# Multi-Modal Knowledge-Primed Neural Networks (MM-KPNNs)

## Purpose & Biological Context

This exercise guides you through developing a Graph Neural Network (MM-KPNN) pipeline that integrates single-cell multi-omic data—combining scRNA-seq and scATAC-seq—into a unified graph-based framework for cell-type classification.

## Pipeline Overview

**Step-by-step methodology:** from data loading and preprocessing to model training, evaluation, explainability, and extension.

1. **Core Workflow (Steps 1–9):** Environment setup, model definition, example data, training & evaluation, embeddings visualization.
2. **Real-Data Integration (Step 10):** Swap in your own multi-omic datasets without changing the core pipeline.
3. **Technical Enhancements (Step 11):** Hyperparameter tuning, graph variations, and model extensions.
4. **Biological Expansion (Step 12):** Integrate regulatory priors, validate feature importance, and adapt to spatial or batch contexts.

- **Application cases:**

1. **scRNA-seq PBMC3k:** classify peripheral blood mononuclear cell types using transcriptomic features.
2. **Spatial Transcriptomics (Visium Lymph Node):** incorporate spatial coordinates and transcription-factor network priors to resolve tissue microenvironments.

By explicitly fusing gene expression and chromatin accessibility into node features and leveraging k-NN and radius graphs for spatial context, MM-KPNN captures molecular, regulatory, and phenotypic relationships—enabling robust classification and biological insight.

---

## Step 1. Imports & Environment Setup

| Library | Version | Purpose |
|---|---|---|
| Python | 3.9.16 | Language runtime |
| PyTorch (+cpu) | 2.7.0+cpu | Core DL framework |
| PyTorch-Geometric | 2.6.1 | GNN layers & utilities |
| Scanpy | 1.9.1 | Single-cell data I/O & preprocessing |
| scikit-learn | 1.0.2 | k-NN, train/test splits, metrics |
| umap-learn | 0.5.3 | Embedding visualization |
| Matplotlib | 3.5.3 | Plotting |

---

# Step 2. Model Definition

Define **MM-KPNN** with:

- **Two GCNConv layers** → ReLU → **Linear head** → cell-type logits
- Rationale: successive graph convolutions propagate features across 2 hops, capturing local and broader neighborhood structure.

```
class MMKPNN(nn.Module):
    def __init__(...):
        super().__init__()
        self.conv1 = GCNConv(in_feats, hidden)
        self.conv2 = GCNConv(hidden,  hidden)
        self.fc    = nn.Linear(hidden, out_feats)

    def forward(self, x, edge_index):
        x = F.relu(self.conv1(x, edge_index))
        x = F.relu(self.conv2(x, edge_index))
        return self.fc(x)
```

---

# Step 3. Dataset Loading & Preprocessing

1. **Load example data:** `scanpy.datasets.pbmc3k_processed()` → HVG matrix + PCA
2. **Graph construction:** k-NN (k=10) on PCA coords via `NearestNeighbors` → `edge_index`
3. **Splitting:** 80/10/10 train/val/test masks (node-level boolean arrays).

---

# Step 4. Model Instantiation & Device Setup

- Detect device: `device = torch.device("cuda" if ... else "cpu")`
- Move model and data: `model.to(device)`, `data.to(device)`

---

## Step 5. Loss & Metrics

- **Loss:** `nn.CrossEntropyLoss()` (optionally with class weights or FocalLoss)
- **Metrics:** accuracy via masked indexing; `classification_report`, macro-F1 via scikit-learn.

---

## Step 6. Training & Validation Loop

- Iterate epochs: `optimizer.zero_grad()`, forward, `loss.backward()`, `optimizer.step()`
- Log train/val loss & accuracy every N epochs.

---

## Step 7. Test Set Evaluation

- Compute test loss & accuracy on `test_mask`
- Print `classification_report(true, preds)` with precision/recall/F1 per cluster.

---

## Step 8. Explainability

- **Gradient Saliency:** compute ∂(target_logit)/∂x for a target node
- **Bar plot:** top-K features (genes/LS dims) by absolute gradient.

---

## Step 9. Embedding Visualization & Cluster Quality

- Extract second-layer embeddings, run UMAP → 2D
- Compute silhouette score on embeddings
- 2D scatter plot colored by true cell type.

---

## Step 10. Swap in Real Data

Load your own scRNA `.h5ad` and scATAC 10x-mtx:

- QC, normalization, HVG selection (RNA); TF-IDF + LSI (ATAC)
- Concatenate features, build k-NN graph, stratified splits
- **Plug-and-play:** downstream steps 4–9 remain unchanged.

---

## Step 11. Technical Recommendations

Hyperparameter Tuning with Optuna. Graph variations, Model extensions and Cross-validation

---

## Step 12. Biological Expansion

- **TF-Prior Integration:** add self-loop edges for top-quartile co-expression of TF→target pairs
- **Spatial Transcriptomics:** use physical coords for graph construction
- **Next expansions:** pathway enrichment, comparative biology -test healthy vs. disease samples, cross-tissue analysis, batch integration -correct batch effects, pre-train on large atlases, fine-tune per datase

---