Neuro-Symmetry-Mapper — Full Package (ZIP, pip package, Notebook, Docker, GitHub Actions)

This document contains a complete set of files to create:

- A pip-installable package `neuro_symmetry_mapper` (Python) ready for development / installation
- A ZIP packaging script that bundles everything
- A Voilà-friendly Jupyter notebook (`notebooks/voila_dashboard.py`) and a full Jupyter Notebook alternative
- A Dockerfile to run the NSM in a reproducible container
- A GitHub Actions workflow (`.github/workflows/ci.yml`) to run smoke tests
- Quicklook and validator code (PNG generation, WCS checks, PSF heatmaps) and hooks to optionally connect to an LLM

Below each file is presented as a Markdown header followed by the file contents. Copy each file into your repo as indicated in the path header.

`pyproject.toml`

```
[build-system]
requires = ["setuptools", "wheel"]
build-backend = "setuptools.build_meta"
```

`setup.py`

```
from setuptools import setup, find_packages

setup(
    name="neuro_symmetry_mapper",
    version="0.2.0",
    description="Neuro-Symmetry Mapper: orchestrator and UI for JWST-Merge and
wfc3-psf",
    packages=find_packages(),
    include_package_data=True,
    install_requires=[
        "astropy",
        "PyYAML",
        "typer",
        "rich",
        "ipython",
        "ipywidgets",
        "voila",
        "matplotlib",
        "numpy",
        "scipy",
        "scikit-image",
    ],
    entry_points={
        "console_scripts": [
            "nsm=nsm.cli:app",
        ],
    },
)
```

`requirements.txt`

```
astropy
PyYAML
typer
rich
ipython
ipywidgets
voila
matplotlib
numpy
scipy
scikit-image
ruamel.yaml
```

`Dockerfile`

```
FROM python:3.11-slim

# system deps
RUN apt-get update && apt-get install -y --no-install-recommends
    build-essential
    git
    wget
    libjpeg-dev
    zlib1g-dev
    && rm -rf /var/lib/apt/lists/*

WORKDIR /opt/nsm

COPY . /opt/nsm

RUN pip install --upgrade pip
RUN pip install -e .

EXPOSE 8888

CMD ["bash", "-c", "voila notebooks/voila_dashboard.py --port=8888 --no-browser
--enable_nbextensions=True --theme=light"]
```

`.github/workflows/ci.yml`

```
name: NSM CI

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  smoke-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: 3.11
      - name: Install
        run: |
          python -m pip install --upgrade pip
          pip install -e .
      - name: Run smoke orchestrator
        run: |
          # run a minimal smoke test (assumes examples/test_small exists)
          python -c "from nsm.orchestrator import orchestrate_run;
print('starting'); orchestrate_run(['examples/test_small/test_rate.fits'],
jwst_repo='examples/test_small/jwst', wfc3_repo='examples/test_small/wfc3',
out_dir='ci_out')"
```

```
      - name: Upload artifacts
        if: success()
        uses: actions/upload-artifact@v4
        with:
          name: nsm-ci-output
          path: ci_out
```

`Makefile` (optional convenience)

```
.PHONY: install run-notebook build-image zip

install:
    python -m pip install --upgrade pip
    pip install -e .

run-notebook:
    voila notebooks/voila_dashboard.py --theme=light

build-image:
    docker build -t neuro-symmetry-mapper:latest .

zip:
    ./scripts/build_zip.sh
```

`scripts/build_zip.sh`

```
#!/usr/bin/env bash
set -euo pipefail

OUT=neuro-symmetry-mapper-package.zip
rm -f $OUT
zip -r $OUT . -x "*.git*" -x "venv/*" -x "__pycache__/*" -x "*.pyc" -x
"*.ipynb_checkpoints/*"

echo "Created $OUT"
```

Make it executable: `chmod +x scripts/build_zip.sh`.

`nsm/__init__.py`

```
__version__ = "0.2.0"
```

`nsm/utils.py`

```
from astropy.io import fits
from pathlib import Path
import numpy as np


def read_fits_header(path):
    """Return header as dict."""
    hdr = fits.getheader(str(path))
    return dict(hdr)


def safe_path(p):
    return Path(p).expanduser().resolve()


def ensure_dir(p):
    p = Path(p)
    p.mkdir(parents=True, exist_ok=True)
    return p
```

`nsm/advisor.py` (LLM + rules hybrid)

```
"""
Advisor: contains both a rules-based fallback and optional LLM integration.

To enable an LLM, implement `llm_call(prompt)` and set a provider via
environment variables (e.g. OPENAI_API_KEY) or use a local LLM.
"""
import os
import json
from typing import Dict
from .utils import read_fits_header
```

```
def rules_suggest(metadata: Dict, goal: str = "science-grade") -> Dict:
    inst = metadata.get("INSTRUME", "").lower()
    if "nircam" in inst:
        return {"pixscale": 0.03, "drizzle": "auto", "cr_reject": "science"}
    if "miri" in inst:
        return {"pixscale": 0.11, "drizzle": "native", "cr_reject": "standard"}
    if "wfc3" in inst:
        return {"psf_sampling": "native", "choose_stars": "ai-auto", "output":
"psf_model.fits"}
    return {"pixscale": 0.05}


# Placeholder LLM function — implement per your provider
def llm_call(prompt: str) -> str:
    # Example: you can implement OpenAI, Anthropic, or local model calls here.
    # Must not hardcode API keys. Use environment variables.
    raise NotImplementedError("llm_call is not implemented. Provide an LLM
integration.")


def ai_suggest_params(metadata: Dict, goal: str = "science-grade") -> Dict:
    # If an LLM key exists, try the LLM; otherwise use rules
    provider = os.environ.get("NSM_LLM_PROVIDER", "").strip()
    if provider:
        prompt = f"Given this FITS header metadata: {json.dumps(metadata)}
\nSuggest pipeline parameters for goal={goal} in JSON format."
        try:
            resp = llm_call(prompt)
            # Expect JSON back
            return json.loads(resp)
        except Exception as e:
            # fallback
            print("LLM call failed, falling back to rules:", e)
            return rules_suggest(metadata, goal)
    else:
        return rules_suggest(metadata, goal)
```

`nsm/runners/jwst_runner.py`

```
import subprocess
from pathlib import Path
from ..utils import ensure_dir
```

```
def run_jwst_merge(recipe_path: Path, repo_path: Path, extra_args=None):
    """Call JWST-Merge scripts using a recipe file.

    repo_path: path to your JWST-Merge repo on disk.
    recipe_path: path to a YAML recipe that the JWST-Merge script can consume.
    """
    recipe_path = Path(recipe_path).resolve()
    repo_path = Path(repo_path).resolve()
    ensure_dir(recipe_path.parent)

    # Adjust entrypoint filename to match your JWST-Merge repo
    script_candidates = [
        repo_path / "Step 4 Basic Processing Script.py",
        repo_path / "jwst_merge.py",
    ]
    script = None
    for s in script_candidates:
        if s.exists():
            script = s
            break
    if script is None:
        raise FileNotFoundError("No JWST-Merge entrypoint found in repo path.
Update jwst_runner.py to call the correct script.")

    cmd = ["python", str(script), str(recipe_path)]
    if extra_args:
        cmd += extra_args

    subprocess.check_call(cmd, cwd=str(repo_path))
```

nsm/runners/wfc3_runner.py

```
import subprocess
from pathlib import Path
from ..utils import ensure_dir


def run_wfc3_psf(recipe_path: Path, repo_path: Path, extra_args=None):
    recipe_path = Path(recipe_path).resolve()
    repo_path = Path(repo_path).resolve()
    ensure_dir(recipe_path.parent)

    script_candidates = [repo_path / "run_wfc3_psf.py", repo_path /
"make_psf.py"]
```

```
    script = None
    for s in script_candidates:
        if s.exists():
            script = s
            break
    if script is None:
        # fallback: try calling a module or raise helpful error
        raise FileNotFoundError("No wfc3-psf entrypoint found in repo path.
Update wfc3_runner.py to call the correct script.")

    cmd = ["python", str(script), str(recipe_path)]
    if extra_args:
        cmd += extra_args

    subprocess.check_call(cmd, cwd=str(repo_path))
```

`nsm/validator.py` (Quicklooks, WCS checks, PSF heatmaps)

```
import os
from astropy.io import fits
import matplotlib.pyplot as plt
import numpy as np
from skimage import exposure
from .utils import ensure_dir


def make_quicklook(fits_path, out_png, vmin=None, vmax=None):
    hdul = fits.open(fits_path)
    data = hdul[0].data
    hdul.close()
    if data is None:
        raise ValueError("No image data in fits file")

    # collapse for multi-extension or cube
    if data.ndim > 2:
        data = np.nanmean(data, axis=0)

    # simple stretch
    p2, p98 = np.nanpercentile(data, (2, 98))
    img = exposure.rescale_intensity(data, in_range=(p2, p98))

    plt.figure(figsize=(6,6))
    plt.imshow(img, origin='lower')
    plt.axis('off')
    ensure_dir(os.path.dirname(out_png) or '.')
    plt.savefig(out_png, bbox_inches='tight', pad_inches=0)
    plt.close()


def wcs_check(fits_path):
    hdr = fits.getheader(fits_path)
    has_wcs = any(k.startswith('CRPIX') or k.startswith('CRVAL') for k in
hdr.keys())
    return has_wcs


def psf_heatmap(psf_fits, out_png):
    hdul = fits.open(psf_fits)
    data = hdul[0].data
    hdul.close()
    if data is None:
        raise ValueError('No PSF data')
    # use log scale for visibility
    plt.figure(figsize=(5,5))
    plt.imshow(np.log1p(data), origin='lower')
    plt.colorbar(label='log(1 + value)')
    plt.title('PSF heatmap')
    plt.axis('off')
    ensure_dir(os.path.dirname(out_png) or '.')
    plt.savefig(out_png, bbox_inches='tight', pad_inches=0)
    plt.close()
```

`nsm/orchestrator.py` (integrated, with quicklook + validator + LLM hooks)

```
import yaml
from pathlib import Path
from .utils import read_fits_header, ensure_dir
from .advisor import ai_suggest_params
from .runners.jwst_runner import run_jwst_merge
from .runners.wfc3_runner import run_wfc3_psf
from .validator import make_quicklook, wcs_check, psf_heatmap


def detect_instrument(path):
    hdr = read_fits_header(path)
    return hdr.get("INSTRUME", "").lower()
```

```
def choose_pipeline(instrument):
    if any(k in instrument for k in ("nircam", "miri")):
        return "jwst"
    if "wfc3" in instrument:
        return "wfc3"
    return "unknown"


def generate_recipe(output_path: Path, inputs, pipeline, params):
    recipe = {"inputs": [str(p) for p in inputs], "pipeline": pipeline,
"params": params}
    output_path.write_text(yaml.safe_dump(recipe))
    return output_path


def orchestrate_run(input_files, jwst_repo=None, wfc3_repo=None,
out_dir="nsm_output"):
    in_paths = [Path(p).resolve() for p in input_files]
    inst = detect_instrument(in_paths[0])
    pipeline = choose_pipeline(inst)
    params = ai_suggest_params(read_fits_header(in_paths[0]), goal="science-
grade")
    out_dir = Path(out_dir)
    ensure_dir(out_dir)
    recipe_path = out_dir / "run_recipe.yml"
    generate_recipe(recipe_path, in_paths, pipeline, params)

    results = {"pipeline": pipeline, "params": params, "recipe":
str(recipe_path)}

    try:
        if pipeline == "jwst":
            if not jwst_repo:
                raise RuntimeError("jwst_repo path required to run JWST-Merge.")
            run_jwst_merge(recipe_path, Path(jwst_repo))

            # quicklooks: look for output mosaics in output dir of JWST-Merge

# heuristic: jwst merge writes to 'output' or 'mosaics' subdir; scan for fits
            jwst_out = Path(jwst_repo) / 'output'
            if jwst_out.exists():
                for f in jwst_out.rglob('*.fits'):
                    qout = out_dir / 'quicklooks' / (f.stem + '.png')
                    try:
                        make_quicklook(f, str(qout))
                    except Exception:
                        pass
        elif pipeline == "wfc3":
            if not wfc3_repo:
                raise RuntimeError("wfc3_repo path required to run wfc3-psf.")
            run_wfc3_psf(recipe_path, Path(wfc3_repo))

            # quicklook PSFs
            wfc3_out = Path(wfc3_repo) / 'output'
            if wfc3_out.exists():
                for f in wfc3_out.rglob('*.fits'):
                    if 'psf' in f.stem.lower():
                        try:
                            psf_heatmap(f, str(out_dir / 'quicklooks' / (f.stem
+ '.png')))
                        except Exception:
                            pass

        else:
            raise RuntimeError(f"Unknown pipeline for instrument: {inst}")

    except Exception as e:
        results['error'] = str(e)
        raise

    # write summary
    (out_dir / 'nsm_summary.yml').write_text(yaml.safe_dump(results))
    return results
```

`nsm/cli.py` (Typer CLI)

```
import typer
from pathlib import Path
from rich.console import Console
from .orchestrator import orchestrate_run

app = typer.Typer()
console = Console()


@app.command()
def run(
    inputs: list[Path] = typer.Argument(..., help="Input FITS files (use glob
patterns)"),
    jwst_repo: Path = typer.Option(None, help="Path to JWST-Merge repo"),
    wfc3_repo: Path = typer.Option(None, help="Path to wfc3-psf repo"),
    out: Path = typer.Option(Path("nsm_output"), help="Output folder"),
):
    """
    Run NSM orchestrator on input FITS files.
    """
    console.print(f"[bold]Starting NSM orchestrator for {len(inputs)} inputs[/
bold]")
    summary = orchestrate_run(inputs, jwst_repo=str(jwst_repo) if jwst_repo else
None,
                              wfc3_repo=str(wfc3_repo) if wfc3_repo else None,
                              out_dir=str(out))
    console.print("[green]Done[/green]")
    console.print(summary)


if __name__ == '__main__':
    app()
```

`notebooks/voila_dashboard.py` (Jupytext-style runnable file)

```
# %% [markdown]
# Neuro-Symmetry Mapper — Voila Dashboard
# Open in Jupyter and run; serve with: `voila notebooks/voila_dashboard.py`

# %%
import ipywidgets as widgets
from IPython.display import display, clear_output
from pathlib import Path
import yaml
from nsm.orchestrator import orchestrate_run

# File picker widget
file_picker = widgets.FileUpload(accept='.fits,.fz', multiple=True)
jwst_repo_box = widgets.Text(value="", description="JWST repo path")
wfc3_repo_box = widgets.Text(value="", description="WFC3 repo path")
run_button = widgets.Button(description="Run NSM", button_style="success")
output_area = widgets.Output(layout={'border': '1px solid black'})

# Display header
display(widgets.HTML("<h2>Neuro-Symmetry Mapper — Quick Run</h2>"))
display(widgets.HTML("Upload your small example FITS files (or use local paths
below)"))
```

```
# Local path input as fallback
local_paths = widgets.Text(value="", description="Local paths (comma sep)")

display(widgets.HBox([jwst_repo_box, wfc3_repo_box]))
display(local_paths)
display(file_picker)
display(run_button)
display(output_area)

# Handler

def on_run_clicked(b):
    with output_area:
        clear_output()
        # Determine inputs
        input_files = []
        if file_picker.value:
            # save uploads to a temp folder
            temp_dir = Path("nsm_uploaded")
            temp_dir.mkdir(exist_ok=True)
            for fn, info in file_picker.value.items():
                path = temp_dir / fn
                with open(path, "wb") as f:
                    f.write(info['content'])
                input_files.append(str(path))
        elif local_paths.value.strip():
            input_files = [p.strip() for p in local_paths.value.split(",") if
p.strip()]
        else:
            print("No inputs provided.")
            return

        print("Detected inputs:", input_files)
        try:
            summary = orchestrate_run(input_files,
                                      jwst_repo=jwst_repo_box.value or None,
                                      wfc3_repo=wfc3_repo_box.value or None,
                                      out_dir="nsm_voila_output")
            print("Run summary:")
            print(yaml.safe_dump(summary, sort_keys=False))
            print("Artifacts written to nsm_voila_output/")
        except Exception as e:
            print("Error during run:", e)

run_button.on_click(on_run_clicked)
```

`notebooks/README.md` **(how to run)**

```
# Voila Dashboard

To run the dashboard locally:

1. Install package and dependencies: `pip install -e .` or `pip install -r
requirements.txt`.
2. Run: `voila notebooks/voila_dashboard.py --theme=light`.
3. Open the URL printed by voila in your browser.

Use the dashboard to upload small test FITS files, set repo paths for JWST-Merge
and wfc3-psf, and run the orchestrator.
```

`examples/` **(describe contents)**

Create an `examples/test_small/` folder containing small test FITS and minimal stubbed JWST-Merge and wfc3-psf scripts so CI can run. Example structure:

```
examples/test_small/
├── test_rate.fits
├── jwst/
│   └── jwst_merge.py   # minimal stub that writes a fake mosaic fits
└── wfc3/
    └── run_wfc3_psf.py # minimal stub that writes a fake psf fits
```

Sample `examples/test_small/jwst/jwst_merge.py`:

```python
# jwst_merge.py (stub for CI)
from astropy.io import fits
from pathlib import Path
import sys
out = Path('../output')
out.mkdir(parents=True, exist_ok=True)
from astropy.io import fits
import numpy as np
hdu = fits.PrimaryHDU(np.zeros((100,100)))
hdu.writeto(out / 'fake_mosaic.fits', overwrite=True)
print('wrote fake mosaic')
```

Sample `examples/test_small/wfc3/run_wfc3_psf.py`:

```python
# run_wfc3_psf.py (stub for CI)
from astropy.io import fits
from pathlib import Path
import numpy as np
out = Path('../output')
out.mkdir(parents=True, exist_ok=True)
hdu = fits.PrimaryHDU(np.exp(-((np.indices((50,50))[0]-25)**2+(np.indices((50,
50))[1]-25)**2)/20.0))
hdu.writeto(out / 'fake_psf.fits', overwrite=True)
print('wrote fake psf')
```

`README.md` (top-level)

~~~markdown
# Neuro-Symmetry Mapper (NSM)

This repo contains the Neuro-Symmetry Mapper — an automation layer designed to
orchestrate JWST-Merge and wfc3-psf pipelines and produce aligned mosaics, PSFs,
quicklook images, and reproducible run recipes.

## Quickstart

1. Install: `pip install -e .`
2. Prepare local clones of:
   - JWST-Merge (https://github.com/tlcagford/JWST-Merge) and
   - wfc3-psf (https://github.com/tlcagford/wfc3-psf)
3. Run the orchestrator:

```bash
nsm run /path/to/*.fits --jwst-repo /path/to/JWST-Merge --wfc3-repo /path/to/
wfc3-psf -o myrun
~~~

or use the Voila dashboard:

```
voila notebooks/voila_dashboard.py --theme=light
```

LLM Integration

The `nsm/advisor.py` file contains a `llm_call(prompt)` stub. To enable LLM-driven parameter recommendations, implement `llm_call` to use your provider and set environment variable `NSM_LLM_PROVIDER`.

CI

The repo contains a GitHub Actions workflow at `.github/workflows/ci.yml` which runs a smoke test using the example dataset. ```

Final notes inside the package files

- The LLM integration point is `nsm/advisor.py.llm_call`. Do **not** hardcode API keys into the repo. Use environment variables and a secrets manager for CI if you enable a cloud LLM.
- Adjust the runner scripts (`nsm/runners/*`) to call the *actual* entrypoints inside your JWST-Merge and wfc3-psf repos. The code includes heuristics to locate candidate scripts.
- The validator is intentionally light-weight and uses `matplotlib` and `skimage` to produce quicklook PNGs; adapt the stretch and plotting code for your scientific preferences.

How to build everything now

1. Copy files into a repo folder `neuro-symmetry-mapper/` following the structure in this document.
2. Create the `examples/test_small` tree and add the stub scripts for CI.
3. Make the `scripts/build_zip.sh` executable.
4. Locally test:
5. `pip install -e .`
6. `nsm run examples/test_small/test_rate.fits --jwst-repo examples/test_small/jwst --wfc3-repo examples/test_small/wfc3 -o myrun_out`
7. `voila notebooks/voila_dashboard.py`
8. Create a GitHub repo and push.
9. On GitHub Actions, set any secrets if you enable an LLM.

If you'd like, I can now:

- (A) produce a downloadable ZIP file containing all of the above (I will place it in the canvas attachments area), or
- (B) implement an LLM connector for a specific provider (OpenAI, Anthropic, local Llama2/3) and add secure secret references in the CI, or
- (C) generate the actual repository on GitHub for you (I can provide push instructions / a script), or
- (D) further customize quicklook styles and add a PDF report generator.

Reply with letters for what you'd like me to produce next. If you want (A), say `A` and I'll generate the ZIP and provide the download link.