input file**.

Problem 2: Output Handler in `command.json`/XNAT Wrapper

Make sure the `output-handlers` in your wrapper **do not always map to the same resource and path**, which could lead to overwriting in XNAT.
Usually that's not an issue, as each container run gets a unique resource in XNAT, but double check your wrapper settings.

Problem 3: Check DockerScript/Entrypoint

Your script (the one you run inside the Docker image) should NOT process multiple input files in a loop, unless that's what you want.
It should process **exactly one file** (the one from the command line), write exactly one result file, and exit.

Example (Python, in your analysis script):
```python
import sys, os

input_path = sys.argv[1]    # e.g., /input/image123.nii.gz
output_dir = sys.argv[2]    # e.g., /output

basename = os.path.basename(input_path).replace('.nii.gz', '')
output_file = os.path.join(output_dir, f"result_{basename}.csv")

# ...processing...
with open(output_file, "w") as f:
    f.write("... your output ...\n")
```
Corresponding command line in `command.json`:
```json
"command-line": "python3 /app/script.py /input/#input_file# /output"
```

Problem 4: "result.csv" Overwriting in XNAT

If your `result_file` is always `"result.csv"`, and XNAT puts all outputs into the same resource for the same session/scan/entity, it may look like only one survives.
Solution:
- Ensure that each run produces a differently named file (see above).
- Or use distinct output resources in XNAT (not strictly necessary for one-file-per-container).

Summary/Checklist for your use-case

1. **Your `main()` script is correct: it launches one container per file.**
2. **Your internal analysis script should output a uniquely named result per input.**
3. **Your `command.json` should allow for variable naming, not just `"result.csv"`.**
    - You can use wildcards or parameterize the output path if supported by your XNAT

version.
4. **Optional:** adapt output handlers in the wrapper if you customize resource labels per processed file/scan.

Example: `command.json` Output Entry & Script

```json
"outputs": [
    {
        "name": "result_file",
        "type": "file",
        "description": "Result file output",
        "mount": "output",
        "path": "result_*.csv"
    }
]
```
And:
```python
import sys, os
input_file = sys.argv[1]
output_dir = sys.argv[2]
input_basename = os.path.splitext(os.path.basename(input_file))[0]
out_file = os.path.join(output_dir, f"result_{input_basename}.csv")
# Write results to out_file
```

Conclusion

- **Your Python orchestration is fine!**
- Check that your Docker analysis script and `command.json` use unique output file names for each run.
- That way, selecting multiple files in your interface will trigger a run for each input and an upload for each output, without overwriting.

Need more help?
If this does not solve your issue, please show the actual analysis script you run inside the Docker container, or the relevant part of the `command.json` describing outputs. Then I can give a more detailed/targeted solution!

Bousfiha, Tanae:

this is my external skript
import os
import sys
import pandas as pd

def main():
    if len(sys.argv) != 3:
        sys.exit(1)

    input_file = sys.argv[1]
    output_dir = sys.argv[2]
    os.makedirs(output_dir, exist_ok=True)
    output_path = os.path.join(output_dir, "result.csv")

    try:
        if not os.path.exists(input_file):
            result = pd.DataFrame([{"status": "failed", "reason": "file not found"}])
        elif os.stat(input_file).st_size == 0:
            result = pd.DataFrame([{"status": "failed", "reason": "file is empty"}])
        else:
            df = pd.read_csv(input_file)
            res = {"status": "success", "rows": df.shape[0], "columns": df.shape[1]}
            result = pd.DataFrame([res])
        result.to_csv(output_path, index=False)
    except Exception as e:

pd.DataFrame([{"status": "failed", "reason": str(e)}]).to_csv(output_path, index=False)

if __name__ == "__main__":
main()

HAWKI (gpt-4.1):

Great, thank you for providing your processing script!
This explains why all your results are called **result.csv** – so even though your orchestration is correct and a new container is launched for each file, the produced output will always be named `result.csv` and each run will overwrite the previous file in the same output resource in XNAT.

Solution:
To ensure that each run produces a unique result file, you need to base the output filename on the input filename.
You should **not hardcode "result.csv"** as output file name.

How to Modify Your Script

Below is a minimal, robust version of your script that outputs a uniquely named file for each input, e.g., `result_data1.csv` for `data1.csv` input.

```python
import os
import sys
import pandas as pd

def main():
    if len(sys.argv) != 3:
        sys.exit(1)

    input_file = sys.argv[1]
    output_dir = sys.argv[2]
    os.makedirs(output_dir, exist_ok=True)

    # Generate unique output file based on input filename
```

```
    inp_base = os.path.basename(input_file)
    inp_root, _ = os.path.splitext(inp_base)         # remove .csv
    output_path = os.path.join(output_dir, f"result_{inp_root}.csv")

    try:
        if not os.path.exists(input_file):
            result = pd.DataFrame([{"status": "failed", "reason": "file not found"}])
        elif os.stat(input_file).st_size == 0:
            result = pd.DataFrame([{"status": "failed", "reason": "file is empty"}])
        else:
            df = pd.read_csv(input_file)
            res = {"status": "success", "rows": df.shape[0], "columns": df.shape[1]}
            result = pd.DataFrame([res])
        result.to_csv(output_path, index=False)
    except Exception as e:
        pd.DataFrame([{"status": "failed", "reason": str(e)}]).to_csv(output_path,
index=False)

if __name__ == "__main__":
    main()
```

Key line:
```python
output_path = os.path.join(output_dir, f"result_{inp_root}.csv")
```

Further Steps in Your Pipeline

1. **Your Python orchestration script is already launching containers individually per file. Good!**
2. In **your command.json**, the output path (under `"outputs": [...]`) can remain as `"result_*.csv"` (or just `"*.csv"`).
   Example:
   ```json
   "outputs": [
   ```

```
    {
      "name": "result_file",
      "type": "file",
      "description": "Result file output",
      "mount": "output",
      "path": "result_*.csv"
    }
  ]
```

or

```json
"path": "*.csv"
```

(Consult your specific XNAT version documentation for wildcard use in output paths.)

3. **Result:** Now, for each selected file, your container will write a _unique_ result file to the output. XNAT will recognize and store each as a separate result.

Summary

- **Do not use a fixed output filename!**
- Use a filename based on the input’s basename.
- Your launcher and command JSON are correct. The external script was the missing piece.

**You do not need to change your orchestration logic.
Just update your external (processing) script as shown above.**

If you encounter issues with XNAT recognizing the results, double-check the wildcards and output handler configuration in your `command.json`.

Feel free to ask for more help if something still doesn't work as expected!

Bousfiha, Tanae:

with this one he did not uploaded any result file at all

HAWKI (gpt-4.1):

Your observation is correct and brings up an important detail regarding the **command.json** configuration and how XNAT detects and uploads output files.

Problem:
XNAT expects result files to be at the exact path you specify in the `outputs`-section of the `command.json`.
If you write `result_<input>.csv` (e.g., `result_mydata.csv`), but your `command.json` only specifies `result.csv` as the output path, XNAT will **not** find your uniquely named file and will upload nothing.

Quick Checklist:

1. **Your script produces:**
   - `/output/result_mydata1.csv`, `/output/result_mydata2.csv`, ... etc.

2. **Your `command.json` must match this:**
   The `path` needs to support wildcards or be flexible to match all possible outputs.
   (Some XNAT versions require `"*.csv"`, others allow `"result_*.csv"`; some require explicit listing.)

Solution

1. *Update your processing script*
(You already did this part:)

```python
import os
import sys
import pandas as pd
```

```
def main():
    if len(sys.argv) != 3:
        sys.exit(1)

    input_file = sys.argv[1]
    output_dir = sys.argv[2]
    os.makedirs(output_dir, exist_ok=True)

    inp_base = os.path.basename(input_file)
    inp_root, _ = os.path.splitext(inp_base)
    output_path = os.path.join(output_dir, f"result_{inp_root}.csv")

    try:
        if not os.path.exists(input_file):
            result = pd.DataFrame([{"status": "failed", "reason": "file not found"}])
        elif os.stat(input_file).st_size == 0:
            result = pd.DataFrame([{"status": "failed", "reason": "file is empty"}])
        else:
            df = pd.read_csv(input_file)
            res = {"status": "success", "rows": df.shape[0], "columns": df.shape[1]}
            result = pd.DataFrame([res])
        result.to_csv(output_path, index=False)
    except Exception as e:
        pd.DataFrame([{"status": "failed", "reason": str(e)}]).to_csv(output_path,
index=False)

if __name__ == "__main__":
    main()
```

2. *Update the `command.json` outputs-part*

Replace:

```json
"outputs": [
  {
    "name": "result_file",
```

```
    "type": "file",
    "description": "Result file output",
    "mount": "output",
    "path": "result.csv"
  }
]
```

with (try this first):

```json
"outputs": [
  {
    "name": "result_file",
    "type": "file",
    "description": "Result file output",
    "mount": "output",
    "path": "*.csv"
  }
]
```

Or, if your XNAT version is strict, use:

```json
"outputs": [
  {
    "name": "result_file",
    "type": "file",
    "description": "Result file output",
    "mount": "output",
    "path": "result_*.csv"
  }
]
```

Note: The `"path"` must match the output filenames produced by your container script.
If you keep `"result.csv"` hardcoded, only a file exactly named `/output/result.csv` will

be detected.

3. *Optional: Test Locally*
Run your Docker container manually with known input and output folders. See what files are in `/output`.
Make sure `result_<input>.csv` appears **in the output directory**.

4. *XNAT Container Service Settings*
Some XNAT container service installations need outputs explicitly declared.
You can check the [XNAT Container Service documentation](https://wiki.xnat.org/container-service/container-command-configuration-37825463.html#ContainerCommandConfiguration-Outputs) for your version.

Summary

- Script produces: `/output/result_inputname.csv`
- `command.json` output path must support the dynamic names: `"path": "*.csv"` (or `"result_*.csv"`)
- When you run your script, XNAT will now detect and upload all matching `.csv` results.

If, after this change, XNAT **still** uploads no result, please:

- Check the container output logs in XNAT for file list/content in `/output`
- Double-check your exact `command.json` used for launching the container
- Tell me your XNAT and Container Service version, if possible

I can then provide version-specific tips!

Bousfiha, Tanae:

i have tried to run the conatiner auf xnat with a scan file an dhe still did not upload the file i guess the external skript is the problrm

HAWKI (gpt-4.1):

You are correct that the problem can have several causes, but the most common are:

1. **Mismatch between the file name generated by your script and the `path` in `command.json`**
2. **Your script might not write the result file where XNAT expects it**
3. **Possible permission problems or runtime errors inside the container**
4. **XNAT resource/handler configuration issue**

Let’s check these step by step using best scientific and diagnostic practices.

1. Diagnose: Check the Log Output (Container Service Log)

- In XNAT’s Admin UI, go to the container’s *log* after your run.
  - Do you see any output like `Wrote result to ...` or any errors?
  - Do you see your output file in `/output`? You can often find a “list files” step in logs or manually add one.
- If the logs are not informative,
  modify your entry script to print what is happening:

```python
import os
import sys
import pandas as pd

def main():
    print(f"Python version: {sys.version}")
    print("Args:", sys.argv)
    if len(sys.argv) != 3:
        print("Usage error.")
        sys.exit(1)
```

```
    input_file = sys.argv[1]
    output_dir = sys.argv[2]
    print(f"input_file: {input_file}")
    print(f"output_dir: {output_dir}")
    os.makedirs(output_dir, exist_ok=True)

    inp_base = os.path.basename(input_file)
    inp_root, _ = os.path.splitext(inp_base)

    output_path = os.path.join(output_dir, f"result_{inp_root}.csv")
    print(f"Output path: {output_path}")

    try:
        if not os.path.exists(input_file):
            print("Input file not found!")
            result = pd.DataFrame([{"status": "failed", "reason": "file not found"}])
        elif os.stat(input_file).st_size == 0:
            print("Input file is empty!")
            result = pd.DataFrame([{"status": "failed", "reason": "file is empty"}])
        else:
            df = pd.read_csv(input_file)
            res = {"status": "success", "rows": df.shape[0], "columns": df.shape[1]}
            result = pd.DataFrame([res])
        result.to_csv(output_path, index=False)
        print("Result CSV written.")
        print("Files in output dir:", os.listdir(output_dir))
    except Exception as e:
        print("Exception:", e)
        pd.DataFrame([{"status": "failed", "reason": str(e)}]).to_csv(output_path,
index=False)

if __name__ == "__main__":
    main()
```

- This way, you see exactly what is written and where.

2. Diagnose: Manually Test Your Image and Script

On your development machine (outside XNAT!):

```bash
mkdir /tmp/in /tmp/out
echo -e "a,b\n1,2" > /tmp/in/sample.csv

# Build your image and run it manually:
docker run --rm -v /tmp/in:/input -v /tmp/out:/output yourimage:tag python3
/app/yourscript.py /input/sample.csv /output

ls /tmp/out
cat /tmp/out/result_sample.csv
```

Do you see the right file?
- If yes: your script and Dockerfile are correct.
- If no: script or Dockerfile needs fixing.

3. Specify Correct Paths in `command.json`

For your script above, **the output file is named** `result_<input_basename>.csv`.
So your output config MUST be:

```json
"outputs": [
  {
    "name": "result_file",
    "type": "file",
    "description": "Result CSV output",
    "mount": "output",
    "path": "result_*.csv"
  }
]
```

Or, if wildcards are problematic in your XNAT instance, fall back to `"path": "*.csv"`.

4. XNAT Resource / Output Handler Configuration Consistency

- Ensure your *wrapper* in the command plugin uploads from the correct resource directory and output handler matches the path above.
- Be sure the output handler **label** exactly matches what is expected if you look in the XNAT web UI under the session/scan you launched on.

Troubleshooting Table

Step	Action	What to Look For
Container Log	View log in XNAT admin	Output errors, print outputs, file not found
Manual Run	Test with `docker run -v ...` locally	Output file in mount, correct naming
Output config	`command.json` outputs:path matches generated filename	Uses wildcard if needed
Output handler	Resource/label matches wrapper's destination	Matched names

---

If after all this, **XNAT still does not upload anything**:

- Copy and paste the *relevant part of your container log output* (especially lines containing file paths and errors).
- Double check: is your output directory mounted as `/output` as the script expects?
- Make sure you launch the *correct version* of your container (no old version cached).
- Make sure your input file is actually copied into `/input` and matches what you expect (you can print `os.listdir("/input")` in your script).

Summary
- The script as you posted can be made robust with added print-debug.
- If the script works locally but not in XNAT, it’s almost always:
  a) path mismatch (output not matching command.json),
  b) permission/mount issue,

c) or error handling.
- Debug with log output and local runs before focusing on XNAT-specific config.
- If stuck, post your *command.json* output section and (sanitized) container logs.

Feel free to paste your log output here and I'll help you interpret it step-by-step!

Bousfiha, Tanae:

the conatiner is auf complet and th estdout are empty so there is no warning sighns

HAWKI (gpt-4.1):

Thank you for your clear description.
If the **container is marked as "complete"** and **stdout is empty** (no logs at all), but XNAT uploads nothing, then **one or more of the following** is happening:

1. The Script Ran Without Error but Did Not Write Any Result to the Output Directory

This can happen if:

- The input file does not exist inside the container (so your script does nothing visible and quits).
- The output filename or path is not as XNAT expects, so even if a file is created, XNAT does not recognize and upload it.

2. Command.json Output **Path** Does Not Match Produced Filename

XNAT will only upload output files that:
- Are in **/output** (unless your mount is somewhere else) **AND**
- Match the **"path"** pattern in command.json output section (`result.csv`, `result_*.csv`, `*.csv`, etc.)

Minimal Diagnostic and Fix Steps

1. Add Debugging Output to Your Script

Modify your script to **always print what is in the `/input` and `/output` directories** and to **print errors**! For example (add prints at the start and end):

```python
import os
import sys
import pandas as pd

def main():
    print("Script started...")
    print("Input directory listing:", os.listdir("/input") if os.path.isdir("/input") else "no /input dir")
    print("Output directory listing:", os.listdir("/output") if os.path.isdir("/output") else "no /output dir")
    if len(sys.argv) != 3:
        print("Argument error: got", sys.argv)
        sys.exit(1)

    input_file = sys.argv[1]
    output_dir = sys.argv[2]
    os.makedirs(output_dir, exist_ok=True)

    inp_base = os.path.basename(input_file)
    inp_root, _ = os.path.splitext(inp_base)
    output_path = os.path.join(output_dir, f"result_{inp_root}.csv")
    print(f"Input file path: {input_file}")
    print(f"Output path: {output_path}")

    try:
        if not os.path.exists(input_file):
            print("Input file not found!")
            result = pd.DataFrame([{"status": "failed", "reason": "file not found"}])
        elif os.stat(input_file).st_size == 0:
            print("Input file is empty!")
            result = pd.DataFrame([{"status": "failed", "reason": "file is empty"}])
        else:
```

```
            df = pd.read_csv(input_file)
            res = {"status": "success", "rows": df.shape[0], "columns": df.shape[1]}
            result = pd.DataFrame([res])
        result.to_csv(output_path, index=False)
        print("Output file written:", output_path)
    except Exception as e:
        print("Exception during processing:", str(e))
        pd.DataFrame([{"status": "failed", "reason": str(e)}]).to_csv(output_path,
index=False)

    print("Final output directory content:", os.listdir(output_dir))

if __name__ == "__main__":
    main()
```

Now, after running **check the logs** in XNAT: you will see exactly what happens.

---

2. Check Your `command.json` Output Section

Your output section **must match the script!**
If your script writes `/output/result_inputfilename.csv`, then use:

```json
"outputs": [
  {
    "name": "result_file",
    "type": "file",
    "description": "Result CSV output",
    "mount": "output",
    "path": "*.csv"
  }
]
```
Or:
```json
```

"path": "result_*.csv"
```

If you still have `"path": "result.csv"`, **XNAT will not pick up `result_inputfilename.csv`**.

---

## **3. Test Locally**

**Always test the container on your machine** before running via XNAT!

```sh
# Example
mkdir /tmp/in /tmp/out
echo -e "a,b\n1,2" > /tmp/in/testfile.csv
docker run --rm -v /tmp/in:/input -v /tmp/out:/output <yourimage>:<tag> python3 /app/yourscript.py /input/testfile.csv /output
ls /tmp/out
cat /tmp/out/result_testfile.csv
```
- If this works, your script is good.

---

4. Permissions / File Ownership

- XNAT Docker Service sometimes uses non-root users; make sure your script **does not use root-specific permissions** and writes as current user.

---

5. Check Input File Presence in the Container

- If your input path is incorrect (`/input/filename.csv` doesn't exist!), nothing is processed and no output is written.
- In XNAT log, you'll see this now because of your debug prints.
```

Common Pitfall

You are NOT required to use "result.csv" as output!!!
You can use any consistent wildcard or name—just make sure the script+json match.

Next Steps

1. Update & run the debug-version of your script.
2. Launch a test in XNAT.
3. Download and **paste the log output here** (text only!)—it will tell us where it fails:

   - Does the input file exist?
   - Does the script crash?
   - What gets written to /output?
4. Also, paste the content of your current command.json `outputs` section.

With this information, I can give you a precise academic-style diagnosis and proven solution.
The process is about matching the output path and ensuring the script does what you expect.
If everything looks correct in logs and still fails—there may be a bug in your XNAT setup; but verify the above first.

Feel free to paste the logs and your output block to proceed!