Phase 1 Complete Implementation Plan

Overview

This document outlines all the files needed for your Phase 1 baseline architecture selection system. Each file is designed with clear responsibilities and follows software engineering best practices for research code.

Directory Structure with File Descriptions

```
ids_phase1_research/
│
├── src/
│   ├── __init__.py
│   │
│   ├── data/
│   │   ├── __init__.py
│   │   ├── downloaders.py     # Dataset downloading with progress tracking
│   │   ├── loaders.py         # Data loading (sampled and streaming)
│   │   └── validators.py      # Data validation and quality checks
│   │
│   ├── preprocessing/
│   │   ├── __init__.py
│   │   ├── feature_engineering.py  # Canonical feature schema implementation
│   │   ├── windowing.py       # Flow windowing (Option A)
│   │   ├── scalers.py         # Normalization and encoding
│   │   └── pipeline.py        # End-to-end preprocessing pipeline
│   │
│   ├── models/
│   │   ├── __init__.py
│   │   ├── mlp.py             # Small MLP baseline
│   │   ├── ds_cnn.py           # DS-1D-CNN implementation
│   │   ├── lstm.py             # Small LSTM baseline
│   │   └── base_model.py       # Base class with common functionality
│   │
│   ├── training/
│   │   ├── __init__.py
│   │   ├── trainer.py          # Unified training loop
│   │   ├── evaluator.py        # Evaluation metrics and timing
│   │   └── callbacks.py        # Early stopping, checkpointing, LR scheduling
│   │
│   └── utils/
│       ├── __init__.py
│       ├── config.py           # Configuration loading and validation
│       ├── logging_utils.py    # Logging setup and utilities
│       ├── metrics.py          # Custom metrics (FLOPs counting, etc.)
│       ├── visualization.py    # Plotting and visualization
│       └── system_utils.py     # System monitoring (CPU, memory)
│
├── configs/
│   ├── data_config.yaml        # [CREATED] Dataset configuration
│   ├── preprocess_config.yaml  # [CREATED] Preprocessing configuration
│   └── phase1_config.yaml      # [CREATED] Main experiment configuration
│
├── data/
│   ├── raw/                    # Downloaded datasets (gitignored)
│   ├── processed/              # Preprocessed features (gitignored)
│   └── samples/                # Sampled datasets for prototyping
│
├── experiments/                # Timestamped experiment runs
│   └── phase1_YYYYMMDD_HHMMSS/
│       ├── config.yaml         # Exact config used for this run
│       ├── models/             # Saved model checkpoints
│       ├── logs/               # Training logs
│       ├── plots/            # Generated visualizations
│       └── reports/          # Evaluation reports and decision
│
├── outputs/
│   ├── plots/                # Shared plots across experiments
│   ├── reports/              # Shared reports
│   └── models/               # Final selected models
│
├── logs/                     # Application logs
│
├── main_phase1.py            # [TO CREATE] Main entry point
├── setup_phase1.py           # [CREATED] Project setup script
├── requirements.txt          # [CREATED] Dependencies
├── README.md                 # [CREATED] Project documentation
└── .gitignore                # [CREATED] Git ignore rules
```

Implementation Priority & Dependencies

Phase 1A: Foundation (Day 1)

Status: Ready to implement

1. **setup_phase1.py** ✓ [CREATED]
    - Creates directory structure
    - Generates configuration files
    - Initializes project
2. **src/utils/config.py** [NEXT]
    - Load and validate YAML configurations
    - Merge configs with command-line arguments
    - Handle different experiment modes

3. **src/utils/logging_utils.py** [NEXT]
    - Setup structured logging
    - Progress bars for long operations
    - Experiment tracking

Phase 1B: Data Pipeline (Day 1-2)

Dependencies: Phase 1A complete

4. **src/data/downloaders.py**
    - Download CIC-IDS2017 from Kaggle
    - Download TON-IoT from official source
    - Progress tracking and checksum verification
    - Resume interrupted downloads
5. **src/data/validators.py**
    - Check data integrity
    - Validate schema and columns
    - Report missing values and quality issues
6. **src/preprocessing/feature_engineering.py**
    - Implement canonical feature schema
    - Feature mapping from both datasets
    - Strict and flexible modes
    - Generate preprocessing report
7. **src/preprocessing/windowing.py**
    - Sliding window over consecutive flows (Option A)

- Flow grouping by source IP
- Window labeling (any malicious)
- Handle padding for incomplete windows

8. **src/data/loaders.py**
    - PyTorch Dataset for sampled data
    - Memory-mapped streaming loader for full data
    - Batch generation with prefetching

Phase 1C: Models (Day 2-3)

Dependencies: Can start in parallel with 1B

9. **src/models/base_model.py**
    - Abstract base class for all models
    - Common methods: forward, get_params, get_flops
    - Model summary and architecture printing
10. **src/models/mlp.py**
    - 2-3 dense layer MLP
    - ~50K parameters
    - Input flattening for windowed data
11. **src/models/ds_cnn.py**
    - Depthwise separable convolution implementation
    - 2-3 conv blocks
    - ~80K parameters

- Proper PyTorch implementation

12. **src/models/lstm.py**
    - 1-2 LSTM layers
    - ~90-120K parameters
    - Proper handling of sequences

Phase 1D: Training & Evaluation (Day 3-4)

Dependencies: Phases 1B and 1C complete

13. **src/training/callbacks.py**
    - Early stopping
    - Model checkpointing
    - Learning rate scheduling
    - Progress logging
14. **src/training/trainer.py**
    - Unified training loop for all models
    - Gradient clipping
    - Mixed precision support
    - Metric tracking
15. **src/training/evaluator.py**
    - Compute all metrics (accuracy, F1, recall)
    - CPU inference timing with warmup
    - FLOPs counting using thop

   - Parameter counting
   - Cross-dataset evaluation

16. **src/utils/metrics.py**
    - Custom metric implementations
    - FLOPs calculation wrapper
    - Inference time profiler
    - Memory usage tracker

Phase 1E: Visualization & Reporting (Day 4-5)

Dependencies: Phase 1D complete

17. **src/utils/visualization.py**
    - Training curves
    - Comparison plots
    - Confusion matrices
    - ROC and PR curves
    - Architecture diagrams
18. **Decision report generator** (in main_phase1.py)
    - Apply decision criteria automatically
    - Generate final recommendation
    - Highlight red flags
    - Create summary tables
19. **main_phase1.py** [CRITICAL]

  - Command-line interface
  - Orchestrate entire pipeline
  - Handle different modes (quick/medium/full)
  - Generate final report

Implementation Strategy

Week 1 Timeline

Monday-Tuesday: Foundation + Data Pipeline

- Set up project structure (setup_phase1.py)
- Implement configuration and logging utilities
- Create data downloaders with progress tracking
- Build feature engineering pipeline
- Implement windowing logic
- Test with small samples

Wednesday-Thursday: Models + Training

- Implement all three model architectures
- Create unified trainer
- Add evaluation metrics
- Test training loop on small samples
- Verify FLOPs and parameter counting

Friday: Integration + Testing

- Create main_phase1.py entry point
- Run quick-mode experiments end-to-end
- Debug any issues
- Generate visualization and reports

Weekend: Full Experiments

- Run medium-mode experiments
- Start full-dataset runs overnight
- Collect results
- Generate final decision report

Week 2 Focus

Monday-Tuesday: Cross-Dataset Validation

- Train on CIC → evaluate on TON
- Train on TON → evaluate on CIC (if time)
- Analyze performance drops
- Update decision report

Wednesday-Thursday: Analysis & Refinement

- Analyze results against decision criteria
- Address any red flags
- Fine-tune hyperparameters if needed
- Re-run experiments with adjustments

Friday: Documentation & Deliverables

- Final decision report
- Documentation of methodology
- Code cleanup and comments
- Prepare for Phase 2

Testing Strategy

Each module should be testable independently:

1. **Unit tests** for individual functions
    - Feature extraction
    - Windowing logic
    - Model forward pass
2. **Integration tests** for pipelines
    - Full preprocessing pipeline
    - Training one epoch
    - Evaluation on small dataset
3. **End-to-end tests**
    - Quick mode with 1000 samples
    - Verify all outputs generated
    - Check decision criteria application

Risk Mitigation

Potential Issues & Solutions

1. **Dataset download failures**
   - Solution: Multiple download sources, resume capability
   - Fallback: Manual download instructions
2. **Memory issues with full datasets**
   - Solution: Streaming loaders with memory mapping
   - Fallback: Work with sampled data only
3. **Feature mismatch between datasets**
   - Solution: Strict mode catches this early
   - Fallback: Flexible mode with imputation
4. **Slow CPU training**
   - Solution: Optimized batch sizes, efficient implementations
   - Fallback: Focus on sampled datasets
5. **Model convergence issues**
   - Solution: Gradient clipping, LR scheduling
   - Fallback: Simpler architectures, fewer layers

Next Steps

1. **Immediate**: Create utility files (config, logging)
2. **Then**: Data download and preprocessing
3. **Then**: Model implementations
4. **Then**: Training pipeline
5. **Finally**: Main script and integration

This modular approach allows you to test each component independently and catch issues early. Each file is self-contained with clear interfaces, making debugging much easier.