Fusion Forecasting for Residential PV: An Adaptive Regression Blend of AI and GTI with Online Calibration

Erik Schmidt
erik@f97.io

August 2025

Abstract

I present Fusion Forecasting, an adaptive regression framework that blends AI-based and physics-based (GTI) solar forecasts, followed by online calibration. The method addresses weaknesses of individual models by dynamically learning weights and correcting bias using recent data. This paper describes the motivation, mathematical formulation, and system design, with emphasis on practical deployment for residential PV forecasting.

1 Introduction

Forecasting photovoltaic (PV) output is a challenge due to the variability of weather and local conditions. Traditionally, Global Tilted Irradiance (GTI) models provide estimates based on solar geometry and irradiance forecasts. More recently, machine learning (ML) models trained on historical data have been applied. Each has strengths and weaknesses: physics-based GTI is stable but blind to local quirks, while AI can adapt to local patterns but may overreact to unusual conditions.

To overcome these limitations, I propose Fusion Forecasting: a regression-based ensemble that learns the optimal combination of AI and GTI forecasts, applies an intercept to reduce bias, and uses online calibration for robustness.

2 Fusion Architecture

Inputs and Notation

Let t index calendar days. Denote by $\widehat{y}_{\mathrm{AI}}(t) \in \mathbb{R}_{\geq 0}$ the calibrated machine–learning forecast, $\widehat{y}_{\mathrm{GTI}}(t) \in \mathbb{R}_{\geq 0}$ the physics–based irradiance forecast, and $y(t) \in \mathbb{R}_{\geq 0}$ the realized daily production from the inverter. Define the feature vector $x(t) = \begin{bmatrix} \widehat{y}_{\mathrm{AI}}(t) & \widehat{y}_{\mathrm{GTI}}(t) \end{bmatrix}^{\top}$.

Linear Blending Layer

Fusion predicts a pre-calibrated production via an affine map

$$\widehat{y}_{\mathrm{lin}}(t) \;=\; w_{\mathrm{AI}}\,\widehat{y}_{\mathrm{AI}}(t) \;+\; w_{\mathrm{GTI}}\,\widehat{y}_{\mathrm{GTI}}(t) \;+\; b, \tag{1}$$

with parameters $\theta = \{w_{\mathrm{AI}}, w_{\mathrm{GTI}}, b\}$. Unlike a fixed convex blend, Fusion does *not* require $w_{\mathrm{AI}} + w_{\mathrm{GTI}} = 1$ or $w_{\cdot} \geq 0$; the affine form is strictly more general and can correct systematic scale/offset biases in either input.

Estimation Objective (Time-Series Aware)

Parameters are learned from a historical window $\mathcal{T}_{\text{train}} = \{t_1, \ldots, t_N\}$ by minimizing a robust, regularized objective

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

where $\rho_\delta(\cdot)$ is the Huber loss (quadratic near 0, linear in the tails), and $\lambda \geq 0$ is an L_2 (ridge) penalty to reduce variance and prevent pathological weights when inputs are collinear. In practice I use a time-series split (forward chaining) for cross-validation to tune λ and to select the training span N that balances recency and sample size.

Optional constraints. If desired, soft constraints can be added:

$$w_{\text{AI}}, w_{\text{GTI}} \geq 0 \quad \text{(non-negativity)} \qquad \text{or} \qquad w_{\text{AI}} + w_{\text{GTI}} \approx 1 \quad \text{(via penalty)}.$$

I generally prefer the unconstrained affine form and let calibration (below) handle residual scale.

Online Calibration (Median-Ratio Scaling)

Raw affine predictions can drift seasonally or inherit residual biases. I correct them with a robust, one-parameter scale estimated from the most recent K days with reliable data:

$$\mathcal{D}_K = \big\{ d \in \text{last } K \text{ days} \;\big|\; y(d) > 0,\ \widehat{y}_{\text{lin}}(d) > 0 \big\}, \tag{3}$$

$$r(d) = \frac{y(d)}{\widehat{y}_{\text{lin}}(d)}, \qquad d \in \mathcal{D}_K, \tag{4}$$

$$\text{scale} = \text{median}\big(\text{winsorize}_\alpha[\, r(d) \,] \big). \tag{5}$$

Winsorization at small tails α (e.g., 5%) dampens the influence of outliers (sensor dropouts, extreme haze). The final forecast is

$$\widehat{y}_{\text{fusion}}(t) \;=\; \text{scale} \cdot \widehat{y}_{\text{lin}}(t). \tag{6}$$

If $|\mathcal{D}_K|$ is too small, I keep the last valid scale or revert to scale = 1.

Fallback and Cold-Start Behavior

When N (training samples) is below a minimum threshold or the regression fit becomes ill-conditioned (e.g., near-collinear inputs over a short span), I bypass (2) and revert to a proven convex blend:

$$\widehat{y}_{\text{lin}}^{\text{fallback}}(t) \;=\; 0.7\, \widehat{y}_{\text{AI}}(t) \;+\; 0.3\, \widehat{y}_{\text{GTI}}(t) \;+\; b_0, \tag{7}$$

with b_0 set to the median residual over the last available window (or 0 if unavailable), followed by the same median-ratio calibration.

Uncertainty Quantification

I report day-ahead uncertainty using residual dispersion on a rolling window. The uncertainty is calculated on the final calibrated forecasts $\widehat{y}_{\text{fusion}}$:

$$\varepsilon(d) = y(d) - \widehat{y}_{\text{fusion}}(d), \tag{8}$$

$$\widehat{\sigma} = 1.4826 \cdot \text{MAD}\big(\{\varepsilon(d)\}_{d \in \mathcal{D}_K} \big), \tag{9}$$

and form symmetric prediction bands

$$\widehat{y}_{\text{fusion}}(t) \pm z_p\, \widehat{\sigma}, \qquad z_{0.84} \approx 1.0,\ z_{0.975} \approx 1.96. \tag{10}$$

This simple, distribution-free approach is robust and adequate for operational dashboards; more elaborate conformal or quantile methods are possible future work.

Operational Safeguards

I enforce physically plausible ranges post-hoc:

$$0 \;\le\; \widehat{y}_{\text{fusion}}(t) \;\le\; P_{\max} \cdot \text{daylight\_hours}(t),$$

and ignore calibration updates on days flagged as incomplete or with inverter anomalies. Missing inputs are imputed conservatively (carry-forward for AI; clear-sky capped proxy for GTI) and flagged to the user.

Relation to the Hybrid Baseline

The fixed Hybrid method $\widehat{y}_{\text{hyb}} = \alpha\,\widehat{y}_{\text{AI}} + (1-\alpha)\,\widehat{y}_{\text{GTI}}$ with $\alpha = 0.7$ is a special case of (1) with $b = 0$, $w_{\text{AI}} = \alpha$, $w_{\text{GTI}} = 1-\alpha$, and no learning. Fusion generalizes this by learning $\{w., b\}$ from data and adding an explicit online calibration step (6), which empirically reduces bias and improves stability under regime shifts.

3 Implementation and Deployment

The Fusion model is implemented in Python using `scikit-learn` for regression, `pandas` for data handling, and runs efficiently on commodity hardware. Online calibration uses a rolling 14-day window, updating scale factors continuously as new production data arrives.

3.1 Pseudocode of Training and Inference

To make the Fusion framework fully transparent and reproducible, I outline its main operations in pseudocode. The workflow is divided into four core procedures:

- **TRAIN_FUSION:** learns the regression weights $w_{\text{AI}}, w_{\text{GTI}}$ and the intercept b from historical data using a robust ridge regression. If too few training samples are available, the procedure falls back to the fixed 70/30 split.
- **CALIBRATE_SCALE:** performs online calibration by computing a robust median ratio between observed and predicted values over a recent rolling window, stabilizing forecasts against short-term drift.
- **FORECAST_DAY:** generates the daily Fusion forecast by applying the learned weights and calibration factor, while clamping the output within physical plausibility bounds.
- **UNCERTAINTY:** estimates predictive uncertainty via robust dispersion (MAD) of residuals over recent days, enabling interval forecasts.

These building blocks together form the daily operation loop: retraining when necessary, recalibrating scale, producing forecasts, and quantifying uncertainty. The following pseudocode captures these steps in detail:

```
PROCEDURE TRAIN_FUSION(D_train):
  # D_train = {t} indices with valid AI, GTI, actual
  if |D_train| < N_min:
      # Fallback to fixed 70/30 weights if insufficient data
      w_AI, w_GTI = 0.7, 0.3
      # Intercept = median residual of fallback model
      b = median_over(D_train)[ y - (0.7*y_hat_AI + 0.3*y_hat_GTI) ]
      # Note: median_over([]) is defined to return 0 for this fallback.
```

```
      return (w_AI, w_GTI, b)

  # Build design matrix and target
  X = [ y_hat_AI, y_hat_GTI ]  # two columns
  y = actual

  # Time-series CV (forward chaining) over D_train
  # **to choose lambda in Lambda**
  best_lambda = argmin_over_lambda_in(Lambda)
                  mean_CV_Huber_Ridge_MAE(X, y, lambda=lambda)

  # Fit final model on all D_train with best_lambda
  (w_AI, w_GTI, b) = fit_Huber_Ridge(X, y, lambda=best_lambda)
  return (w_AI, w_GTI, b)


PROCEDURE CALIBRATE_SCALE(D_recent, w_AI, w_GTI, b, alpha, prev_scale):
  # D_recent = most-recent K days with valid inputs & actuals
  R = empty list
  for d in D_recent:
      y_lin = w_AI*y_hat_AI(d) + w_GTI*y_hat_GTI(d) + b
      if y(d) > 0 and y_lin > 0:
          R.append( y(d) / y_lin )

  # **Require at least 3 points or 30% of the window**
  if length(R) < max(3, ceil(0.3*K)):
      return (prev_scale if prev_scale is not None else 1.0)

  # Winsorize ratios to reduce outlier impact
  R_w = winsorize(R, lower=alpha, upper=1-alpha)
  return median(R_w)


PROCEDURE FORECAST_DAY(t, w_AI, w_GTI, b, scale, P_max):
  y_lin    = w_AI*y_hat_AI(t) + w_GTI*y_hat_GTI(t) + b
  y_fusion = scale * y_lin

  # Physical plausibility clamp
  y_fusion = max(0, min(y_fusion, P_max * daylight_hours(t)))
  return y_fusion


PROCEDURE UNCERTAINTY(D_recent, w_AI, w_GTI, b, scale):
  # Rolling residual dispersion via robust MAD
  # Calculates error on the final, calibrated forecast
  E = empty list
  for d in D_recent:
      y_lin = w_AI*y_hat_AI(d) + w_GTI*y_hat_GTI(d) + b
      y_fus = scale * y_lin
      # Use same validity criteria as calibration for robustness
      if y(d) is not None and y(d) > 0 and y_fus > 0:
```

```
          E.append( y(d) - y_fus )

  # Avoid estimating sigma from too few samples
  if length(E) < 5:
      return sigma = None

  MAD   = median( |E - median(E)| )
  sigma = 1.4826 * MAD      # Convert MAD to robust sigma estimate
  return sigma


# Daily operation (end-to-end):
# 1) If retraining day or drift detected:
#       (w_AI, w_GTI, b) = TRAIN_FUSION(D_train)
# 2) scale = CALIBRATE_SCALE(D_recent, w_AI, w_GTI, b, alpha, prev_scale)
# 3) y_hat_fusion(t) = FORECAST_DAY(t, w_AI, w_GTI, b, scale, P_max)
# 4) sigma = UNCERTAINTY(D_recent, w_AI, w_GTI, b, scale)
# 5) Report y_hat_fusion(t) and optionally
#       y_hat_fusion(t) ± z_p * sigma
```

4 Conclusion

I presented Fusion Forecasting: a regression-based blend of AI and GTI forecasts with online calibration. This approach improves robustness, reduces bias, and adapts dynamically to changing conditions. Unlike Hybrid, Fusion does not enforce weights to sum to 1, enabling more flexible modeling. Together with safeguards in calibration, this yields a practical yet accurate forecasting method. It is efficient enough for embedded deployment while providing accuracy gains that are valuable for residential energy planning.

Acknowledgements

I thank the open-source community for tools such as scikit-learn and pandas, and the renewable energy research community for inspiration.

References

[1] Pedregosa, F. et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

[2] Hyndman, R. J., & Athanasopoulos, G. (2008). Forecasting: Principles and Practice. OTexts.

[3] Antonanzas, J., Osorio, N., Escobar, R., Urraca, R., de Pison, F. M., & Antonanzas-Torres, F. (2016). Review of photovoltaic power forecasting. *Solar Energy*, 136, 78–111.

[4] Richardson, M. & Wallace, S. (2012). Getting Started with Raspberry Pi. O'Reilly Media.