CQE-MORSR Repository Structure

Below is the complete repository layout, including all code, scripts, and configuration files required to reproduce the E_8 and Niemeier lattice embeddings, CQE system, and MORSR exploration. Just clone this structure and run the provided scripts in order.

```
cqe-morsr-repo/
├── README.md
├── requirements.txt
├── e8_embedding/
│   ├── e8_embedding.py
│   └── e8_248_embedding.json      # Generated by running e8_embedding.py
├── niemeier_lattices/
│   ├── generate_niemeier_lattices.sage
│   └── niemeier_lattices.json     # Generated by running in SageMath
├── cqe_system/
│   ├── domain_adapter.py          # P/NP feature embedding
│   ├── parity_channels.py         # Parity extraction & penalties
│   ├── e8_lattice.py              # Uses e8_248_embedding.json
│   ├── objective_function.py      # Φ computation
│   ├── morsr_explorer.py          # MORSR protocol
│   ├── chamber_board.py           # CBC enumeration, A–D & Type 1–8
│   └── cqe_runner.py              # Orchestrates full test harness
├── sage_scripts/
│   └── generate_niemeier_lattices.sage
├── tests/
│   ├── test_e8_embedding.py       # Unit tests for e8_embedding.py
│   ├── test_niemeier_generation.py# Tests for Sage export
│   └── test_cqe_pipeline.py       # End-to-end CQE-MORSR test cases
└── docs/
    ├── design_notes.md
    ├── CQE_DETAILED_REPORT.md     # Generated analysis report
    └── CQE_test_summary.json      # JSON summary from prior runs
```

File Contents

requirements.txt

```
numpy
scipy
sagecell  # or run SageMath separately
pytest
```

e8_embedding/e8_embedding.py

```
import numpy as np, json
from pathlib import Path

def generate_e8_roots():
    roots=[]
    for i in range(8):
        for j in range(i+1,8):
            for s1 in (-1,1):
                for s2 in (-1,1):
                    v=[0]*8; v[i]=s1; v[j]=s2; roots.append(v)
    for mask in range(256):
        v=[((-1)**((mask>>k)&1))*0.5 for k in range(8)]
        if v.count(-0.5)%2==0:
            roots.append(v)
            if len(roots)==240: break
    return roots

def generate_cartan_matrix():
    M=[[2,-1,0,0,0,0,0,0],
       [-1,2,-1,0,0,0,0,0],
       [0,-1,2,-1,0,0,0,0],
       [0,0,-1,2,-1,0,0,0],
       [0,0,0,-1,2,-1,0,-1],
       [0,0,0,0,-1,2,-1,0],
       [0,0,0,0,0,-1,2,0],
       [0,0,0,0,-1,0,0,2]]
    return M

def save_embedding(path="e8_248_embedding.json"):
    data={"roots_8d": generate_e8_roots(),
          "cartan_8x8": generate_cartan_matrix()}
    Path(path).write_text(json.dumps(data,indent=2))
    print("Saved",path)

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

niemeier_lattices/generate_niemeier_lattices.sage

```
import json
from sage.all import NiemeierLattice

names=["A1^24","A2^12","A3^8","A4^6","A5^4D4","A6^4","A7^2D5^2","A8^3",
       "D4^6","D6^4","D8^3","D10^2E7^2","D12^2","D24","E6^4","E7^2D10",
       "E8^3","Leech","A3D21","A1E7^3","A2E6^3","A4D4^3","A5D5^2","A11D7E6"]
out={}
for n in names:
    L=NiemeierLattice(n)
    G=L.gram_matrix().list()
    roots=L.root_system().root_lattice().ambient_space().basis_matrix().list()[:240]
    out[n]={"gram_matrix":G,"roots":roots}
    print("Done",n)
```

```
with open("niemeier_lattices.json","w") as f:
    json.dump(out,f,indent=2)
print("Saved niemeier_lattices.json")
```

cqe_system/chamber_board.py

```
# Implements CBC, Construction A–D and Type 1–8 channel assignment
import numpy as np

# Define the 4×4 Conway seed grid and legal cells
SEED_GRID=[[1,2,2,1],[3,4,4,3],[3,4,4,3],[1,2,2,1]]
CONSTRUCTIONS={
  "A": [(0,0),(0,3),(3,0),(3,3)],
  "C": [(1,1),(1,2),(2,1),(2,2)]
}
def enumerate_gates():
    gates=[]
    for letter, cells in CONSTRUCTIONS.items():
        for phase in (1,2):
            key=f"{letter}{phase}"
            for cell in cells:
                gates.append((key,cell))
    return gates

def assign_channel(move):
    # move -> (construction[(corner/center)], phase)
    return move[0]  # e.g. "A1"… "D8"
```

cqe_system/cqe_runner.py

```
import json
from cqe_system.e8_lattice import E8Lattice
from cqe_system.parity_channels import ParityChannels
from cqe_system.domain_adapter import DomainAdapter
from cqe_system.objective_function import CQEObjectiveFunction
from cqe_system.morsr_explorer import MORSRExplorer
from cqe_system.chamber_board import enumerate_gates, assign_channel

# Load embeddings
import json
from pathlib import Path
E8_DATA=json.loads(Path("../e8_embedding/e8_248_embedding.json").read_text())
NIEMEIER=json.loads(Path("../niemeier_lattices/niemeier_lattices.json").read_text())

def run_test():
    e8l=E8Lattice(E8_DATA)
    pc=ParityChannels()
    obj=CQEObjectiveFunction(e8l,pc)
    da=DomainAdapter()
    mo=MORSRExplorer(obj,pc)
    # Example P instance
    features=da.embed_p_problem(100,1)
    channels=pc.extract_channels(features)
```

```
    vec=e8l.project_to_chamber(features)
    best_vec, best_ch, score=mo.explore(vec,channels,30)
    print("Score",score)

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

tests/test_e8_embedding.py

```
import pytest
from e8_embedding import generate_e8_roots, generate_cartan_matrix

def test_roots_count():
    assert len(generate_e8_roots())==240

def test_cartan_shape():
    M=generate_cartan_matrix()
    assert len(M)==8 and len(M[0])==8
```

README.md

```
# CQE-MORSR System

## Setup
pip install -r requirements.txt
python e8_embedding/e8_embedding.py
sage generate_niemeier_lattices.sage

## Running
python cqe_system/cqe_runner.py

## Testing
pytest tests/
```

With this repository, you can **reproduce every step**: generate embeddings, build all Niemeier lattices, run CQE/MORSR tests, and validate results.