**📚 Training Pipeline Documentation**

This document outlines the key design improvements, configuration flexibility, and engineering practices followed in the model training and evaluation pipeline.

**🧩 Modularity & Reusability**

All core functions (train, evaluate) are decoupled from global state and now accept the following as **explicit arguments**:

* model
* data loaders (train\_loader, val\_loader)
* optimizer, scheduler
* loss criterion
* device
* config parameters (epochs, metrics, early stopping, etc.)

This enhances:

* Reusability across different models/projects
* Easier unit testing
* Cleaner experimentation

**⚙️ Configuration Management**

Key hyperparameters are now passed as arguments, avoiding hardcoded values:

* Training params: epochs, early\_stopping\_patience, clip\_grad\_norm, scheduler\_step\_per\_batch
* Architecture params: input\_dim, hidden\_dim, output\_dim, dropout
* File paths: model\_save\_path, log\_save\_path
* Evaluation logic: metric\_func, metric\_name

Benefits:

* Easy hyperparameter tuning
* Clear separation of concerns
* No need to edit function logic for minor changes

**💾 Model Checkpointing**

Efficient saving strategy:

* Saves only the **best model** based on the chosen validation metric.
* Overwrites previous best model to conserve storage.
* Save path is fully configurable.

**📈 Evaluation & Metrics**

Enhancements include:

* Metric function and name passed as arguments: supports accuracy, F1, AUC, etc.
* Evaluation uses torch.no\_grad() for memory and performance optimization
* Best model chosen based on user-defined metric, not hardcoded

**🛑 Early Stopping**

* Controlled via early\_stopping\_patience
* Monitors **validation loss** instead of accuracy (more stable for stopping decisions)
* Clean and readable implementation logic

**📉 Learning Rate Scheduling**

* Supports both epoch-wise and batch-wise scheduling using scheduler\_step\_per\_batch flag
  + True: batch-level (e.g., OneCycleLR)
  + False: epoch-level (e.g., StepLR, ReduceLROnPlateau)

**✂️ Gradient Clipping**

* Explicitly handled via clip\_grad\_norm parameter in train
* Easily enabled/disabled or tuned
* Prevents exploding gradients

**📊 Logging & Monitoring**

* Improved TQDM progress display (train\_loss shown per batch)
* Tracks metrics and losses using a structured history dictionary
* Logging to CSV via:

**🧠 Model Architecture & Initialization**

* \_\_init\_\_ method accepts model dimensions and dropout rate
* Layer improvements:
  + Activation applied **after** BatchNorm
  + ReLU applied **after** fusion layers
* Weight initialization:
  + kaiming\_uniform\_, xavier\_uniform\_ for better convergence

**🧱 Code Robustness**

* Uses pathlib for safe path handling and mkdir(parents=True, exist\_ok=True)
* Clear variable naming:
  + best\_val\_metric instead of max\_auc\_score
  + Consistent naming improves readability

**✅ Summary**

| **Feature** | **✅ Status** |
| --- | --- |
| Modular Design | ✔️ |
| Configurable Training | ✔️ |
| Metric Flexibility | ✔️ |
| Early Stopping | ✔️ |
| Batch/Epoch Scheduling | ✔️ |
| Gradient Clipping | ✔️ |
| Efficient Checkpointing | ✔️ |
| Robust Logging | ✔️ |
| Proper Weight Initialization | ✔️ |

This modular and well-structured design enables fast experimentation, easier debugging, and production-ready robustness. Suitable for both research and deployment contexts.