🧬 Biomedical Dataset Downloader Notebook

This notebook downloads or connects to the **SIDER**, **ChEMBL**, and **TDC** datasets (public access) and creates placeholders for **MIMIC-IV** integration (requires PhysioNet credentials).

```
# Cell 1 — Setup & Imports
import os
import pandas as pd
import requests

print("Environment ready.")
```

```
Environment ready.
```

🏥 1. MIMIC-IV (Requires PhysioNet Access)

This dataset cannot be directly downloaded without approval.

Steps (manual):

1. Visit https://physionet.org/content/mimiciv/
2. Create an account and complete the **CITI training**.
3. Sign the **Data Use Agreement**.
4. After approval, download with:

   ```
   wget -r -N -c -np https://physionet.org/files/mimiciv/2.2/
   ```

5. You can then link the downloaded CSV files to your local path below.

```
# Placeholder: Connect to local MIMIC-IV files (after manual download)
mimic_path = "path_to_your_local_mimiciv_folder"
if os.path.exists(mimic_path):
    print("MIMIC-IV data available at:", mimic_path)
else:
    print("MIMIC-IV data not found. Please set the correct path.")
```

```
MIMIC-IV data not found. Please set the correct path.
```

💊 2. SIDER (Side Effect Resource)

Publicly available dataset containing drug–side effect associations.

```
# Cell 2 — Download SIDER dataset
sider_url = "http://sideeffects.embl.de/media/download/meddra_all_se.tsv.gz"
os.makedirs("datasets", exist_ok=True)
sider_path = "datasets/sider_meddra_all_se.tsv.gz"

if not os.path.exists(sider_path):
    print("Downloading SIDER dataset...")
    r = requests.get(sider_url)
    open(sider_path, 'wb').write(r.content)
    print("✅ SIDER downloaded successfully!")
else:
    print("SIDER already exists.")

# Load example
sider = pd.read_csv(sider_path, sep="\t", compression="gzip", header=None)
```

```
print("SIDER sample:")
print(sider.head())
```

```
Downloading SIDER dataset...
✅ SIDER downloaded successfully!
SIDER sample:
              0             1         2    3         4                     5
0  CID100000085  CID000010917  C0000729  LLT  C0000729       Abdominal cramps
1  CID100000085  CID000010917  C0000729   PT  C0000737         Abdominal pain
2  CID100000085  CID000010917  C0000737  LLT  C0000737         Abdominal pain
3  CID100000085  CID000010917  C0000737   PT  C0687713  Gastrointestinal pain
4  CID100000085  CID000010917  C0000737   PT  C0000737         Abdominal pain
```

🧪 3. ChEMBL Bioactivity Data (via API)

You can access ChEMBL directly via the official API.

```
# Cell 3 — ChEMBL API access
!pip install chembl_webresource_client -q

from chembl_webresource_client.new_client import new_client

molecule = new_client.molecule
res = molecule.filter(pref_name__icontains="aspirin")
print("Example ChEMBL record:")
print(res[0])
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.2/55.2 kB 1.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.4/61.4 kB 4.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 70.7/70.7 kB 2.6 MB/s eta 0:00:00
Example ChEMBL record:
{'atc_classifications': ['B01AC06', 'N02BA01', 'N02BA51', 'A01AD05', 'N02BA71'], 'availability_type':
```

🔬 4. TDC (Therapeutics Data Commons)

Public datasets for drug discovery, repurposing, and ADMET prediction.

```
# Cell 4 — TDC Dataset Example
!pip install PyTDC -q

from tdc.multi_pred import DrugRes
data = DrugRes(name='GDSC1')
split = data.get_split()

print("TDC Dataset: GDSC1 loaded")
print("Available splits:", split.keys())
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.7/43.7 kB 3.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 151.2/151.2 kB 11.8 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 151.2/151.2 kB 12.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 151.1/151.1 kB 13.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 151.1/151.1 kB 13.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 151.1/151.1 kB 12.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
```

```
Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 148.2/148.2 kB 13.1 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 146.6/146.6 kB 11.8 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 146.8/146.8 kB 12.5 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 145.6/145.6 kB 12.4 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 144.4/144.4 kB 11.8 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 143.7/143.7 kB 11.2 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 141.5/141.5 kB 11.3 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━  46.8/46.8 kB 3.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.2/3.2 MB 83.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.6/55.6 kB 4.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.1/43.1 MB 31.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.0/18.0 MB 79.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 34.8/34.8 MB 20.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.2/21.2 MB 69.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 256.2/256.2 kB 21.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 172.3/172.3 kB 13.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.7/46.7 kB 3.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 34.1/34.1 MB 19.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 201.0/201.0 kB 16.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 81.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 86.0/86.0 kB 7.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.2/58.2 kB 4.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 76.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 88.8/88.8 kB 7.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 276.4/276.4 kB 23.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.1/14.1 MB 76.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.8/8.8 MB 97.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 88.0/88.0 kB 8.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 80.0/80.0 kB 6.9 MB/s eta 0:00:00
Building wheel for PyTDC (setup.py) ... done
```

✅ Summary

Dataset	Access	How to Download
MIMIC-IV	Restricted (PhysioNet)	Requires approval
SIDER	Public	Direct HTTP download
ChEMBL	Public	API or FTP
TDC	Public	Python `PyTDC` package

Hybrid Quantum Self-Attention (QSVT-SA) Notebook

This notebook contains runnable skeleton code, diagram generation, and simulation scaffolding to reproduce figures and experiments described in the manuscript:

Hybrid Quantum Self-Attention Model Using QSVT for Real-World Biomedical Drug Prediction.

Sections:

- Setup and environment
- Data preprocessing placeholders (MIMIC-IV, SIDER, ChEMBL, TDC)
- Embedding (BioWordVec) example
- Block-encoding & QSVT skeleton (PennyLane)
- Quantum self-attention overlap calculation
- Diagram generation: architecture flowchart and circuit schematic
- Experiment logging and results placeholders

NOTE: This notebook is a reproducible scaffold. Actual access to MIMIC-IV, SIDER, ChEMBL and TDC requires data agreements/downloads; the cells include instructions and placeholders.

```
# Setup: install required packages (uncomment and run if needed)
# Note: In some environments packages may already be installed.
# !pip install pennylane qiskit matplotlib networkx gensim pandas scikit-learn

# Basic imports
import os
import math
import json
import time
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
import sys

print('Environment ready. Python version:', sys.version)
```

Data placeholders and instructions

This section explains how to prepare datasets. **Do not** attempt to download MIMIC-IV within this notebook unless you have proper credentials.

- **MIMIC-IV:** Request access via PhysioNet and follow the official instructions; extract medication, diagnosis, clinical notes. Map drugs to RxNorm and terms to UMLS.
- **SIDER:** Download SIDER CSVs, map drug names to RxNorm, and extract side-effect text.
- **ChEMBL:** Use ChEMBL webservices or download the database to extract bioactivity (IC50) and compound descriptors.
- **TDC:** Use TDC datasets available on the Therapeutics Data Commons site.

Below we show skeleton code to load CSVs if you have them locally.

```
# Example: load local CSVs (replace paths with your files)
data_folder = 'datasets'  # modify as needed

# Example placeholders - replace with actual file paths
mimic_csv = os.path.join(data_folder, 'mimic_sample.csv')
sider_csv = os.path.join(data_folder, 'sider_sample.csv')
chembl_csv = os.path.join(data_folder, 'chembl_sample.csv')
tdc_csv = os.path.join(data_folder, 'tdc_sample.csv')

print('Paths set. Place your datasets in:', data_folder)
for p in [mimic_csv, sider_csv, chembl_csv, tdc_csv]:
    print(p, 'exists?', os.path.exists(p))
```

Embedding: BioWordVec example (or fallback random embeddings)

If you have BioWordVec embeddings available locally, load them with gensim; otherwise we demonstrate a fallback that creates deterministic random embeddings per token.

```
# Simple token-to-vector embedding example (fallback)
from collections import defaultdict
```

```
def get_dummy_embedding(token, dim=200, seed=42):
    rng = np.random.RandomState(abs(hash(token)) % (2**32 - 1))
    return rng.normal(scale=0.1, size=(dim,))

# Example tokenization and embedding
sample_texts = ['fever cough fatigue', 'headache nausea dizziness']

def text_to_embedding(text):
    tokens = text.split()
    embs = np.array([get_dummy_embedding(tok) for tok in tokens])
    return embs.mean(axis=0)

embs = [text_to_embedding(t) for t in sample_texts]
embs = np.vstack(embs)
print('Embeddings shape:', embs.shape)
```

Covariance computation and block-encoding

Compute covariance from embeddings and illustrate the smaller toy example. For real d=200, follow the same steps but ensure normalization by spectral norm for block-encoding.

```
# Toy covariance example
X = embs  # from above, shape (n_samples, d)
n, d = X.shape
Xc = X - X.mean(axis=0)
Sigma = (Xc.T @ Xc) / (n - 1)
alpha = np.linalg.norm(Sigma, 2)
Sigma_scaled = Sigma / alpha
print('Sigma shape:', Sigma.shape, 'alpha:', alpha)
```

QSVT skeleton using PennyLane (simulated)

This cell provides a schematic implementation of QSVT-like polynomial transformation using PennyLane. It is illustrative: full QSVT requires careful phase sequence design (quantum signal processing). Here we simulate a polynomial filter by repeated application of the block-encoding unitary and single-qubit rotations.

```
# QSVT-like skeleton with PennyLane (simulated)
try:
    import pennylane as qml
    from pennylane import numpy as pnp
    print('PennyLane version:', qml.__version__)
except Exception as e:
    print('PennyLane not available in this environment:', e)
    qml = None

# Define a small example QSVT-like circuit if PennyLane is available
if qml is not None:
    n_data_qubits = 3  # small example
    dev = qml.device('default.qubit', wires=n_data_qubits+1)  # +1 ancilla

    @qml.qnode(dev)
    def qsvt_example(state_prep_angles, phi_angles):
        # state_prep_angles: prepare a data state (Ry rotations)
        # phi_angles: phase angles for single-qubit rotations (placeholder)
        # Prepare data register on wires 1..n
        for i in range(n_data_qubits):
            qml.RY(state_prep_angles[i], wires=i+1)
        # Placeholder block-encoding: controlled rotations using ancilla (wire 0)
```

```
        for i in range(len(phi_angles)):
            # Example: controlled-RZ with ancilla as control (very schematic)
            qml.ctrl(qml.RZ, control=0)(phi_angles[i], wires=(i % n_data_qubits)+1)
        return qml.state()

    # Run example
    angles = [0.3, 0.5, -0.2]
    phis = [0.1]*5
    state = qsvt_example(angles, phis)
    print('Statevector length:', len(state))
```

Quantum self-attention overlap calculation (simulated)

This cell shows how to compute overlaps between amplitude-encoded states and form attention weights. On hardware, overlaps can be estimated via SWAP tests.

```
# Simulated overlap-based attention (classical simulation using numpy)
def amplitude_encode(vec):
    # Normalize and interpret as amplitude vector; pad to power of two
    vec = np.array(vec, dtype=float)
    norm = np.linalg.norm(vec)
    if norm == 0:
        return np.zeros(1)
    amp = vec / norm
    # pad to next power of two
    m = int(2**np.ceil(np.log2(len(amp))))
    if m > len(amp):
        amp = np.concatenate([amp, np.zeros(m - len(amp))])
    return amp

# Example vectors
v1 = amplitude_encode(embs[0])
v2 = amplitude_encode(embs[1])

# Overlap
overlap = np.vdot(v1, v2).real
print('Overlap (simulated):', overlap)
# Softmax attention
def softmax(xs):
    ex = np.exp(xs - np.max(xs))
    return ex / ex.sum()
weights = softmax(np.array([overlap, 0.2]))
print('Attention weights:', weights)
```

Diagram generation: architecture flowchart and schematic

We draw a simple flowchart for the QSVT-SA architecture and a schematic circuit-like figure using matplotlib. The generated PNGs can be inserted into the Word manuscript.

```
# Create a flowchart-like figure using networkx & matplotlib
G = nx.DiGraph()
nodes = ['Data (MIMIC/SIDER/ChEMBL/TDC)', 'Preprocessing & Embedding', 'Covariance Σ', 'Block-encoding
for n in nodes:
    G.add_node(n)
edges = [(nodes[i], nodes[i+1]) for i in range(len(nodes)-1)]
G.add_edges_from(edges)

pos = {
```

```
    nodes[0]:(0,0),
    nodes[1]:(2,0),
    nodes[2]:(4,0),
    nodes[3]:(6,0),
    nodes[4]:(8,0),
    nodes[5]:(10,0),
    nodes[6]:(12,0)
}

plt.figure(figsize=(14,2.5))
nx.draw(G, pos, with_labels=True, node_size=2500, node_color='skyblue', arrowsize=20, font_size=8)
plt.title('QSVT-SA Architecture Flowchart')
outflow_path = 'qsvt_qsa_flowchart.png'
plt.savefig(outflow_path, bbox_inches='tight', dpi=150)
plt.show()
print('Saved flowchart to', outflow_path)
```

```
# Simple schematic of a circuit stack (not a real quantum circuit diagram)
fig, ax = plt.subplots(figsize=(10,3))
y = 0.5
x_positions = [0.5, 2.0, 3.5, 5.0, 6.5]
labels = ['State Prep', 'Block-Encode', 'QSVT Layers', 'Overlap/Attention', 'Measurement']
for x, lab in zip(x_positions, labels):
    rect = plt.Rectangle((x-0.6,y-0.2),1.2,0.4,facecolor='lightgrey',edgecolor='black')
    ax.add_patch(rect)
    ax.text(x, y, lab, ha='center', va='center', fontsize=10)
# arrows
for i in range(len(x_positions)-1):
    ax.annotate('', xy=(x_positions[i+1]-0.6,y), xytext=(x_positions[i]+0.6,y), arrowprops=dict(arrow
ax.set_xlim(0,8)
ax.set_ylim(0,1)
ax.axis('off')
circuit_out = '/mnt/data/qsvt_qsa_circuit_schematic.png'
plt.savefig(circuit_out, bbox_inches='tight', dpi=150)
plt.show()
print('Saved circuit schematic to', circuit_out)
```

Experiment logging and example plots

This cell simulates logging of experiment metrics and creates summary plots for accuracy and fidelity.

```
# Simulated results for plotting
models = ['QSVT-SA','LSTM-CNN','drGAT']
mimic_acc = [0.85, 0.885, 0.902]
sider_acc = [0.823, 0.841, 0.867]
np.random.seed(0)

plt.figure(figsize=(8,4))
x = np.arange(len(models))
width = 0.25
plt.bar(x - width, mimic_acc, width, label='MIMIC')
plt.bar(x, sider_acc, width, label='SIDER')
plt.xticks(x, models)
plt.ylim(0.7,1.0)
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison (simulated)')
plt.legend()
plot_out = 'qsvt_qsa_accuracy.png'
plt.savefig(plot_out, bbox_inches='tight', dpi=150)
plt.show()
print('Saved accuracy plot to', plot_out)
```