ML Engine Fixes Implemented

This document summarizes the fixes and optimizations implemented for the cryptocurrency mining monitoring ML recommendation engine based on the issues identified in the code review.

1. Code Structure Improvements

1.1 Circular Import Dependencies

- Created a new `utils` package to hold shared functionality
- Moved constants to `utils/constants.py` to avoid circular imports
- Restructured imports to use absolute imports consistently
- Separated configuration into logical components

1.2 Module Organization

- Created proper package structure with `__init__.py` files
- Standardized import patterns across modules
- Extracted common functionality to utility modules

2. Algorithm Implementation Enhancements

2.1 Feature Selection

- Implemented proper feature selection in `profit_model.py` using `SelectFromModel`
- Added feature importance analysis and reporting
- Added ability to prune low-importance features

2.2 Validation Strategy

- Implemented time-series cross-validation to handle temporal data properly
- Added comprehensive model evaluation metrics
- Improved directional accuracy calculation with proper handling of edge cases

2.3 Hyperparameter Management

- Made hyperparameters configurable through the config module
- Added support for hardware-specific parameter spaces in power optimization

3. Data Processing Optimizations

3.1 Vectorized Operations

- Replaced inefficient loops with vectorized pandas/numpy operations
- Optimized window calculations for time-series data
- Improved memory efficiency by reducing intermediate DataFrame creation

3.2 Input Validation

- Added comprehensive input validation with custom `ValidationError` class
- Implemented proper error handling and reporting
- Added domain-specific validation for mining telemetry data

3.3 Missing Data Handling

- Implemented robust missing data handling with domain-specific strategies
- Added outlier detection and handling
- Improved data type conversion and validation

4. Feature Engineering Improvements

4.1 Advanced Features

- Added more sophisticated trend analysis with polynomial fitting
- Implemented momentum and volatility features
- Added cross-source derived features that combine data from multiple sources

4.2 Computation Efficiency

- Optimized window calculations using vectorized operations

- Implemented caching for repeated calculations
- Improved memory usage with in-place operations where appropriate

5. Model Training and Evaluation Enhancements

5.1 Cross-Validation

- Implemented proper time-series cross-validation
- Added comprehensive evaluation metrics including confidence intervals
- Improved early stopping implementation

5.2 Model Persistence

- Enhanced model versioning with proper metadata
- Added separate metadata storage for easy access
- Improved model loading with better error handling

6. API Integration Improvements

6.1 Error Handling

- Implemented comprehensive error handling with specific error types
- Added proper logging throughout the API
- Improved error reporting to clients

6.2 Data Transfer

- Optimized data transfer between components
- Added asynchronous processing for long-running operations
- Implemented proper validation of API requests

6.3 API Documentation

- Enhanced API documentation with OpenAPI specifications
- Added example requests and responses
- Improved endpoint descriptions

7. Error Handling and Logging

7.1 Logging System

- Implemented a comprehensive logging system
- Added structured logging with proper log levels
- Configured file and console logging

7.2 Exception Handling

- Created custom exception types for different error scenarios
- Implemented consistent error handling patterns
- Added context information to error messages

8. Performance Optimizations

8.1 Batch Processing

- Implemented batch prediction for multiple feature sets
- Added parallel processing capabilities
- Optimized memory usage in data processing

8.2 Model Optimization

- Added model pruning capabilities
- Implemented feature importance-based optimization
- Added performance benchmarking

9. Dependency Management

9.1 Pinned Dependencies

- Created requirements.txt with pinned versions
- Added separate development dependencies
- Created setup script for easy environment setup

Next Steps

1. Implement unit and integration tests for the ML engine
2. Set up continuous integration and deployment
3. Add monitoring and alerting for the ML engine
4. Implement A/B testing for model improvements
5. Add support for additional mining hardware types

Usage Instructions

1. Clone the repository
2. Run the setup script: `./setup.sh` (add `--dev` for development dependencies)
3. Activate the virtual environment: `source venv/bin/activate`
4. Start the API server: `python -m ml_engine.api`
5. Access the API documentation at `http://localhost:8000/docs`