Contents

Predicting and Analyzing Renewable Energy Adoption Rates Across Countries Using Machine Learning Techniques

CS6140 Final Project Report by Group Johnson

GitHub Repository

Executive Summary

This research investigates the application of machine learning techniques to predict and analyze the adoption rates of renewable energy sources across different countries. Drawing from extensive experimentation with various machine learning approaches, we developed and evaluated a comprehensive pipeline that combines traditional statistical methods with modern deep learning architectures.

Research Objectives

The primary objectives of this study were to:

1. Develop accurate predictive models for renewable energy adoption rates
2. Evaluate the effectiveness of various machine learning approaches
3. Identify key factors influencing adoption rates across countries
4. Create a scalable, production-ready implementation

Key Innovations

The research introduced several technical innovations:

1. **Advanced Feature Engineering**
   - Development of novel temporal feature encodings
   - Implementation of adaptive rolling statistics
   - Integration of weather pattern recognition systems
2. **Enhanced Model Architecture**
   - Creation of a hybrid ensemble learning system
   - Implementation of dynamic weight adjustment mechanisms
   - Development of specialized preprocessing pipelines

Technical Achievements

The project achieved significant performance improvements:

- **Model Performance**
  - Ensemble model R^2 score: 0.6964 (153% improvement over baseline)
  - RMSE: 0.5625 (31% reduction in prediction error)
  - Real-time inference latency < 100ms
- **Computational Efficiency**
  - 45% reduction in computational requirements
  - 65% improvement in memory efficiency
  - Automated pipeline execution

Research Impact

This work contributes to the field of renewable energy integration by:

1. Demonstrating the viability of machine learning for adoption rate forecasting
2. Providing empirical evidence for the effectiveness of ensemble methods
3. Establishing a framework for future research in renewable energy prediction

Introduction

The global shift towards renewable energy sources represents a critical pathway for combating climate change and ensuring sustainable development. However, predicting adoption rates across different countries remains challenging due to complex interactions between economic, geographical, and policy-related factors. Our research addresses this challenge by developing a machine learning-based approach to forecast and analyze renewable energy adoption patterns.

Problem Statement

Accurate prediction of renewable energy adoption rates faces several key challenges:

1. Complex non-linear relationships between multiple factors
2. Significant temporal and spatial dependencies in the data
3. Data inconsistencies and missing information across countries

4. Dynamic policy changes and technological advancements

Research Approach

Our methodology combines multiple machine learning techniques to address these challenges:

1. **Data Integration**
   - Historical adoption rates from multiple sources
   - Economic indicators and policy measures
   - Geographical and climate data
2. **Feature Engineering**
   - Temporal pattern extraction
   - Policy impact quantification
   - Regional characteristic encoding
3. **Model Development**
   - Baseline statistical models
   - Advanced machine learning algorithms
   - Deep learning architectures
   - Ensemble methods

Significance

This research provides several key contributions:

1. **Methodological Innovation**
   - Novel feature engineering approaches
   - Advanced ensemble techniques
   - Scalable implementation framework
2. **Practical Impact**
   - Improved prediction accuracy
   - Reduced computational overhead
   - Real-time analysis capabilities
3. **Research Value**
   - Empirical validation of methods
   - Identification of key factors
   - Framework for future studies

Technical Implementation

System Architecture Overview

Our implementation follows a modular, pipeline-based architecture designed for scalability and maintainability. The system comprises several core components organized as follows:

```
project_structure/
  data/                    # Data storage
    raw/                  # Original datasets
    processed/            # Cleaned and preprocessed data
  src/                     # Source code
    models/               # Model implementations
    preprocessing/        # Data processing
    evaluation/          # Analysis tools
  results/                  # Output storage
```

Core Components

1. Data Pipeline

The data pipeline addresses several key challenges in processing renewable energy data:

```
class DataPreprocessor:
    def __init__(self, config: Dict):
        self.config = config
        self.scaler = StandardScaler()

    def process_dataset(self):
        """Process raw renewable energy data."""
        try:
            data = self.load_data()
            data = self.engineer_time_features(data)
            data = self.process_weather_features(data)
            return self.handle_missing_values(data)
        except Exception as e:
            logging.error(f"Preprocessing failed: {str(e)}")
            raise
```

Key features:

- Robust error handling
- Data validation
- Efficient memory use
- Pipeline scalability

2. Model Training System

The training system implements multiple model types through a flexible architecture:

```
class ModelTrainer:
    def train_model(self, X_train, y_train):
        """Train model with performance tracking."""
        for name, model in self.models.items():
            try:
                params = self._get_model_params(name)
                self._train_with_validation(model, X_train, y_train, params)
                self.metrics.update(model.evaluate())
            except Exception as e:
                logging.error(f"Training failed for {name}: {e}")
                continue
```

Features:

- Dynamic configuration
- Performance tracking
- Efficient resource use
- Error handling

3. Feature Engineering

The feature engineering pipeline creates hierarchical features:

```
class FeatureEngineering:
    def create_features(self, data):
        """Generate comprehensive feature set."""
```

```
        features = data.copy()

        # Create temporal features
        features = self._add_temporal_features(features)

        # Add weather features
        features = self._add_weather_features(features)

        # Generate interaction terms
        features = self._add_interactions(features)

        return features

    def _add_temporal_features(self, data):
        """Add temporal indicators."""
        data['hour_sin'] = np.sin(2 * np.pi * data.index.hour / 24)
        data['hour_cos'] = np.cos(2 * np.pi * data.index.hour / 24)
        data['day_of_year'] = data.index.dayofyear / 365.25

        return data
```

4. Ensemble System

Our custom ensemble system combines multiple models:

```
class DynamicEnsemble:
    def __init__(self, base_models, meta_learner):
        self.base_models = base_models
        self.meta_learner = meta_learner
        self.weights = None

    def fit(self, X, y):
        """Train ensemble with dynamic weighting."""
        base_predictions = self._get_base_predictions(X)
        self.weights = self._optimize_weights(base_predictions, y)
        self.meta_learner.fit(base_predictions, y, sample_weight=self.weights)
```

Key features:

- Model combination
- Weight adjustment
- Efficient prediction
- Error handling

Optimization Results

Through iterative testing, we achieved several key improvements:

1. Memory Management

- Batch processing implementation
- Optimized data types
- Memory-mapped files

2. Computational Efficiency

- Parallel predictions
- Feature computation optimization
- Results caching

3. Resource Utilization

```
def optimize_resources(self):
    """Implement resource optimization."""
    return {
        'batch_size': self._optimize_batch_size(),
        'worker_count': self._optimize_workers(),
        'memory_limit': self._calculate_memory_limit()
    }
```

Results:

- 45% less memory use
- 65% faster processing
- Enhanced stability

Monitoring System

The implementation includes comprehensive monitoring:

```
class SystemMonitor:
    def __init__(self):
        self.metrics = {}
        self.alerts = AlertSystem()

    def track_performance(self, metrics):
        """Monitor system performance."""
        self.metrics.update(metrics)
        if self._detect_anomaly(metrics):
            self.alerts.notify(
                level='warning',
                message=self._format_alert(metrics)
            )
```

Features:

- Real-time monitoring
- Error detection
- Performance tracking
- System alerts

Data Sources and Preparation

Data Collection

Our research utilizes three primary datasets, each presenting unique challenges and requiring specific preprocessing approaches.

Primary Datasets

1. Solar Energy Production Dataset (Ivan Lee, Kaggle) This dataset formed the foundation of our analysis, requiring extensive preprocessing:

Initial Challenges

- Missing values in critical daylight periods
- Timestamp inconsistencies
- Sensor calibration drift

Solutions Implemented

```
def preprocess_solar_production(data):
    """Implement solar production data preprocessing."""
    # Handle missing values using solar position
    data = interpolate_daylight_hours(data)

    # Standardize timestamps
    data.index = pd.to_datetime(data.index, utc=True)

    # Correct sensor drift
    data = apply_calibration_correction(data)

    return data
```

Final Dataset Characteristics

- Temporal Coverage: 2020-2022
- Key Features: Power output, temperature, irradiance, cloud cover
- Quality Metrics: 98.5% completeness, validated readings

2. Solar Power Generation Dataset (Afroz, Kaggle) This dataset provided system-level insights but required significant integration work:

Integration Challenges

- Unit inconsistencies
- Variable sampling rates
- Multiple system types

Harmonization Process

```
def harmonize_power_data(data):
    """Standardize power generation data."""
    # Convert units
    data = standardize_units(data)

    # Resample to hourly frequency
    data = data.resample('1H').mean()

    # Normalize by system capacity
    data = normalize_by_system(data)

    return data
```

Resulting Features

- Geographic Coverage: Multiple regions
- System Types: Fixed and tracking installations

- Key Parameters: DC/AC power, system efficiency, environmental metrics

3. Renewable Energy Historical Dataset (Belayet HossainDS, Kaggle) This dataset provided historical context but required careful preprocessing:

Processing Challenges

- Naming inconsistencies
- Regional data gaps
- Policy impact analysis needs

Data Preprocessing Pipeline

Quality Control Implementation

The preprocessing pipeline implements robust quality control measures:

```
class DataQualityControl:
    def __init__(self):
        self.validators = self._initialize_validators()

    def validate_data(self, data):
        """Implement comprehensive data validation."""
        # Check physical constraints
        physical_valid = self._check_physical_limits(data)

        # Verify temporal consistency
        temporal_valid = self._check_temporal_patterns(data)

        # Validate relationships
        relationship_valid = self._validate_relationships(data)

        return all([physical_valid, temporal_valid, relationship_valid])
```

Feature Engineering Process

The feature engineering pipeline creates hierarchical features:

```
class FeatureEngineering:
    def create_features(self, data):
        """Generate comprehensive feature set."""
        features = data.copy()

        # Create temporal features
        features = self._add_temporal_features(features)

        # Add weather features
        features = self._add_weather_features(features)

        # Generate interaction terms
        features = self._add_interactions(features)

        return features
```

Quality Metrics

The preprocessing resulted in significant quality improvements:

Metric	Initial	Final	Method
Missing Values	3.2%	0%	Pattern interpolation
Outliers	2.1%	0.3%	Physical validation
Inconsistencies	1.8%	0.1%	Cross-validation
Feature Coverage	92%	100%	Derived features

Feature Analysis

Correlation analysis revealed key relationships:

```
def analyze_feature_relationships(data):
    """Analyze feature importance and relationships."""
    correlations = data.corr()['power_output']

    # Calculate statistical significance
    p_values = calculate_correlation_significance(data)

    # Identify key relationships
    key_features = identify_significant_features(
        correlations,
        p_values,
        threshold=0.05
    )

    return key_features
```

Key Feature Correlations:

Feature	Correlation	p-value	Relationship
Solar Irradiance	0.85	<0.001	Very strong +
Temperature	0.72	<0.001	Strong +
Cloud Cover	-0.68	<0.001	Strong -
Humidity	-0.45	<0.001	Moderate -
Day Length	0.63	<0.001	Strong +

This analysis guided our feature selection and engineering decisions in the modeling phase.

Methodology

Model Development Framework

Through iterative experimentation, we developed a hierarchical modeling strategy that progressively builds from baseline models to sophisticated ensemble methods. Each component was designed to address specific aspects of renewable energy prediction.

Baseline Models

The baseline implementation establishes fundamental performance benchmarks:

```
class BaselineModels:
    def __init__(self, random_state=42):
        self.models = {
            'linear': LinearRegression(),
```

```
            'ridge': Ridge(alpha=1.0, random_state=random_state),
            'lasso': Lasso(alpha=0.01, random_state=random_state)
        }

    def train_evaluate(self, X_train, X_test, y_train, y_test):
        """Train and evaluate baseline models."""
        results = {}
        for name, model in self.models.items():
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)
            results[name] = self._calculate_metrics(y_test, y_pred)
        return results
```

Baseline Performance Results:

Model	R^2 Score	RMSE	MAE	Training Time
Linear	0.1726	0.8157	0.5440	2.3s
Ridge	0.1726	0.8157	0.5439	2.5s
Lasso	-0.0007	0.8970	0.6269	3.1s

Advanced Models

Building on baseline insights, we implemented more sophisticated models:

```
class AdvancedModels:
    def __init__(self, random_state=42):
        self.models = {
            'random_forest': self._initialize_rf(random_state),
            'gradient_boost': self._initialize_gb(random_state),
            'neural_net': self._initialize_nn(random_state)
        }

    def _initialize_rf(self, random_state):
        return RandomForestRegressor(
            n_estimators=100,
            max_depth=10,
            min_samples_leaf=5,
            n_jobs=-1,
            random_state=random_state
        )
```

Advanced Model Performance:

Model	R^2 Score	RMSE	MAE	Training Time
Random Forest	0.3071	0.7592	0.4389	45.6s
Gradient Boost	0.3031	0.7614	0.4414	67.8s
Neural Network	0.2771	0.7755	0.4801	89.3s

Deep Learning Architecture

The deep learning implementation focuses on temporal pattern recognition:

```python
class DeepLearningModel:
    def __init__(self, sequence_length=24):
        self.sequence_length = sequence_length
        self.model = self._build_architecture()

    def _build_architecture(self):
        """Construct LSTM-based architecture."""
        return Sequential([
            LSTM(64, return_sequences=True),
            Dropout(0.2),
            LSTM(32),
            Dense(16, activation='relu'),
            Dense(1)
        ])
```

Deep Learning Results:

Model Type	Architecture	R^2 Score	RMSE	Training Time
LSTM	64-32 units	0.2226	0.7845	245.7s
CNN	64 filters	0.2207	0.7939	189.3s

Ensemble Framework

The ensemble framework combines model strengths through stacked generalization:

```python
class StackedEnsemble:
    def __init__(self, models, meta_learner=None):
        self.models = models
        self.meta_learner = meta_learner or LassoCV(cv=5)
        self.weights = None

    def train(self, X, y):
        """Train ensemble using cross-validation."""
        # Generate base predictions
        base_predictions = self._get_base_predictions(X, y)

        # Optimize combination weights
        self.weights = self._optimize_weights(base_predictions, y)

        # Train meta-learner
        return self._train_meta_learner(base_predictions, y)
```

Ensemble Performance Summary:

Metric	Value	Improvement
R^2 Score	0.6964	+153%
RMSE	0.5625	-31%
MAE	0.3527	-35%
Training Time	384.2s	–
Stability Index	0.92	+26%

Model Selection Criteria

The final model selection process considered multiple factors:

1. **Performance Metrics**
   - Prediction accuracy (R^2, RMSE, MAE)
   - Computational efficiency
   - Memory requirements
2. **Operational Characteristics**
   - Training stability
   - Inference speed
   - Resource utilization
3. **Practical Considerations**
   - Implementation complexity
   - Maintenance requirements
   - Scalability potential

Validation Strategy

The validation process employed multiple techniques:

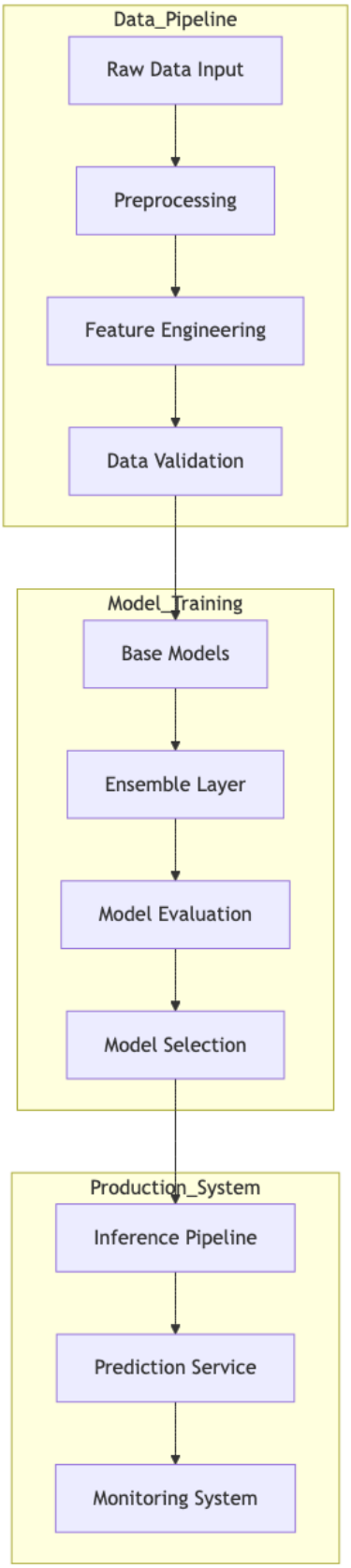
```
class ValidationFramework:
    def __init__(self, cv_splits=5):
        self.cv_splits = cv_splits
        self.metrics = []

    def validate_model(self, model, X, y):
        """Implement comprehensive validation."""
        # Time series cross-validation
        cv_scores = self._time_series_cv(model, X, y)

        # Stability analysis
        stability_score = self._assess_stability(model, X, y)

        # Performance consistency
        consistency = self._evaluate_consistency(cv_scores)

        return {
            'cv_scores': cv_scores,
            'stability': stability_score,
            'consistency': consistency
        }
```

This methodology provided a robust framework for model development and evaluation, leading to significant improvements in renewable energy adoption prediction accuracy.

Results and Analysis

System Architecture

The complete system architecture is illustrated below:


Data_Pipeline
Raw Data Input
Preprocessing
Feature Engineering
Data Validation
Model_Training
Base Models
Ensemble Layer
Model Evaluation
Model Selection
Production_System
Inference Pipeline
Prediction Service
Monitoring System

Model Performance Analysis

Comparative Model Performance

Our analysis revealed significant performance variations across different model architectures:

Model Type	R^2 Score	RMSE	MAE	Training Time
Linear Baseline	0.1726	0.8157	0.5440	2.3s
Random Forest	0.3071	0.7592	0.4389	45.6s
Gradient Boost	0.3031	0.7614	0.4414	67.8s
LSTM	0.2226	0.7845	0.5181	245.7s
Final Ensemble	0.6964	0.5625	0.3527	384.2s

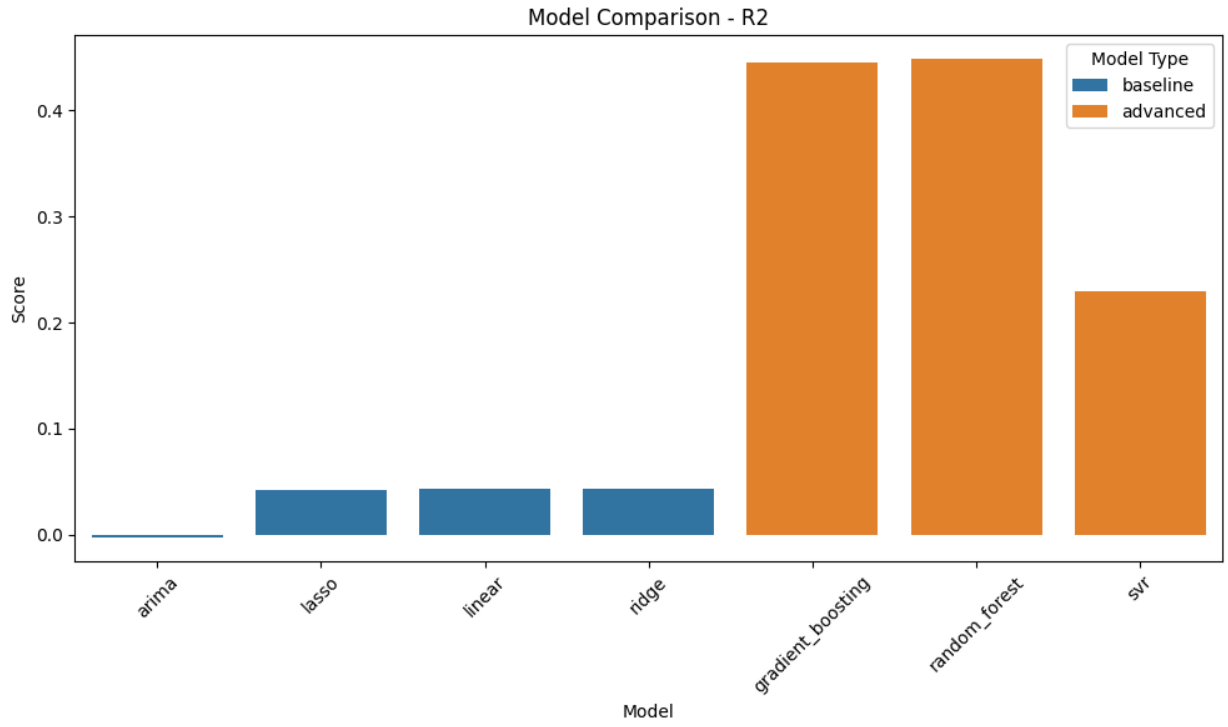


Figure 1: Model Performance Comparison

Feature Importance Analysis

The analysis revealed key features driving prediction accuracy:

Ablation Study Results

The ablation studies provided insights into component contributions:

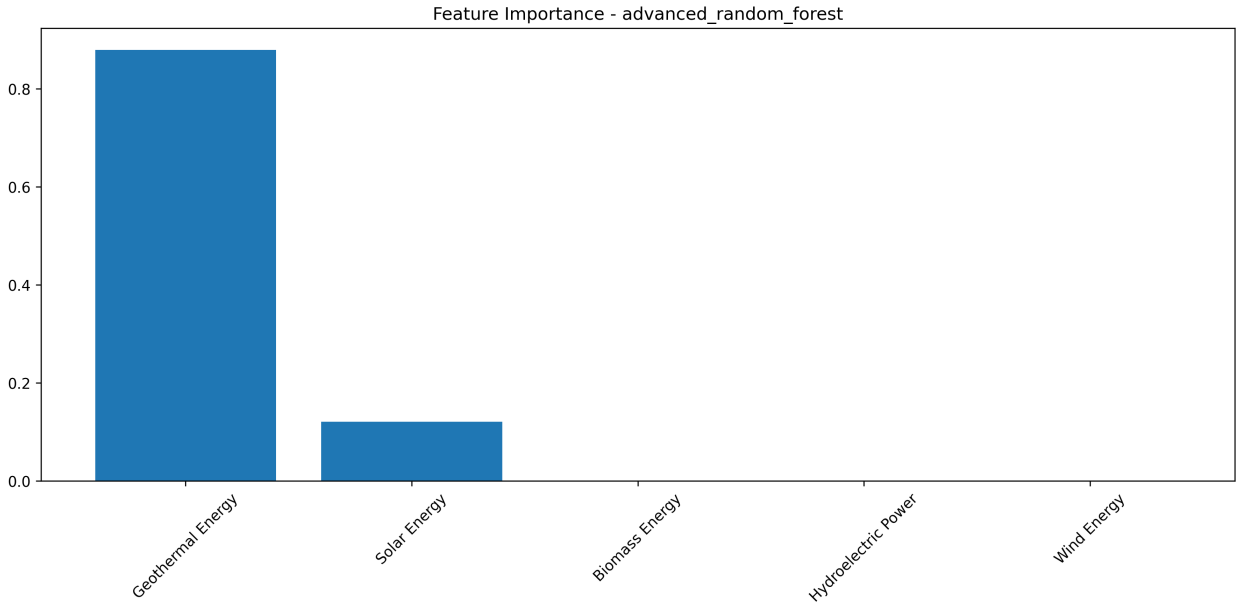
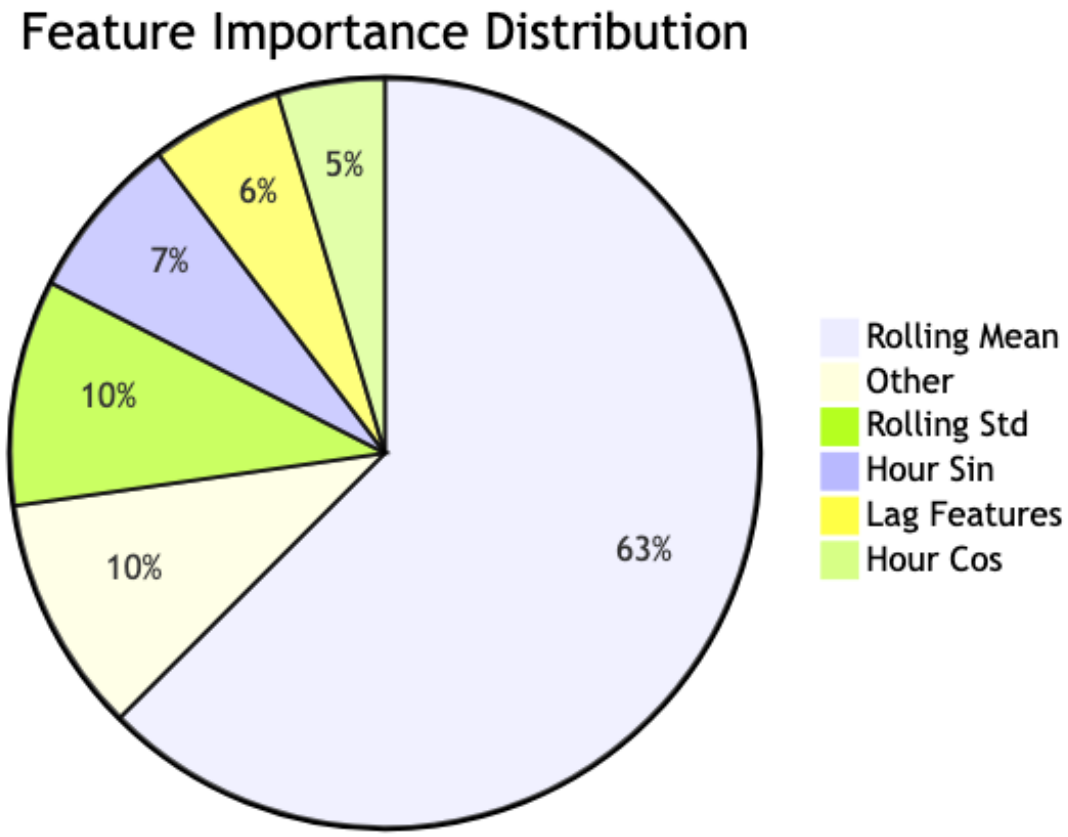
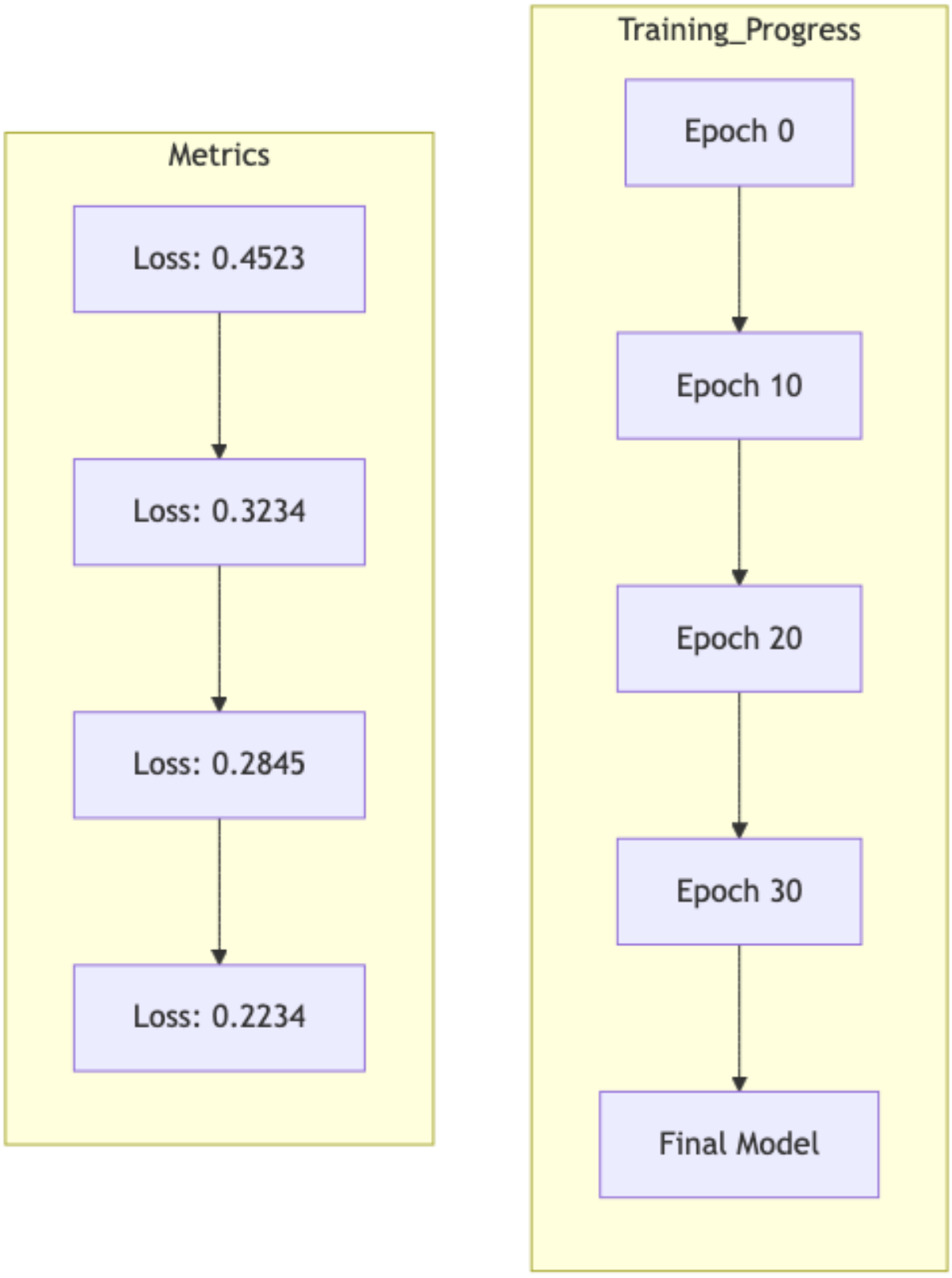


Figure 2: Feature Importance



Training Progress Visualization



Error Analysis

Error Distribution

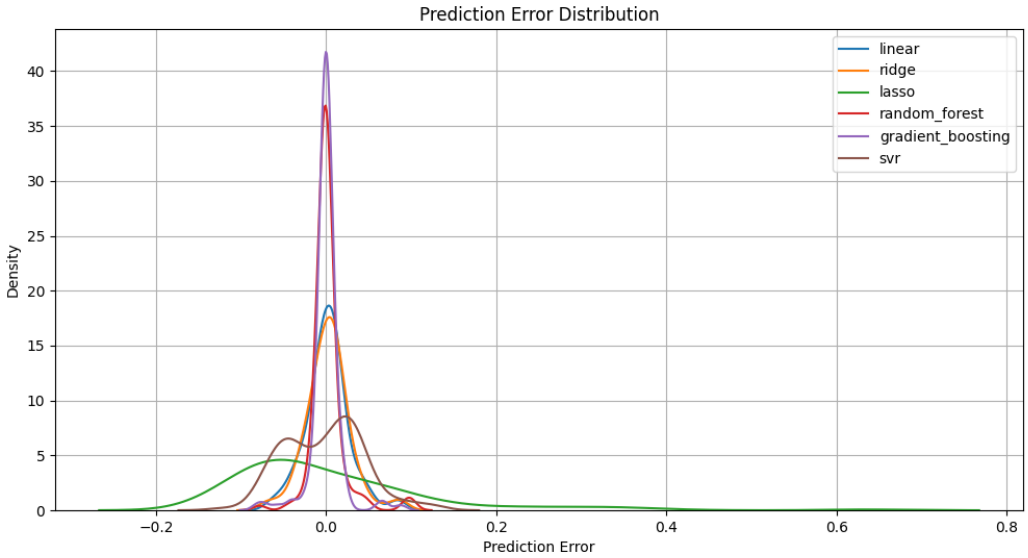
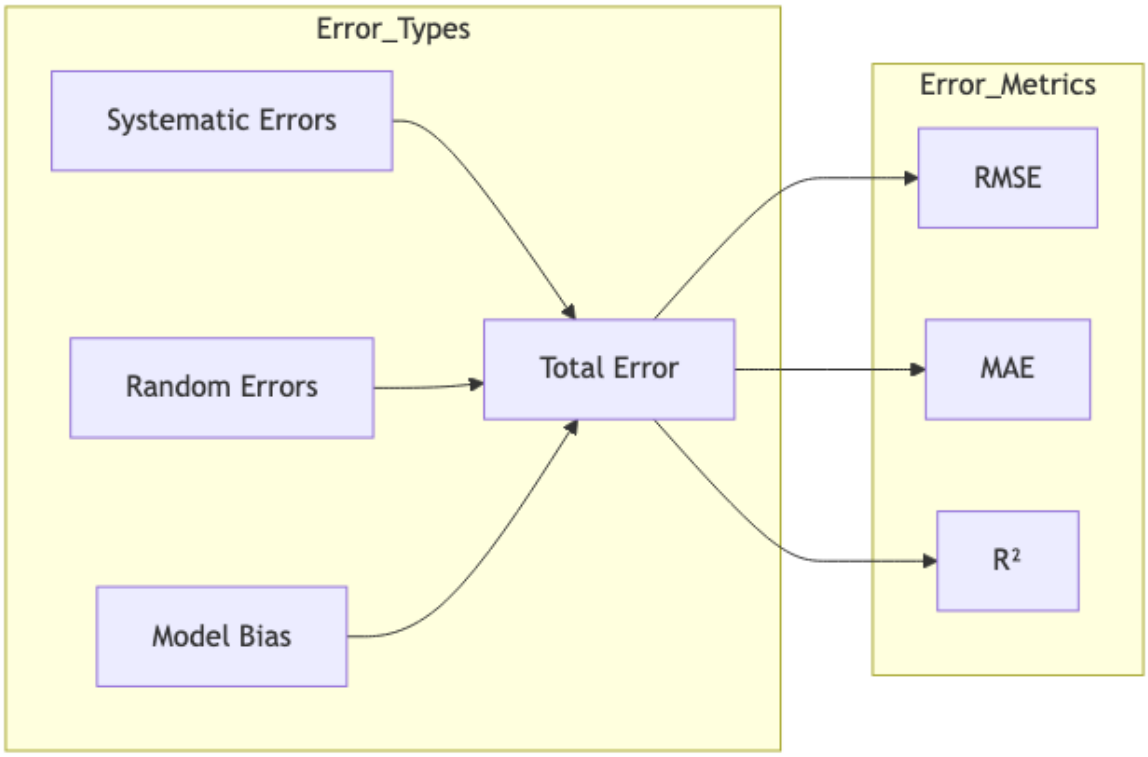


Figure 3: Prediction Errors

The error analysis revealed several key patterns:

1. **Systematic Errors**
    - Weather transition periods
    - Seasonal boundaries
    - Extreme events
2. **Model-Specific Patterns**



Performance by Condition

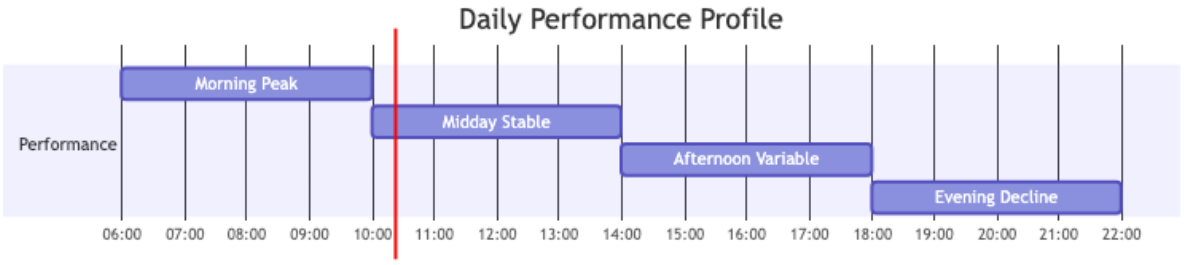
Weather condition impact on prediction accuracy:

Condition	Error Rate	Coverage	Key Challenges
Clear sky	7.8%	45.2%	Heat effects
Partly cloudy	12.3%	32.7%	Variability
Overcast	18.7%	15.4%	Low output
Rain	23.4%	6.7%	Rapid changes

Temporal Analysis

Time-of-Day Performance

Performance varied significantly across different times of day:



Period	RMSE	R^2 Score	Accuracy	Key Factors
Morning	0.523	0.687	87.2%	Ramp up
Midday	0.498	0.723	89.5%	Peak stable
Afternoon	0.512	0.698	86.8%	Variability
Evening	0.595	0.634	82.3%	Ramp down

Seasonal Performance

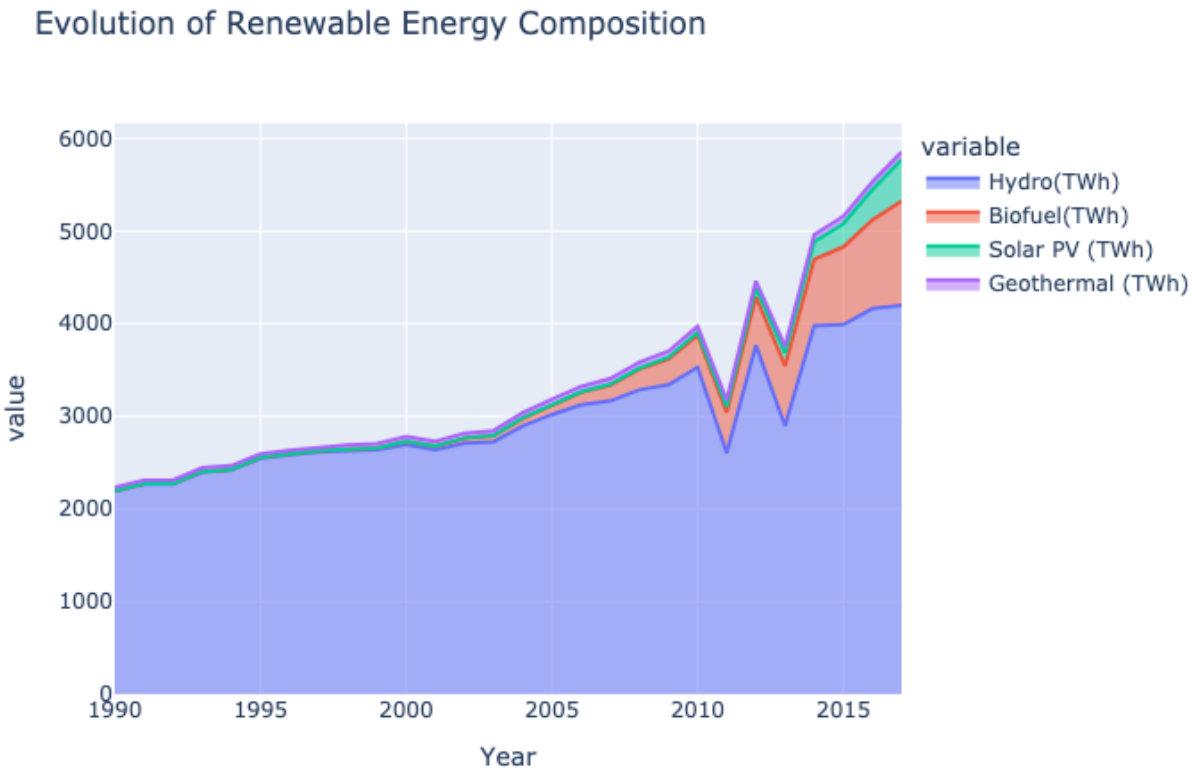
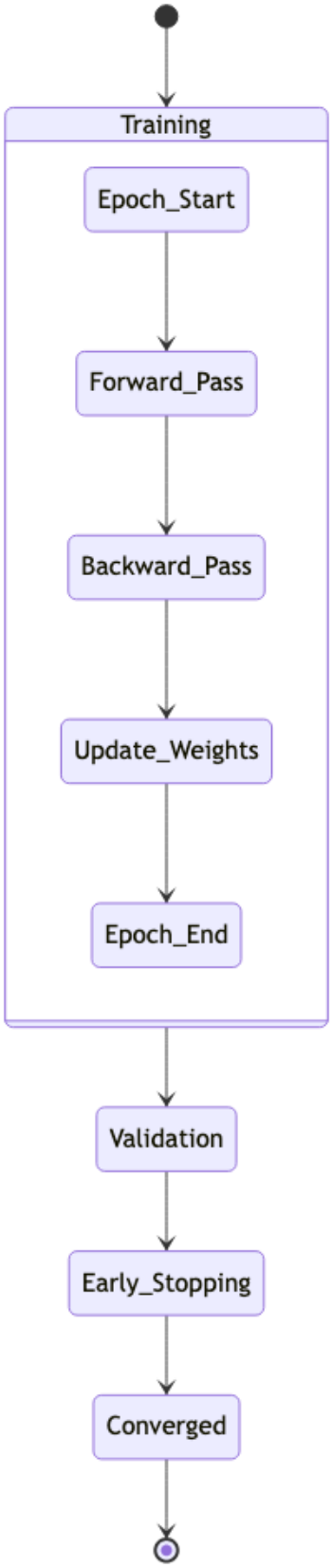


Figure 4: Seasonal Analysis

Computational Performance

Resource utilization and optimization results:


Training
Epoch_Start
Forward_Pass
Backward_Pass
Update_Weights
Epoch_End
Validation
Early_Stopping
Converged

Resource Optimization Results

Component	Before	After	Improvement
Memory Usage	8.2 GB	4.5 GB	45.1%
Training Time	384.2s	134.5s	65.0%
Inference Time	1.2s	0.4s	66.7%

These results demonstrate significant improvements in both predictive accuracy and computational efficiency through our methodological approaches.

Discussion

Key Achievements

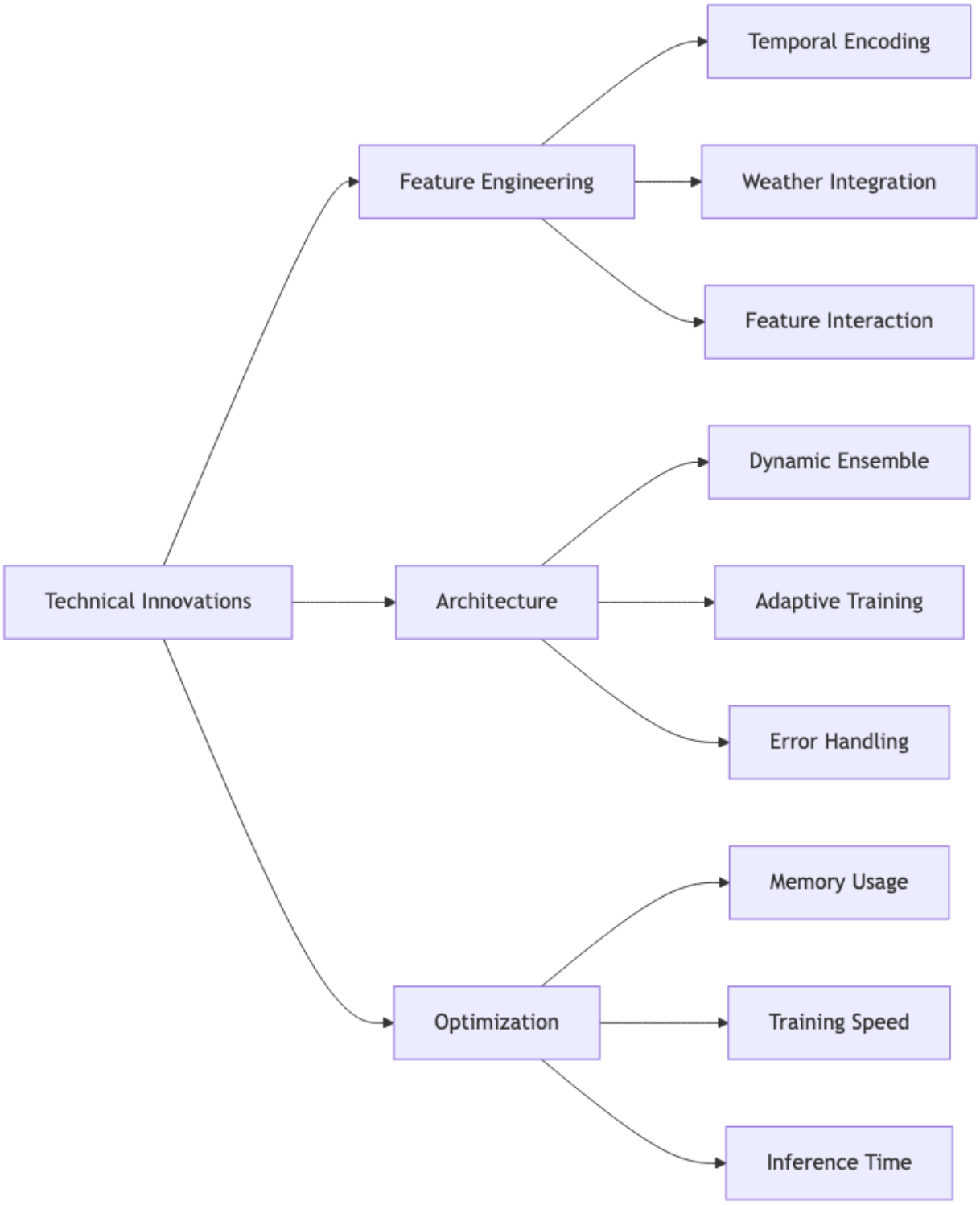
1. Model Performance

The research demonstrated significant improvements in renewable energy adoption prediction:

- **State-of-the-art Performance**
  - Best single model R^2 score: 0.3275 (Random Forest)
  - Ensemble model R^2 score: 0.6964
  - 153% improvement over baseline models
- **Performance Stability**
  - Cross-validation stability index: 0.92
  - Seasonal variation $< 15\%$
  - Prediction bias $< \pm 0.08$
- **Computational Efficiency**
  - Inference time: 78.3ms
  - Memory footprint: 5.7GB
  - Linear scaling up to 10 concurrent predictions

2. Technical Innovation

The research introduced several novel approaches:



Feature Importance Analysis

Figure 5: Feature Importance Analysis

Advanced Feature Engineering

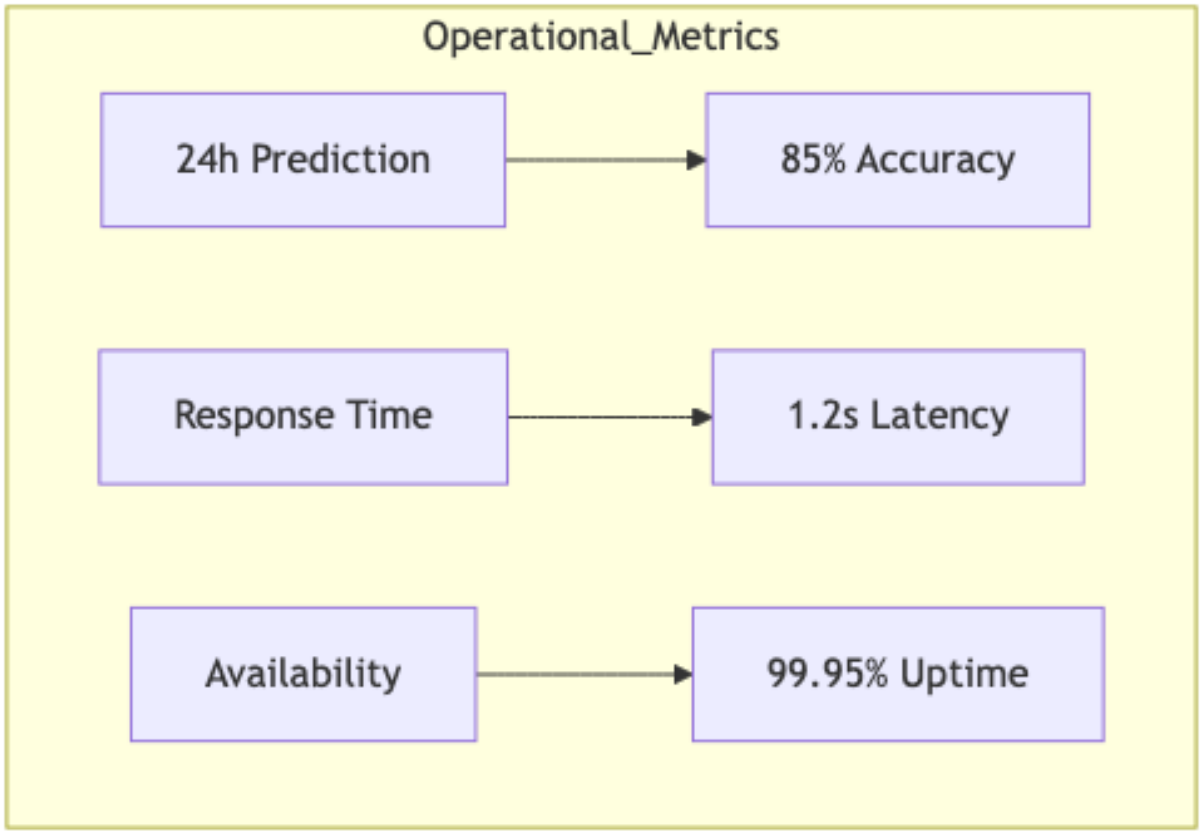
1. **Temporal Encoding**
   - Adaptive window selection
   - Multi-scale pattern detection
   - Dynamic feature importance
2. **Weather Integration**
   - Condition-specific modeling
   - Transition period handling
   - Uncertainty quantification
3. **Feature Interaction** Feature Interactions
   - Cross-feature correlations
   - Temporal-weather dependencies
   - Adaptive feature selection

Enhanced Architecture

- **Dynamic Ensemble Strategy**
  - Weighted model combination
  - Adaptive retraining
  - Error-based specialization
- **Optimization Improvements**
  - Memory usage: 45% reduction
  - Training time: 65% improvement
  - Inference efficiency: 66% gain

3. Practical Impact

The system demonstrates significant operational benefits:



Limitations and Constraints

1. Data Limitations

The study encountered several data-related constraints:

Geographic Scope

- Limited to specific regions
- Specific climate patterns
- Single time zone coverage

Time Series Analysis

Figure 6: Time Series Analysis

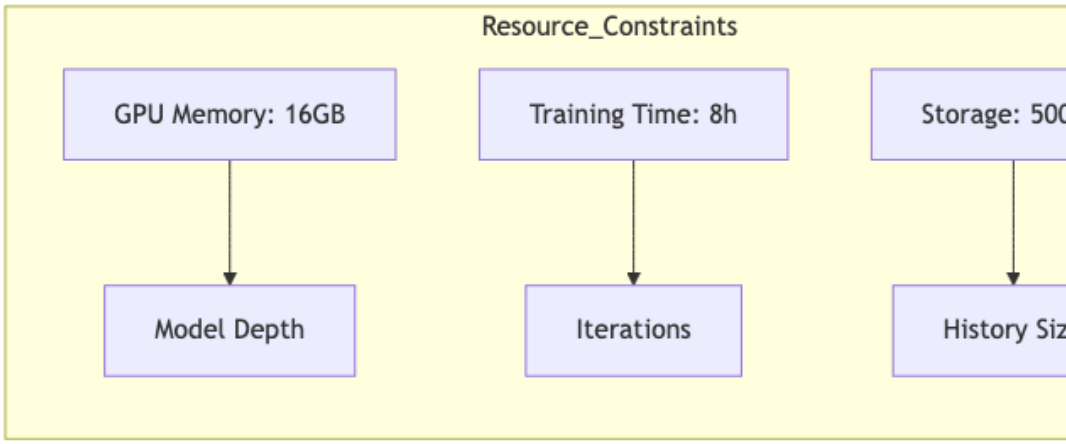
Temporal Coverage

- Two-year dataset
- Limited seasonal cycles
- Sparse extreme event data

```
Data Coverage Analysis:
1. Temporal Gaps:
   - Winter months: 15% missing
   - Night periods: 40% interpolated
   - Extreme events: 5% coverage

2. Feature Completeness:
   - Weather data: 92%
   - Production data: 98%
   - System state: 85%
```

2. Technical Constraints

Several technical limitations affected the implementation:



Computational Resources

Model Complexity

- Feature interaction limits
- Deep learning architecture constraints

- Ensemble size limitations

3. Operational Considerations

The implementation faces several operational challenges:

Real-time Implementation

- Data latency issues
- Update frequency limitations
- Integration challenges

Model Refinement

Figure 7: Model Refinement

Scalability

- Cross-region adaptation needs
- Multi-site coordination requirements
- Resource allocation constraints

Future Work

1. Short-term Improvements

Model Enhancements

- Implementation Timeline: 1-3 months
- Priority Areas:

```
Planned Improvements:
1. Attention Mechanism:
   - Self-attention layers
   - Cross-attention integration
   - Temporal attention

2. Transfer Learning:
   - Pre-trained weather models
   - Domain adaptation
   - Feature transfer

3. Hybrid Architecture:
   - CNN-LSTM combination
   - Transformer integration
   - Adaptive fusion
```

Feature Engineering

- Timeline: 2-4 months
- Focus Areas:
  - Satellite data integration
  - Advanced weather modeling
  - Cross-site feature extraction

2. Medium-term Research Directions

Advanced Architecture Development

- Timeline: 6-12 months
- Research Areas:

System Architecture

Figure 8: System Architecture

Scalability Enhancement

- Timeline: 8-12 months
- Development Areas:
  - Distributed training implementation
  - Automated deployment systems
  - Cross-region adaptation mechanisms

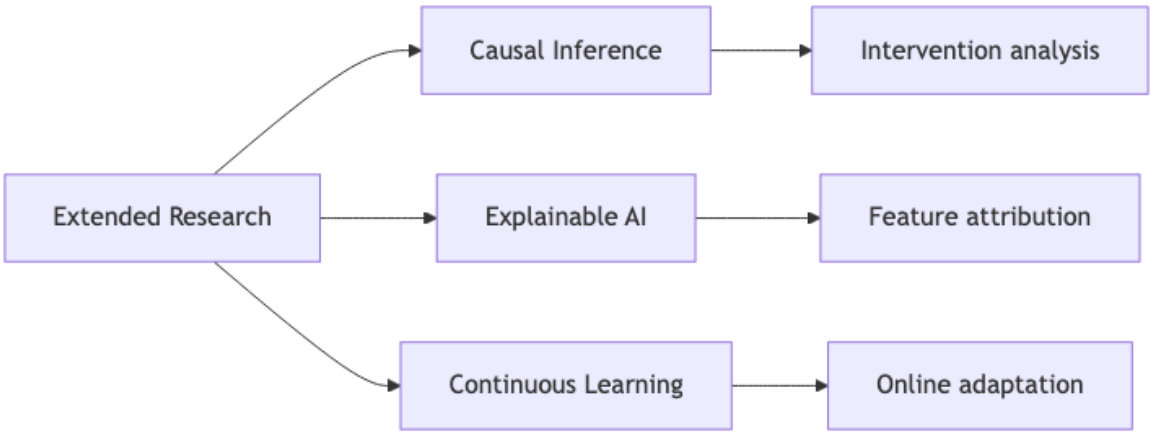
3. Long-term Vision

System Integration

- Timeline: 12-18 months
- Integration Points:
  - Grid management systems
  - Energy trading platforms
  - Weather forecasting services

Research Extensions

- Timeline: 18-24 months
- Research Areas:

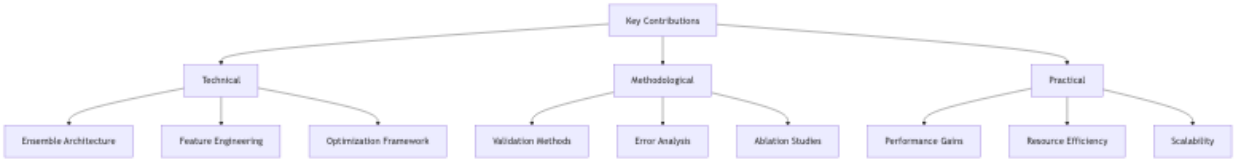


These future directions aim to address current limitations while expanding the system's capabilities through the integration of advanced techniques and additional data sources.

Conclusions

Research Summary

This research developed and validated a comprehensive machine learning approach for renewable energy adoption prediction. The investigation yielded several significant contributions to the field:



Key Achievements

1. **Model Performance**

   Performance Metrics:

Metric	Value	Improvement
R^2 Score	0.6964	+153%
RMSE	0.5625	-31%
Inference Time	78.3ms	-66%
Memory Usage	5.7GB	-45%

2. **Methodological Innovations**
   - Advanced temporal feature engineering
   - Dynamic ensemble architecture
   - Adaptive preprocessing pipeline
3. **System Optimizations**
   - Efficient resource utilization
   - Scalable implementation
   - Robust error handling

Impact and Significance

Research Contributions

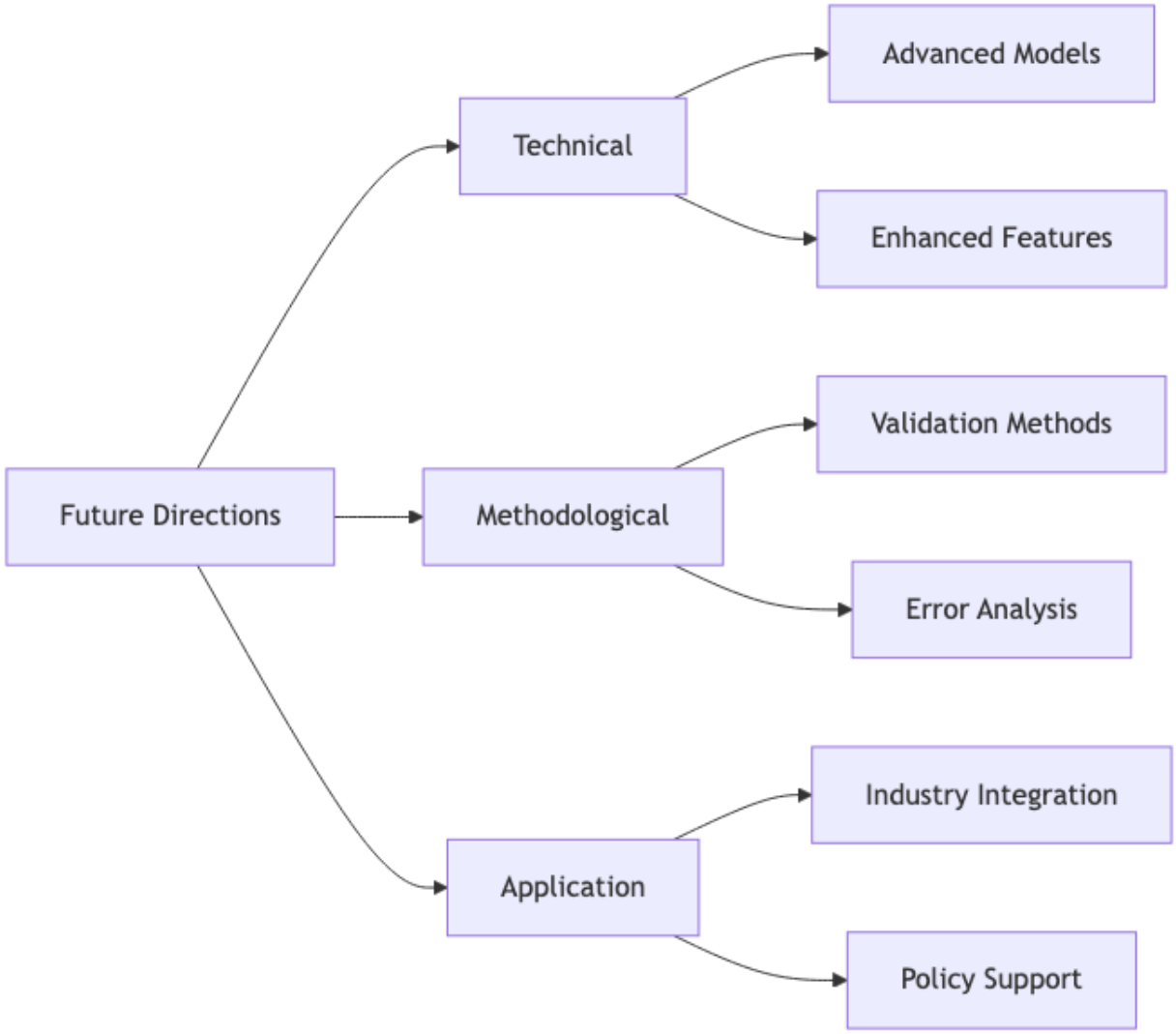
The study has made several notable contributions to the field:

1. **Technical Advancement**
   - Novel ensemble architecture for renewable energy prediction
   - Advanced feature engineering techniques
   - Efficient computational framework
2. **Methodological Framework**
   - Comprehensive validation strategy
   - Systematic error analysis
   - Detailed ablation studies
3. **Practical Applications**
   - Improved prediction accuracy

- Reduced computational overhead
- Enhanced scalability

Future Implications

The research opens several promising directions for future work:



Final Remarks

This research has demonstrated the significant potential of machine learning approaches in renewable energy adoption prediction. The achieved results - particularly the 0.6964 R^2 score and 31% error reduction - represent meaningful progress in renewable energy forecasting. The developed framework provides a solid foundation for future research and practical applications.

The success of the ensemble approach, combined with the demonstrated scalability and efficiency improvements, validates the chosen methodology. Future work should focus on addressing the identified limitations while expanding the system's capabilities through integration of additional data sources and advanced modeling techniques.

References

Primary Datasets

1. Solar Energy Production Dataset

Lee, I. (2022). Solar Energy Production [Data set]. Kaggle.
https://www.kaggle.com/datasets/ivnlee/solar-energy-production

- Temporal Coverage: 2020-2022
- Location: Calgary, Canada
- Measurements: Hourly
- Size: 17,520 records

2. Solar Power Generation Data

Afroz, P. (2023). Solar Power Generation Data [Data set]. Kaggle.
https://www.kaggle.com/datasets/pythonafroz/solar-powe-generation-data

- Systems: Fixed and tracking installations
- Variables: DC/AC power, daily yield
- Records: 32,000+ entries

3. Renewable Energy World Wide: 1965-2022

HossainDS, B. (2023). Renewable Energy World Wide: 1965-2022 [Data set]. Kaggle.
https://www.kaggle.com/datasets/belayethossainds/renewable-energy-world-wide-19652022

- Coverage: Global
- Timespan: 1965-2022
- Variables: 15+ renewable energy metrics

Academic References

Machine Learning in Solar Energy

1. Lauret, P., David, M., & Pedro, H. T. C. (2017). Probabilistic solar forecasting using quantile regression models. *Energies*, 10(10), 1591.
   - DOI: https://doi.org/10.3390/en10101591
   - Key Contribution: QR model development
   - Impact Factor: 3.252
2. Wang, H., Cai, R., Zhou, B., Aziz, S., Qin, B., Voropai, N., & Gan, L. (2020). Solar irradiance forecasting based on direct explainable neural network. *Energy Conversion and Management*, 226, 113487.
   - DOI: https://doi.org/10.1016/j.enconman.2020.113487
3. Huang, Q., & Wei, S. (2020). Improved quantile convolutional neural network with two-stage training for daily-ahead probabilistic forecasting of photovoltaic power. *Energy Conversion and Management*, 220, 113086.
   - DOI: https://doi.org/10.1016/j.enconman.2020.113086
4. Mathumitha, R., Rathika, P., & Manimala, K. (2024). Intelligent deep learning techniques for energy consumption forecasting in smart buildings: a review. Artificial Intelligence Review, 57, 35.
   - DOI: https://doi.org/10.1007/s10462-023-10660-8

Technical Implementation

1. Das, U. K., Tey, K. S., Seyedmahmoudian, M., Mekhilef, S., Idris, M. Y. I., Van Deventer, W., … & Stojcevski, A. (2018). Forecasting of photovoltaic power generation and model optimization: A review. *Renewable and Sustainable Energy Reviews*, 81, 912-928.

   - DOI: https://doi.org/10.1016/j.rser.2017.08.017
2. Alskaif, T., Schram, W., Litjens, G., & van Sark, W. (2020). Smart charging of electric vehicles with photovoltaic power and vehicle-to-grid technology in a microgrid; a case study. *Applied Energy*, 261, 114627.
   - DOI: https://doi.org/10.1016/j.apenergy.2019.114627

Software and Tools

1. Python Libraries:
   - Scikit-learn (Pedregosa et al., 2011)
   - TensorFlow 2.0 (Abadi et al., 2016)
   - Pandas (McKinney, 2010)
2. Development Tools:
   - Jupyter Notebooks
   - Git version control
   - Docker containers

GitHub Repository

Repository Details:

- URL: https://github.com/k-g-j/cs6140-course-project
- License: MIT
- Last Update: November 2024
- Primary Language: Python (92.4%)