Fusion Forecasting for Residential PV: Reference Implementation Documentation

Erik Schmidt
erik@f97.io

August 28, 2025

Abstract

This document provides technical documentation for the reference implementation of Fusion Forecasting, an adaptive regression framework that blends AI-based and physics-based (GTI) solar forecasts with online calibration. The implementation follows the methodology described in the original paper, providing a complete, operational system for residential PV forecasting, including a command-line interface and detailed data specifications.

Contents

1 Introduction

Fusion Forecasting addresses the challenge of photovoltaic (PV) output forecasting by combining the strengths of AI-based and physics-based approaches while mitigating their individual weaknesses. This reference implementation provides a complete, operational system based on the methodology described in the original paper.

The system consists of four core components:

1. Linear blending of AI and GTI forecasts
2. Robust regression with time-series cross-validation
3. Online calibration using median-ratio scaling
4. Uncertainty quantification with robust statistics

This enhanced implementation adds a comprehensive command-line interface and detailed specifications for data input formats.

2 Mathematical Formulation

2.1 Linear Blending Layer

The fusion model predicts daily energy production through an affine combination:

$$\widehat{y}_{\text{lin}}(t) = w_{\text{AI}} \cdot \widehat{y}_{\text{AI}}(t) + w_{\text{GTI}} \cdot \widehat{y}_{\text{GTI}}(t) + b$$

where:

- $\widehat{y}_{\text{AI}}(t)$ is the AI-based forecast
- $\widehat{y}_{\text{GTI}}(t)$ is the GTI-based forecast
- $w_{\text{AI}}, w_{\text{GTI}}$ are learned weights
- b is the intercept term

2.2 Regression Objective

Parameters are learned by minimizing a robust, regularized objective function:

$$\min_{\theta} \frac{1}{N} \sum_{t \in \mathcal{T}_{\text{train}}} \rho_{\delta} \left(y(t) - \widehat{y}_{\text{lin}}(t) \right) + \lambda \left(w_{\text{AI}}^2 + w_{\text{GTI}}^2 \right)$$

where $\rho_{\delta}(\cdot)$ is the Huber loss function and λ is the ridge regularization parameter.

2.3 Online Calibration

The calibration scale factor is computed as:

$$\text{scale} = \text{median} \left(\text{winsorize}_{\alpha} \left[\frac{y(d)}{\widehat{y}_{\text{lin}}(d)} \right]_{d \in \mathcal{D}_K} \right)$$

where $\mathcal{D}_K$ contains the most recent K days with valid data, and winsorization reduces the influence of outliers.

2.4 Uncertainty Quantification

Prediction uncertainty is estimated using Median Absolute Deviation (MAD):

$$\widehat{\sigma} = 1.4826 \cdot \mathrm{MAD}\left(\{\varepsilon(d)\}_{d \in \mathcal{D}_K}\right)$$

where $\varepsilon(d) = y(d) - \widehat{y}_{\text{fusion}}(d)$.

3 Implementation Architecture

3.1 Class Structure

The implementation is organized around the `FusionForecaster` class, which encapsulates the complete forecasting pipeline.

```
class FusionForecaster:
    def __init__(self,
                 train_window_size=90,
                 calibration_window_size=14,
                 alpha=0.05,
                 p_max=10.0,
                 min_train_samples=30,
                 fallback_weights=(0.7, 0.3)):
        # Initialization code
```

Listing 1: FusionForecaster class initialization

3.2 Core Methods

3.2.1 Model Training

The `train_fusion` method implements the training procedure:

Algorithm 1 Model Training Procedure

procedure TRAIN_FUSION($\mathcal{D}_{\text{train}}$)
  if $|\mathcal{D}_{\text{train}}| < N_{\min}$ **then**
    return Fallback weights $(0.7, 0.3)$
  end if
  Prepare feature matrix X and target vector y
  Perform time-series cross-validation to select λ
  Fit Huber regression model with best λ
  return learned parameters $w_{\text{AI}}, w_{\text{GTI}}, b$
end procedure

3.2.2 Online Calibration

The `calibrate_scale` method implements online calibration:

3.2.3 Forecast Generation

The `forecast_day` method generates predictions:

3.2.4 Uncertainty Estimation

The `estimate_uncertainty` method quantifies prediction uncertainty:

4 Command-Line Interface

The enhanced implementation includes a comprehensive command-line interface for operational use.

Algorithm 2 Online Calibration Procedure

procedure CALIBRATE_SCALE($\mathcal{D}_{\text{recent}}$)
  Initialize empty ratio list R
  for each day $d \in \mathcal{D}_{\text{recent}}$ **do**
    if $y(d) > 0$ and $\widehat{y}_{\text{lin}}(d) > 0$ **then**
      $r(d) \leftarrow y(d)/\widehat{y}_{\text{lin}}(d)$
      Append $r(d)$ to R
    end if
  end for
  if $|R| < \max(3, 0.3 \cdot K)$ **then**
    return previous scale
  end if
  $R_{\text{winsorized}} \leftarrow \text{winsorize}(R, \alpha)$
  return $\text{median}(R_{\text{winsorized}})$
end procedure

Algorithm 3 Forecast Generation Procedure

procedure FORECAST_DAY($\widehat{y}_{\text{AI}}, \widehat{y}_{\text{GTI}}, \text{daylight\_hours}$)
  $\widehat{y}_{\text{lin}} \leftarrow w_{\text{AI}} \cdot \widehat{y}_{\text{AI}} + w_{\text{GTI}} \cdot \widehat{y}_{\text{GTI}} + b$
  $\widehat{y}_{\text{fusion}} \leftarrow \text{scale} \cdot \widehat{y}_{\text{lin}}$
  $\widehat{y}_{\text{fusion}} \leftarrow \max(0, \min(\widehat{y}_{\text{fusion}}, P_{\max} \cdot \text{daylight\_hours}))$
  return $\widehat{y}_{\text{fusion}}$
end procedure

Algorithm 4 Uncertainty Estimation Procedure

procedure UNCERTAINTY($\mathcal{D}_{\text{recent}}$)
  Initialize empty error list E
  for each day $d \in \mathcal{D}_{\text{recent}}$ **do**
    if valid data exists for day d **then**
      $\varepsilon(d) \leftarrow y(d) - \widehat{y}_{\text{fusion}}(d)$
      Append $\varepsilon(d)$ to E
    end if
  end for
  if $|E| < 5$ **then**
    return None
  end if
  $\text{MAD} \leftarrow \text{median}(|E - \text{median}(E)|)$
  $\widehat{\sigma} \leftarrow 1.4826 \cdot \text{MAD}$
  return $\widehat{\sigma}$
end procedure

4.1 Usage Examples

```
# Basic usage with a CSV file
python fusion_forecast.py --train-data historical_data.csv --forecast-dates
    2023-08-01,2023-08-02

# With custom parameters
python fusion_forecast.py --train-data data.csv --train-window 60 --calibration-window
    14 --p-max 8.5 --forecast-dates 2023-08-01

# Save results to a JSON file
python fusion_forecast.py --train-data data.csv --forecast-dates 2023-08-01 --output
    results.json

# Verbose output
python fusion_forecast.py --train-data data.csv --forecast-dates 2023-08-01 --verbose
```

Listing 2: Command-line usage examples

4.2 Command-Line Options

Option	Default	Description
–train-data	(required)	CSV file with historical training data
–forecast-dates	(required)	Dates to forecast (YYYY-MM-DD, comma-separated)
–output	None	Output file for results (JSON format)
–train-window	90	Days used for training
–calibration-window	14	Days used for calibration
–p-max	10.0	System capacity (kW)
–min-train-samples	30	Minimum samples for training
–alpha	0.05	Winsorization parameter
–fallback-ai-weight	0.7	Fallback weight for AI forecast
–fallback-gti-weight	0.3	Fallback weight for GTI forecast
–retrain-frequency	30	Retraining frequency (days)
–confidence	0.95	Confidence level for prediction intervals
–verbose	False	Enable verbose output

Table 1: Command-line options

5 Data Requirements and CSV Format

5.1 CSV File Specification

The system requires a CSV file with historical data for training. The file must contain the following columns:

Column	Type	Description
date	Date (YYYY-MM-DD)	Date of observation
y_ai	Numeric (kWh)	AI-based forecast
y_gti	Numeric (kWh)	GTI-based forecast
y_actual	Numeric (kWh)	Actual energy production
daylight_hours	Numeric (hours)	Number of daylight hours

Table 2: CSV file column specifications

5.2 Example CSV File

```
date,y_ai,y_gti,y_actual,daylight_hours
2023-01-01,8.5,7.2,8.1,9.5
```

```
2023-01-02,9.2,8.1,9.5,9.6
2023-01-03,7.8,6.9,7.2,9.6
2023-01-04,6.5,5.8,6.8,9.7
2023-01-05,10.2,9.5,10.5,9.7
```

Listing 3: Example CSV data format

5.3 Output Format

The program outputs results in JSON format with the following structure:

```
[
  {
    "date": "2023-08-01",
    "forecast": 12.45,
    "uncertainty": 1.23,
    "w_ai": 0.65,
    "w_gti": 0.42,
    "b": -0.32,
    "scale": 1.05,
    "prediction_interval": {
      "lower": 10.12,
      "upper": 14.78,
      "confidence": 0.95
    }
  }
]
```

Listing 4: Output JSON format

6 Implementation Details

6.1 Dependencies

The implementation requires the following Python packages:

- numpy $\geq$ 1.20.0
- pandas $\geq$ 1.3.0
- scikit-learn $\geq$ 1.0.0
- scipy $\geq$ 1.7.0

6.2 Key Parameters

Parameter	Default	Description
train_window_size	90	Days used for training
calibration_window_size	14	Days used for calibration
alpha	0.05	Winsorization parameter
p_max	10.0	System capacity (kW)
min_train_samples	30	Minimum samples for training
fallback_weights	(0.7, 0.3)	Weights when falling back

Table 3: Key configuration parameters

7 Usage Example

7.1 Basic Usage

```python
# Initialize the forecaster
forecaster = FusionForecaster(
    train_window_size=60,
    calibration_window_size=14,
    p_max=10.0  # System capacity: 10 kW
)

# Add historical observations
for i in range(len(historical_data)):
    forecaster.add_observation(
        date=dates[i],
        y_ai=ai_forecasts[i],
        y_gti=gti_forecasts[i],
        y_actual=actual_production[i],
        daylight_hours=daylight_hours[i]
    )

# Generate a forecast
forecast = forecaster.forecast_day(
    y_ai=next_ai_forecast,
    y_gti=next_gti_forecast,
    daylight_hours=next_daylight_hours
)

# Get prediction interval
point_forecast, lower, upper = forecaster.prediction_interval(
    y_ai=next_ai_forecast,
    y_gti=next_gti_forecast,
    daylight_hours=next_daylight_hours,
    confidence=0.95
)
```

Listing 5: Basic usage example

7.2 Daily Operation

For operational use, the `daily_operation` method provides a complete workflow:

```python
# Complete daily operation
result = forecaster.daily_operation(
    date=current_date,
    y_ai=ai_forecast,
    y_gti=gti_forecast,
    daylight_hours=daylight_hours,
    y_actual=actual_production,  # If available
    retrain_frequency=30  # Retrain every 30 days
)

# Result contains forecast, uncertainty, and model parameters
print(f"Forecast: {result['forecast']:.2f} kWh")
print(f"Uncertainty: {result['uncertainty']:.2f} kWh")
print(f"95% Prediction Interval: [{result['lower_95']:.2f}, {result['upper_95']:.2f}]")
```

Listing 6: Daily operational workflow

8 Performance Considerations

8.1 Computational Complexity

The implementation is designed for efficiency on commodity hardware:

- Training: $O(N \cdot M)$ where N is training window size and M is number of λ values tested
- Calibration: $O(K)$ where K is calibration window size
- Forecasting: $O(1)$ per forecast

8.2 Memory Requirements

Memory usage is primarily determined by:

- History storage: $O(T)$ where T is the maximum of training and calibration windows
- Model parameters: Constant size (3 weights + scale factor)

9 Extensions and Customization

The implementation can be extended in several ways:

9.1 Custom Loss Functions

The Huber loss can be replaced with other robust loss functions:

```
from sklearn.linear_model import RANSACRegressor

# Use RANSAC for more robust regression
model = RANSACRegressor(
    base_estimator=Ridge(alpha=best_lambda),
    min_samples=0.8
)
```

Listing 7: Custom loss function example

9.2 Additional Features

The model can be extended to include additional features:

```
# Extended feature vector at time t
# The variables are assumed to be defined elsewhere in the code.
x_t = [
    y_hat_ai(t),    # AI model's prediction
    y_hat_gti(t),   # GTI model's prediction
    temperature(t), # Temperature reading
    humidity(t)     # Humidity reading
]
# x_t is now a list that can be used as a feature vector.
```

Listing 8: Creating an extended feature vector

9.3 Advanced Calibration

The calibration procedure can be enhanced with adaptive window sizing or different robust estimators.

10 Conclusion

This reference implementation provides a complete, operational version of the Fusion Forecasting system described in the original paper. The system combines AI-based and physics-based forecasts using adaptive regression blending with online calibration, resulting in improved accuracy and robustness for residential PV forecasting.

The enhanced implementation includes a comprehensive command-line interface and detailed data specifications, making it suitable for operational deployment. The implementation is designed for practical use, with careful attention to operational safeguards, missing data handling, and computational efficiency. The modular architecture allows for customization and extension to meet specific requirements of different PV forecasting applications.