# Supplementary Information

# Machine learning-guided scalable manufacturing of high-efficiency perovskite solar modules

*Yuchen Bu[1], Junjie Gu[1], Yu Shen[1], Bairu Li[1], Cheng Hu[2,]*, Jianning Ding[1,]* and Long Luo[1,]**

[1]Y. Bu, J. Gu, Y. Shen, B. Li, J. Ding, L. Luo
College of Physical Science and Technology, Institute of Technology for Carbon Neutralization, Yangzhou University, Jiangsu 225009, China
[2]Dr. C. Hu
Business school, Institute of Technology for Carbon Neutralization, Yangzhou University, Jiangsu 225009, China

*Corresponding author:
longluo@yzu.edu.cn; dingjn@yzu.edu.cn; hucheng@yzu.edu.cn

## Dataset construction

We compiled a dataset of 339 data points extracted from 113 peer-reviewed articles published between 2016 and 2025, with reported power conversion efficiencies (PCEs) ranging from 4.2% to 26.93%. The data were systematically collected using the keyword "perovskite solar modules," covering both formal (*n-i-p*) and inverted (*p-i-n*) device architectures, as well as six fabrication techniques: spin-coating, blade-coating, slot-die coating, evaporation, inkjet printing, and vacuum-assisted solution processing (VASP). The dataset comprises 255 large-area devices (active area ≥ 10 $cm^2$) and 77 related small-area devices (active area < 10 $cm^2$), with each entry characterized by 23 input features.

Following initial data compilation, a rigorous multi-step preprocessing protocol was applied, resulting in a refined dataset of 332 valid entries. The preprocessing procedure included the following steps: (1) Removal of statistical outliers that deviated significantly from the majority of the data distribution. (2) Exclusion of samples with excessive missing feature values that could not be reliably imputed. (3) Recalculation and standardization of perovskite chemical compositions based on precursor ion stoichiometries. (4) Imputation of missing continuous variables, such as perovskite component ratios, using mean values to enhance data completeness. (5) Conversion of categorical charge transport layer materials into continuous variables by assigning representative conduction and valence band energy values. (6) Systematic classification of diverse additives and passivators into four major functional categories according to their chemical characteristics and roles in device operation.

## Model selection

**Light Gradient Boosting Machine (LightGBM) algorithm:** LightGBM is a high-efficiency open-source framework based on gradient boosting decision trees. It is designed for large-scale data and high-speed training, and while maintaining high accuracy, it significantly improves training efficiency and reduces memory usage. Compared with other gradient boosting libraries, LightGBM's most significant feature is that it adopts a leaf-growing decision tree growth strategy. It selects the leaf node

with the largest gain for growth at each split, thus reducing the loss function more quickly.

**Support vector regression (SVR) algorithm:** SVR is a regression method based on support vector machines. The core goal is to find a function that allows for an error band of ε precision when fitting data, and strives to make the function as flat as possible. It mainly uses the samples outside the error band to determine the decision boundary through optimization process. This method can effectively model the complex nonlinear relationship between features and targets by introducing kernel functions.

**Random forest (RF) algorithm:** Random forest is an ensemble learning algorithm that constructs multiple decision trees and aggregates their prediction results. It adopts a dual random strategy during training: Bootstrap sampling and feature random selection, effectively improving model diversity and generalization ability, and effectively suppressing overfitting.

**Multilayer perceptron (MLP) algorithm:** MLP is a feedforward neural network that consists of multiple layers. It consists of an input layer, at least one hidden layer, and an output layer, connected by adjustable weights between layers. The model calculates the output through forward propagation and updates the weight parameters using the backpropagation algorithm. This structure enables MLP to automatically learn complex nonlinear transformations between input and output, and has strong function approximation capabilities.

**Adaptive boost (AdaBoost) algorithm:** AdaBoost is an iterative ensemble learning algorithm that trains a series of weak learners sequentially, adjusting sample weights based on the previous round's prediction results each time, so that subsequent learners focus more on the previously incorrectly predicted samples. Finally, the prediction results of all weak learners are integrated through weighted voting.

**Extreme gradient boosting (XGBoost) algorithm:** XGBoost is an optimization implementation based on the gradient boosting framework, which constructs a sequence of decision trees in sequence, with each tree dedicated to correcting the prediction error of the preceding model. Its innovation lies in using second-order Taylor expansion to approximate the loss function and introducing regularization terms to

control model complexity. This algorithm has the advantages of high computational efficiency and strong ability to prevent overfitting. Furthermore, its sparsity-aware split finding enables robust performance even with small sample sizes by automatically learning handling strategies for missing or sparse data. Additionally, XGBoost effectively addresses imbalanced data distributions through customizable scale weights, ensuring high predictive accuracy and generalization in highly skewed datasets.

**Model evaluation**

**Coefficient of determination ($R^2$):** $R^2$ represents the degree to which the quantitative regression model explains the variance changes of the target variable. Its numerical value represents the proportion of the total variance of the dependent variable that can be explained by the independent variables in the model. The value range of $R^2$ is [0,1]. The closer the value is to 1, the better the model's fit to the data, indicating that the model can explain more data fluctuations. On the contrary, if $R^2$ is low, it means that the model has failed to effectively capture the inherent variability of the data.

**Root mean square error (RMSE):** RMSE is the square root of the mean sum of squared differences between predicted and true values, with the same unit as the original data. Due to squaring during the calculation process, it has a stronger penalty for larger prediction errors and is therefore more sensitive to outliers.

**Mean absolute percentage error (MAPE):** MAPE evaluates the relative prediction error of the model by calculating the average ratio of the absolute error of each data point to the corresponding observation value. It is expressed in percentage form, making it a dimensionless indicator that facilitates comparison between prediction tasks at different scales or units. The lower the MAPE value, the smaller the average relative error of the model.

## Code abstract

**Model training**

```
import shap
import pandas as pd
```

```
import numpy as np
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_percentage_error, accuracy_score, classification_report,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rcParams
import warnings
from sklearn.utils import resample
from sklearn.model_selection import RandomizedSearchCV, KFold
from sklearn.isotonic import IsotonicRegression
import xgboost as xgb
from tqdm import tqdm
warnings.filterwarnings('ignore')
rcParams['font.family'] = 'Times New Roman'
rcParams['mathtext.fontset'] = 'stix'
df = pd.read_excel("data set.xlsx", sheet_name="Sheet1")
columns_to_drop = [df.columns[0], df.columns[-1]]
for col in df.columns:
    if 'FF1' in str(col): columns_to_drop.append(col)
df = df.drop(columns=columns_to_drop)
df.columns = [col.strip() for col in df.columns]
df['Cation_sum'] = df[['Cs', 'FA', 'MA']].sum(axis=1)
df['Cs'] /= df['Cation_sum']; df['FA'] /= df['Cation_sum']; df['MA'] /= df['Cation_sum']
df = df.drop(columns=['Cation_sum'])
def adjust_halogens(row):
    br, i, cl = row['Br'], row['I'], row['Cl']
    if br > 0 and cl > 0:
        if br >= cl: cl = 0
        else: br = 0
    total = br + i + cl
    if total > 0:
        scale = 3 / total
        br *= scale; i *= scale; cl *= scale
    return pd.Series([br, i, cl], index=['Br', 'I', 'Cl'])
df[['Br', 'I', 'Cl']] = df.apply(adjust_halogens, axis=1)
additive_1_values = ['none', 'Organic ammonium salt', 'others', 'Inorganic salts']
additive_2_values = ['none', 'others', 'Inorganic salts']
passivator_values = ['none', 'Organic ammonium salts', 'others', 'polymer', 'Inorganic
molecules']
for col, vals in {'additive_1': additive_1_values, 'additive_2': additive_2_values,
'Surface passivator': passivator_values}.items():
    df[col] = df[col].fillna('none')
    for v in vals:
```

```
            df[f"{col}_{v}"] = df[col].apply(lambda x: 1 if str(x).strip() == v else 0)
process_dict = {'Spin coating': 0, 'Blade Coating': 1, 'Slot-Die Coating': 2,
'Evaporation': 3, 'Inkjet printing': 4, 'VASP': 5}
df['Preparation process'] = df['Preparation process'].map(process_dict).fillna(-1)
numeric_cols = ['Voc', 'Jsc', 'FF', 'Area', 'Substrate function', 'perovskite Eg',
                        'ETL_1 CB', 'ETL_1 VB', 'ETL_2 CB', 'ETL_2 VB',
                        'HTL_1 CB', 'HTL_1 VB', 'HTL_2 CB', 'HTL_2 VB', 'electrode
function']
for col in numeric_cols:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
        df[col] = df[col].fillna(df[col].median())
df = df.drop(columns=['additive_1', 'additive_2', 'Surface passivator'])
X = df.drop(columns=['Efficiency', 'Voc', 'Jsc', 'FF'])
y_pce = df['Efficiency']
y_voc_raw = df['Voc']
VOC_LOW  = float(y_voc_raw.quantile(0.25))   # ≈ 1.020V
VOC_HIGH = float(y_voc_raw.quantile(0.75))   # ≈ 1.140V
def voc_to_class(v):
    if v < VOC_LOW:  return 0
    elif v <= VOC_HIGH: return 1
    else:                return 2   y_voc = y_voc_raw.apply(voc_to_class)
y_jsc = df['Jsc']y_ff = df['FF']train_idx = resample(range(len(df)),
n_samples=int(0.8*len(df)), random_state=68)test_idx = resample(range(len(df)),
n_samples=int(0.2*len(df)), random_state=76)
X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
distribution = {
    0: {0: 32, 1: 44, 2: 97, 3: 7,  4: 17},
    1: {0: 4,  1: 9,  2: 23, 3: 4,  4: 5 },
    2: {0: 8,  1: 4,  2: 22, 3: 2,  4: 1 },
    3: {0: 4,  1: 2,  2: 7,  3: 1,  4: 0 },
    4: {0: 5,  1: 3,  2: 2,  3: 1,  4: 11},
    5: {0: 7,  1: 5,  2: 6,  3: 2,  4: 2 },
}
_all_counts = [v for p in distribution.values() for v in p.values() if v > 0]
_median = float(np.median(_all_counts))
MAX_W = 3.0
def get_area_bin(area):
    if area < 1:       return 0
    elif area < 15:    return 1
    elif area < 60:    return 2
    elif area <= 100: return 3
    else:              return 4
def compute_weights(X_sub):
```

```
        w = []
        for i in range(len(X_sub)):
            proc = int(X_sub.iloc[i]["Preparation process"])
            abin = get_area_bin(float(X_sub.iloc[i]["Area"]))
            cnt  = distribution.get(proc, {}).get(abin, 0)
            w.append(min(_median / cnt, MAX_W) if cnt > 0 else 1.0)
        return np.array(w)
def add_highval_weights(w_base, y_series, boost=2.5):
        q75 = float(y_series.quantile(0.75))
        high_mask = (y_series.values > q75)
        w = w_base.copy()
        w[high_mask] *= boost
        return w
w_base = compute_weights(X_train)
w_pce = add_highval_weights(w_base, y_pce.iloc[train_idx], boost=2.0)
w_jsc = add_highval_weights(w_base, y_jsc.iloc[train_idx], boost=3.0)
w_ff  = add_highval_weights(w_base, y_ff.iloc[train_idx],  boost=2.5)
w_voc = None
param_dist = {
        'n_estimators':      [100, 200, 300, 400, 500],
        'max_depth':         [3, 4, 5, 6, 7],
        'learning_rate':     [0.01, 0.05, 0.08, 0.1, 0.15],
        'subsample':         [0.6, 0.7, 0.8, 0.9],
        'colsample_bytree': [0.6, 0.7, 0.8, 0.9],
        'reg_alpha':         [0.0, 0.1, 0.5, 1.0],
        'reg_lambda':        [0.5, 1.0, 2.0, 3.0],
        'min_child_weight': [1, 3, 5, 7],
        'gamma':             [0.0, 0.1, 0.2, 0.5],
}
param_dist_reg = {
        'n_estimators':      [200, 300, 400, 500, 600],
        'max_depth':         [3, 4, 5],
        'learning_rate':     [0.01, 0.03, 0.05, 0.08],
        'subsample':         [0.5, 0.6, 0.7, 0.8],
        'colsample_bytree': [0.5, 0.6, 0.7, 0.8],
        'reg_alpha':         [0.5, 1.0, 2.0, 5.0],
        'reg_lambda':        [2.0, 3.0, 5.0, 8.0],
        'min_child_weight': [5, 7, 10, 15],
        'gamma':             [0.1, 0.3, 0.5, 1.0],
}
def tune_model(estimator, X_tr, y_tr, sw=None, n_iter=80, pd=None):
        cv = KFold(n_splits=5, shuffle=True, random_state=42)
        fit_kw = {'sample_weight': sw} if sw is not None else {}
        _pd = pd if pd is not None else param_dist
```

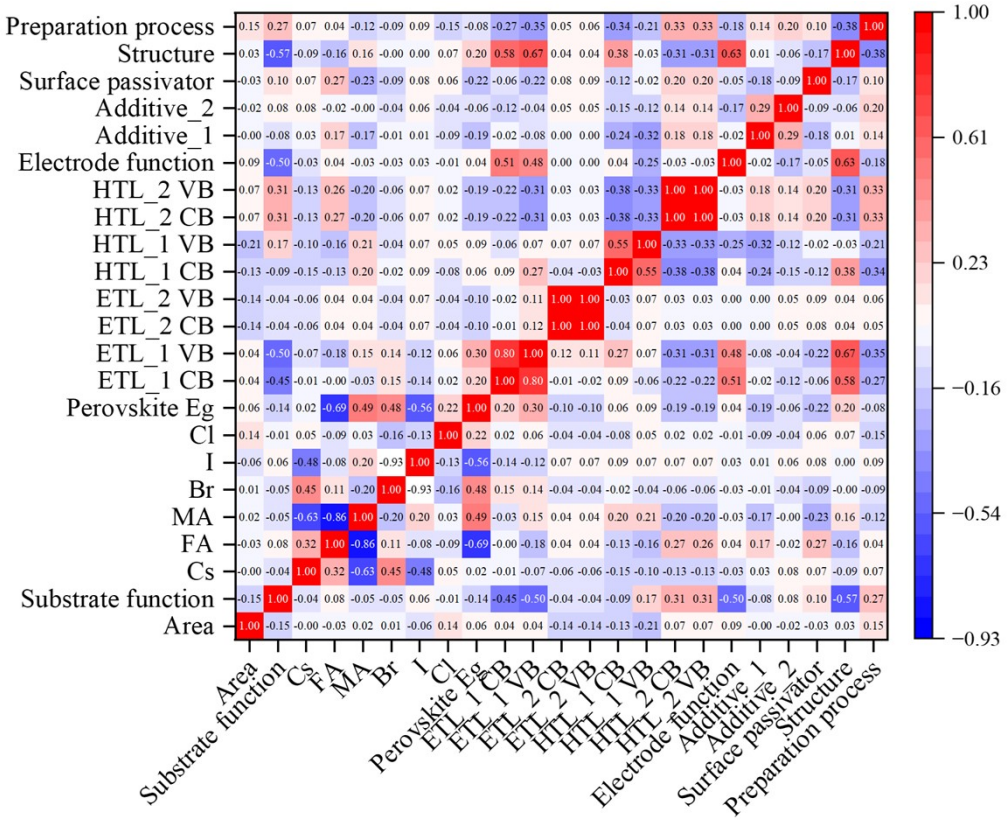
```
        search = RandomizedSearchCV(
            estimator, _pd, n_iter=n_iter, scoring='r2',
            cv=cv, random_state=42, n_jobs=-1, refit=True, verbose=0
        )
        search.fit(X_tr, y_tr, **fit_kw)
        return search.best_estimator_, search.best_params_
def tune_classifier(X_tr, y_tr, sw=None, n_iter=80):
        param_dist_cls = {**param_dist,
                                'use_label_encoder': [False],
                                'eval_metric': ['mlogloss']}
        cv = KFold(n_splits=5, shuffle=True, random_state=42)
        fit_kw = {'sample_weight': sw} if sw is not None else {}
        search = RandomizedSearchCV(
            xgb.XGBClassifier(random_state=42, n_jobs=-1, verbosity=0),
            param_dist_cls, n_iter=n_iter, scoring='accuracy',
            cv=cv, random_state=42, n_jobs=-1, refit=True, verbose=0
        )
        search.fit(X_tr, y_tr, **fit_kw)

        return search.best_estimator_, search.best_params_
def fit_isotonic(model, X_tr, y_tr):
        y_raw = model.predict(X_tr)
        iso = IsotonicRegression(out_of_bounds='clip')
        iso.fit(y_raw, y_tr)
        return iso
def predict_with_iso(model, iso, X):
        y_raw = model.predict(X)
        return iso.predict(y_raw)
iso_pce = fit_isotonic(pce_model, X_train, y_pce.iloc[train_idx])
iso_jsc = fit_isotonic(jsc_model, X_train, y_jsc.iloc[train_idx])
iso_ff  = fit_isotonic(ff_model,  X_train, y_ff.iloc[train_idx])
def get_metrics(y_true, y_pred):
        r2   = r2_score(y_true, y_pred)
        rmse = np.sqrt(mean_squared_error(y_true, y_pred))
        mape = mean_absolute_percentage_error(y_true, y_pred) * 100
        return r2, rmse, mape
y_pce_tr_pred = predict_with_iso(pce_model, iso_pce, X_train)
y_pce_te_pred = predict_with_iso(pce_model, iso_pce, X_test)
y_voc_tr_pred = voc_model.predict(X_train); y_voc_te_pred =
voc_model.predict(X_test)
y_voc_tr_proba = voc_model.predict_proba(X_train); y_voc_te_proba =
voc_model.predict_proba(X_test)
y_jsc_tr_pred = predict_with_iso(jsc_model, iso_jsc, X_train)
y_jsc_te_pred = predict_with_iso(jsc_model, iso_jsc, X_test)
```

```
y_ff_tr_pred = predict_with_iso(ff_model, iso_ff, X_train)
y_ff_te_pred = predict_with_iso(ff_model, iso_ff, X_test)
pce_tr_r2, pce_tr_rmse, pce_tr_mape = get_metrics(y_pce.iloc[train_idx],
y_pce_tr_pred)
pce_te_r2, pce_te_rmse, pce_te_mape = get_metrics(y_pce.iloc[test_idx],
y_pce_te_pred)
voc_tr_acc = accuracy_score(y_voc.iloc[train_idx], y_voc_tr_pred) * 100
voc_te_acc = accuracy_score(y_voc.iloc[test_idx], y_voc_te_pred) * 100
jsc_tr_r2, jsc_tr_rmse, jsc_tr_mape = get_metrics(y_jsc.iloc[train_idx], y_jsc_tr_pred)
jsc_te_r2, jsc_te_rmse, jsc_te_mape = get_metrics(y_jsc.iloc[test_idx],
y_jsc_te_pred)
ff_tr_r2, ff_tr_rmse, ff_tr_mape = get_metrics(y_ff.iloc[train_idx], y_ff_tr_pred)
ff_te_r2, ff_te_rmse, ff_te_mape = get_metrics(y_ff.iloc[test_idx],
y_ff_te_pred)

shap analysis
shap.initjs()
explainer_pce = shap.TreeExplainer(pce_model); shap_values_pce =
explainer_pce.shap_values(X_test)
explainer_jsc = shap.TreeExplainer(jsc_model); shap_values_jsc =
explainer_jsc.shap_values(X_test)
explainer_ff = shap.TreeExplainer(ff_model); shap_values_ff =
explainer_ff.shap_values(X_test)
explainer_voc = shap.TreeExplainer(voc_model)
shap_values_voc_all = explainer_voc.shap_values(X_test)
shap_values_voc = shap_values_voc_all[:, :, 2]
def plot_shap_summary(shap_values, features, title, filename):
    plt.figure(figsize=(12, 8))
    feature_importance = np.abs(shap_values).mean(axis=0)
    top_indices = np.argsort(feature_importance)[-15:][::-1]
    shap.summary_plot(shap_values[:, top_indices], features.iloc[:, top_indices],
show=False)
    plt.title(f'{title} - SHAP', fontsize=20, fontname='Times New Roman')
    plt.tight_layout()
    plt.savefig(f'shap_summary_{filename}.png', dpi=300, bbox_inches='tight')
    plt.show()
plot_shap_summary(shap_values_pce, X_test, 'PCE', 'pce')
plot_shap_summary(shap_values_voc, X_test, 'Voc', 'voc')
plot_shap_summary(shap_values_jsc, X_test, 'Jsc', 'jsc')
plot_shap_summary(shap_values_ff, X_test, 'FF', 'ff')

PCE Prediction
manual_composition = {
    'Cs': 0.1,
```

```
        'FA': 0.888888889,
        'MA': 0.011111111,
        'Br': 0.059055118,
        'I':    2.940944882,
        'Cl': 0.0,
}
manual_process_id = 5
for comp, value in manual_composition.items():
        print(f"    {comp}: {value}")
cation_sum = manual_composition['Cs'] + manual_composition['FA'] +
manual_composition['MA']
if cation_sum > 0:
        manual_composition['Cs'] = manual_composition['Cs'] / cation_sum
        manual_composition['FA'] = manual_composition['FA'] / cation_sum
        manual_composition['MA'] = manual_composition['MA'] / cation_sum
halogen_sum = manual_composition['Br'] + manual_composition['I'] +
manual_composition['Cl']
if halogen_sum > 0:
        scale = 3 / halogen_sum
        manual_composition['Br'] = manual_composition['Br'] * scale
        manual_composition['I']    = manual_composition['I'] * scale
        manual_composition['Cl'] = manual_composition['Cl'] * scale
for comp, value in manual_composition.items():
        print(f"    {comp}: {value:.4f}")
process_names = {
        0: 'Spin coating',
        1: 'Blade Coating',
        2: 'Slot-Die Coating',
        3: 'Evaporation',
        4: 'Inkjet printing',
        5: 'VASP'
}
fixed_process = manual_process_id
other_fixed_params = {
        'Preparation process': fixed_process,
        'Substrate function': 4.5,
        'electrode function': 5.28,
        'Structure': 1,
        'ETL_1 CB': 4.5,
        'ETL_1 VB': 7.5,
        'ETL_2 CB': 4.5,
        'ETL_2 VB': 7.5,
        'HTL_1 CB': 2.2,
        'HTL_1 VB': 5.22,
```

```
    'HTL_2 CB': 0,
    'HTL_2 VB': 0,
}
passivator_features = [col for col in X.columns if col.startswith('Passivator_')]
for pas in passivator_features:
    if pas == 'Passivator_Organic_ammonium_salts':
        other_fixed_params[pas] = 1
    else:
        other_fixed_params[pas] = 0
additive1_features = [col for col in X.columns if col.startswith('Additive1_')]
additive2_features = [col for col in X.columns if col.startswith('Additive2_')]
for add in additive1_features:
    other_fixed_params[add] = 1 if add == 'Additive1_others' else 0
for add in additive2_features:
    other_fixed_params[add] = 1 if add == 'Additive2_none' else 0
for feature in X.columns:
    if feature not in manual_composition and feature not in other_fixed_params and
feature != 'Area':
        if pd.api.types.is_numeric_dtype(X[feature]):
            other_fixed_params[feature]=high_efficiency_samples[feature].median()
        else:
            other_fixed_params[feature]                                    =
high_efficiency_samples[feature].value_counts().idxmax()
start_area, end_area, step_area = 0.03, 100.03, 0.5
area_range = np.arange(start_area, end_area + step_area, step_area)
area_results = []
for i, area in enumerate(tqdm(area_range, desc=f" {manual_process_id} ")):
    param_dict = {'Area': area}
    param_dict.update(manual_composition)
    param_dict.update(other_fixed_params)
    param_dict['Area_inv'] = 1 / (area + 0.001)
    param_dict['log_Area'] = np.log(area + 0.001)
    input_data = pd.DataFrame([param_dict])
    for col in X.columns:
        if col not in input_data.columns:
            input_data[col] = other_fixed_params.get(col, 0)
    input_data = input_data[X.columns]
    try:
        efficiency = predict_with_iso(pce_model, iso_pce, input_data)[0]
        adjusted_efficiency = apply_process_specific_adjustment(efficiency, area,
manual_process_id)
        area_results.append((adjusted_efficiency, area))
    except Exception as e:
        continue
```

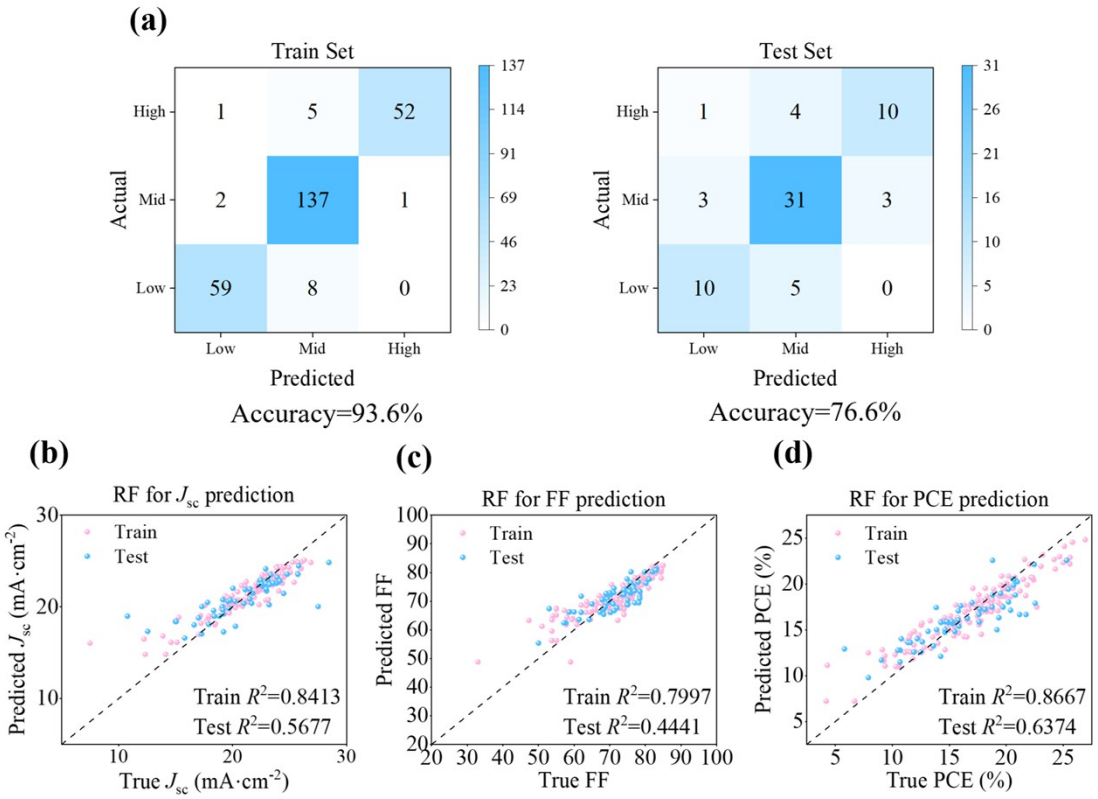
414




415

416 **Fig. S1** The heatmap representing the strength of correlation between input feature

417 quantities.




418

419 **Fig. S2** The RF model was used to train $V_{OC}$, $J_{SC}$, FF and PCE. (a) $V_{OC}$ confusion matrix.

420 In the other figures, red represents the training set and blue represents the testing set:

421 (b) $J_{SC}$, (c) FF, (d) PCE.

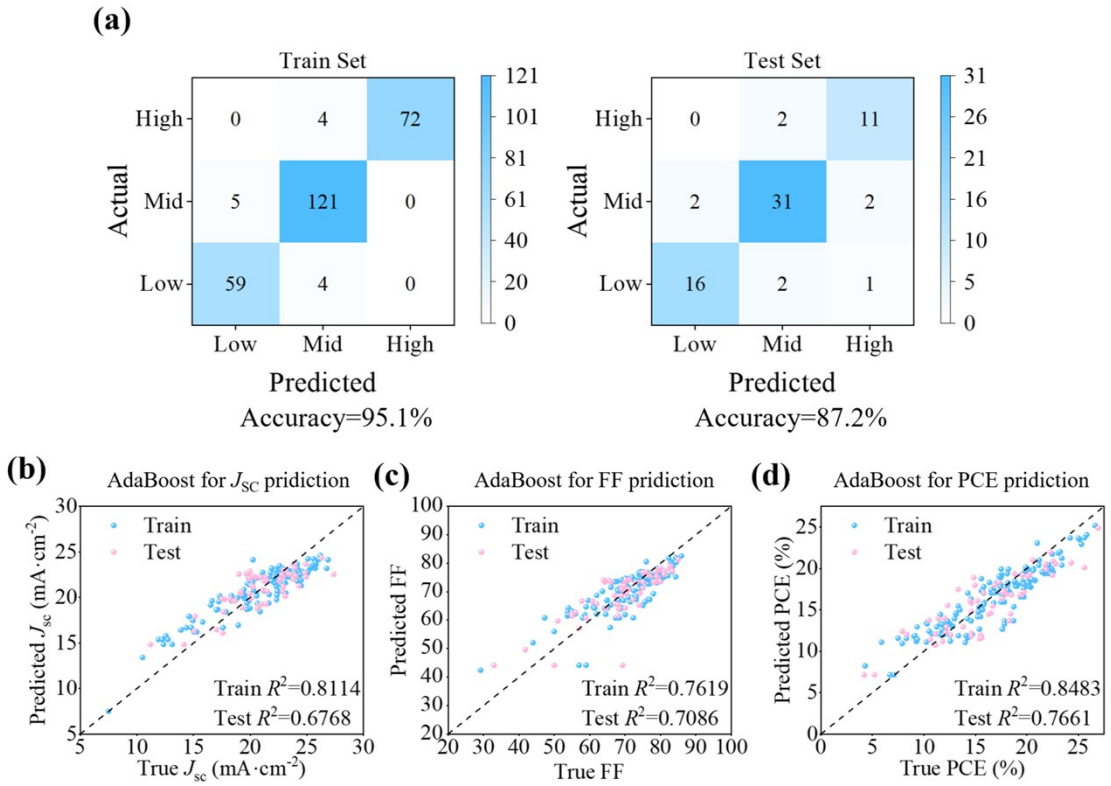



**Fig. S3** The AdaBoost model was used to train $V_{OC}$, $J_{SC}$, FF and PCE. (a) $V_{OC}$ confusion matrix. In the other figures, red represents the training set and blue represents the testing set: (b) $J_{SC}$, (c) FF, (d) PCE.

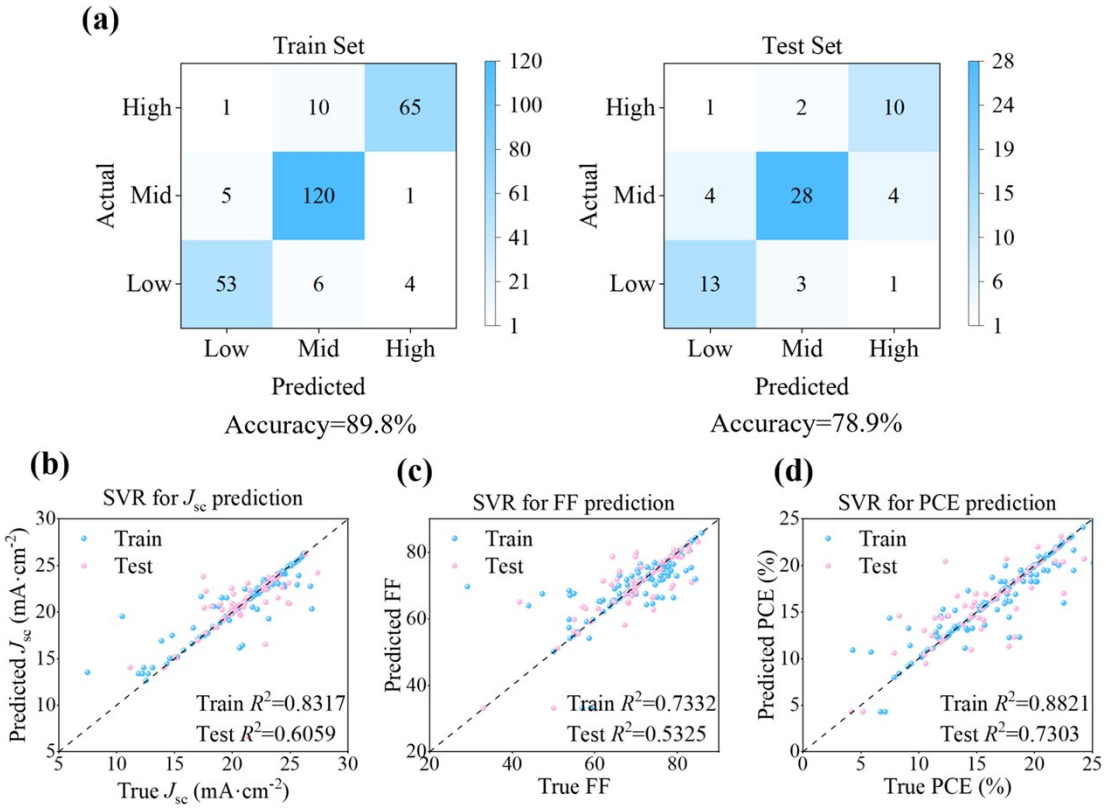



**Fig. S4** The SVR model was used to train $V_{OC}$, $J_{SC}$, FF and PCE. (a) $V_{OC}$ confusion matrix. In the other figures, red represents the training set and blue represents the testing set: (b) $J_{SC}$, (c) FF, (d) PCE.

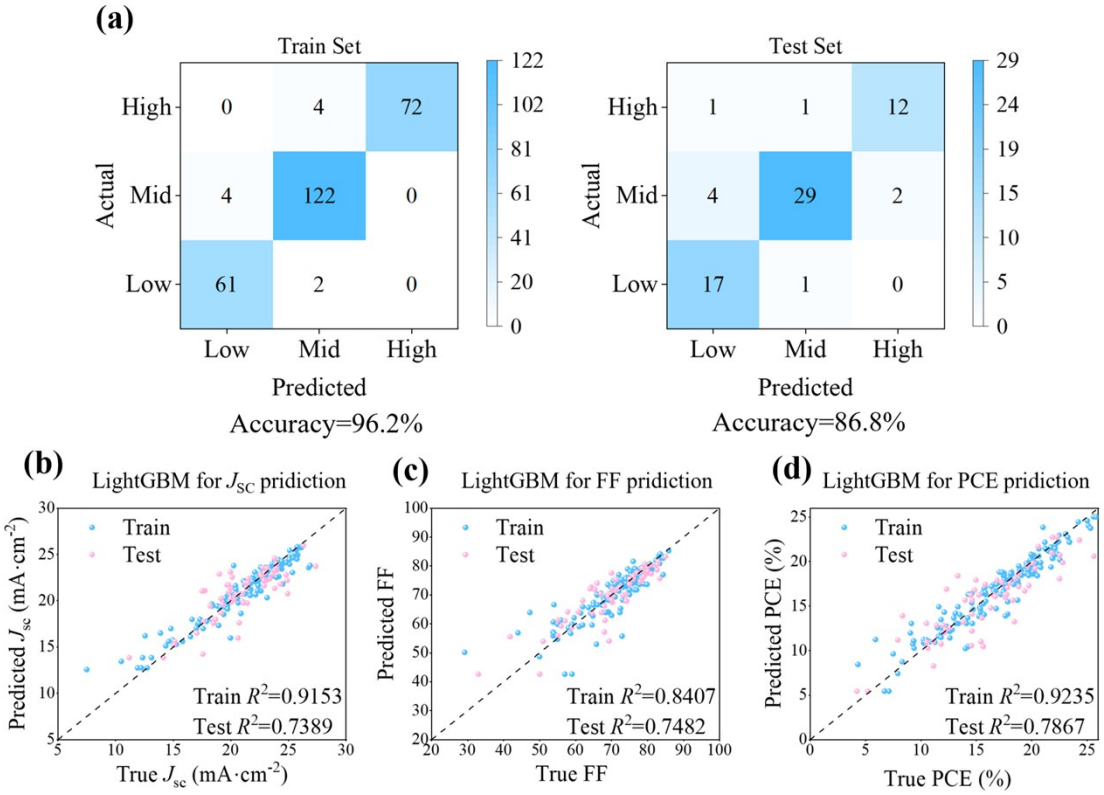



**Fig. S5** The LightGBM model was used to train $V_{OC}$, $J_{SC}$, FF and PCE. (a) $V_{OC}$ confusion matrix. In the other figures, red represents the training set and blue represents the testing set: (b) $J_{SC}$, (c) FF, (d) PCE.

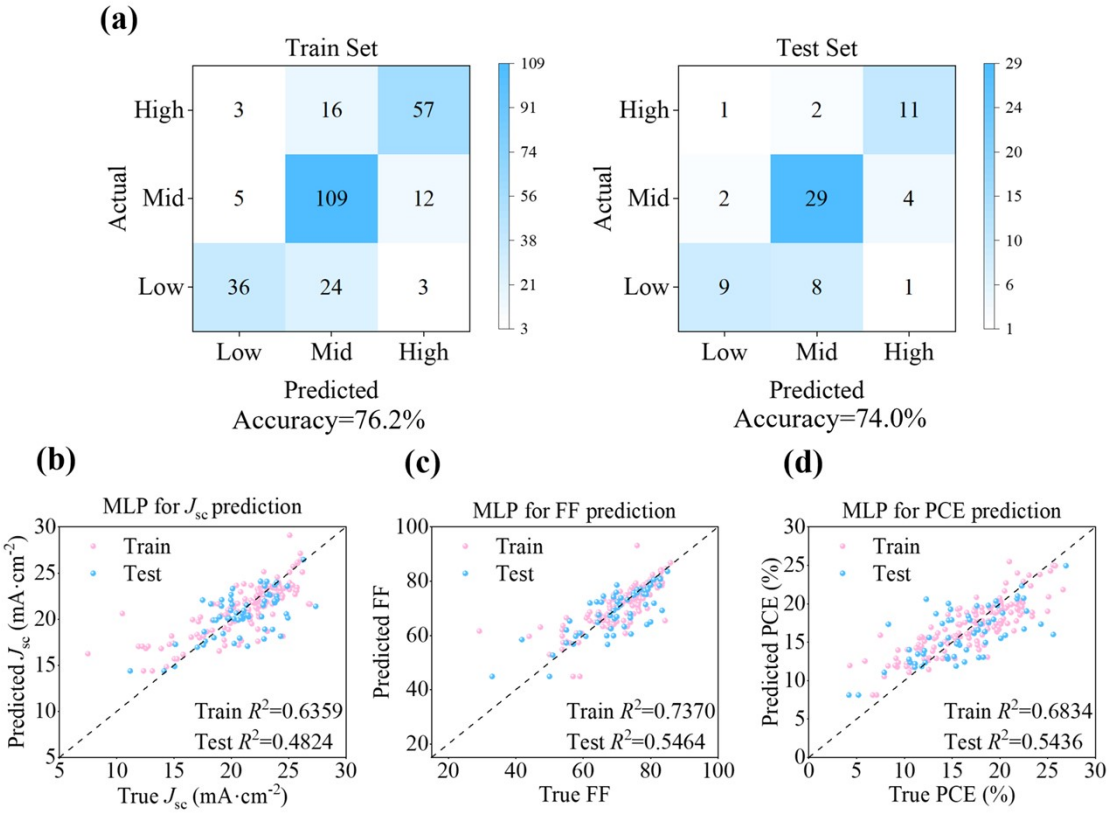
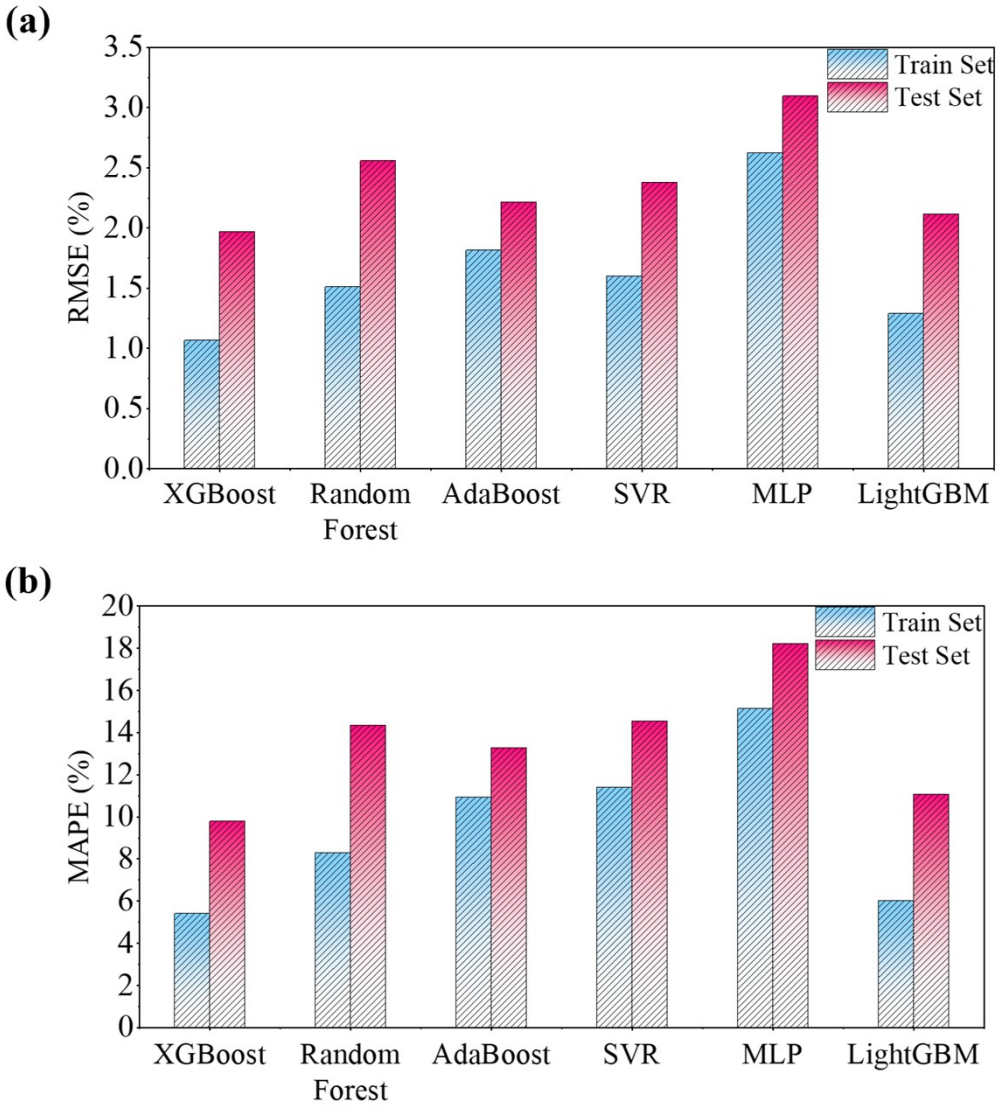



**Fig. S6** The MLP model was used to train $V_{OC}$, $J_{SC}$, FF and PCE. (a) $V_{OC}$ confusion matrix. In the other figures, red represents the training set and blue represents the testing set: (b) $J_{SC}$, (c) FF, (d) PCE.




441

442 **Fig. S7** Performance of 6 ML models: **(a)** RMSE, **(b)** MAPE.

443

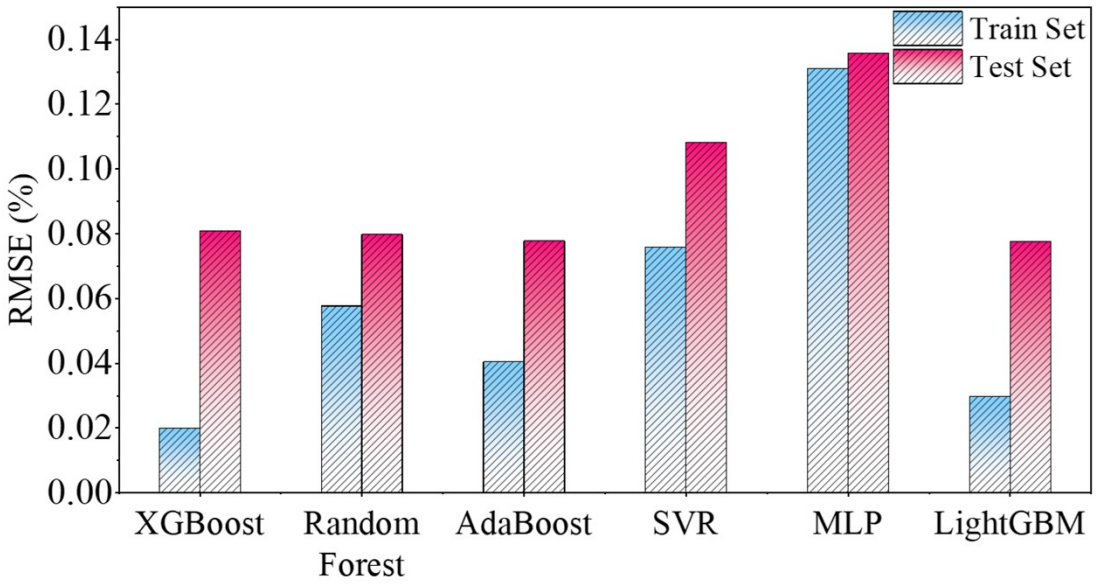



**Fig. S8** RMSE of six models trained on *V*oc.

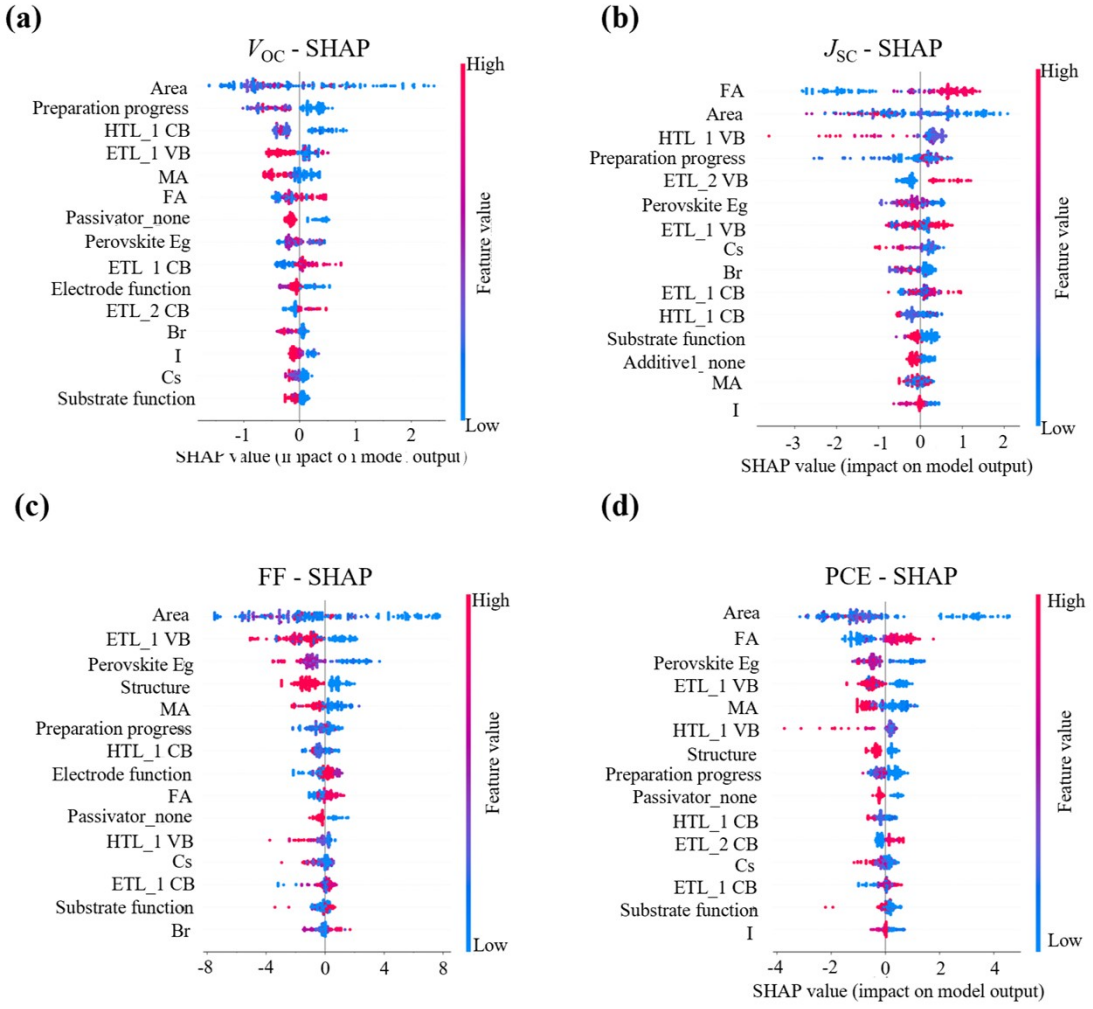



**Fig. S9** The SHAP analysis diagram of characteristic parameters $V_{OC}$, $J_{SC}$, FF, PCE affecting perovskite solar cells based on XGBoost model: **(a)** $V_{OC}$, **(b)** $J_{SC}$, **(c)** FF, and **(d)** PCE.

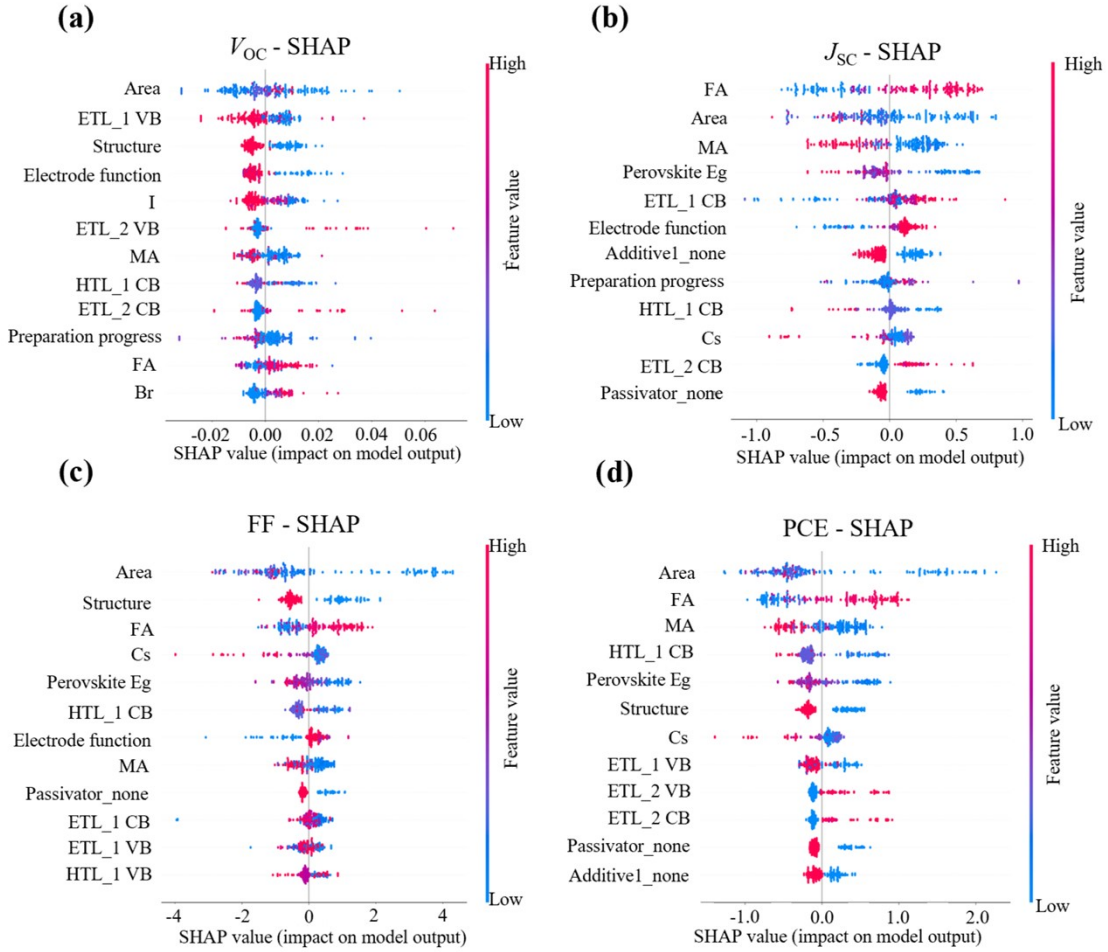



**Fig. S10** The SHAP analysis diagram of characteristic parameters $V_{OC}$, $J_{SC}$, FF, PCE affecting perovskite solar cells based on RF model: **(a)** $V_{OC}$, **(b)** $J_{SC}$, **(c)** FF, and **(d)** PCE.

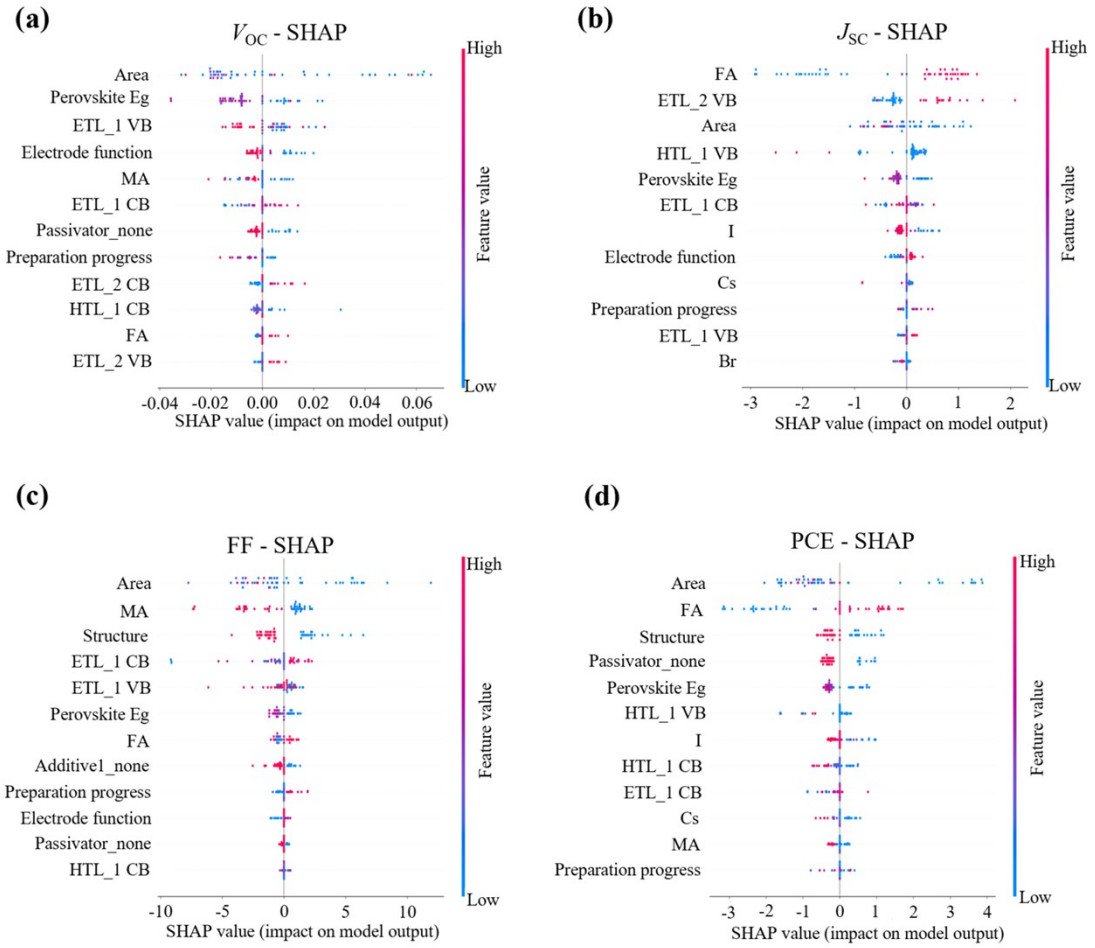



**Fig. S11** The SHAP analysis diagram of characteristic parameters $V_{OC}$, $J_{SC}$, FF, PCE affecting perovskite solar cells based on AdaBoost model: **(a)** $V_{OC}$, **(b)** $J_{SC}$, **(c)** FF, and **(d)** PCE.

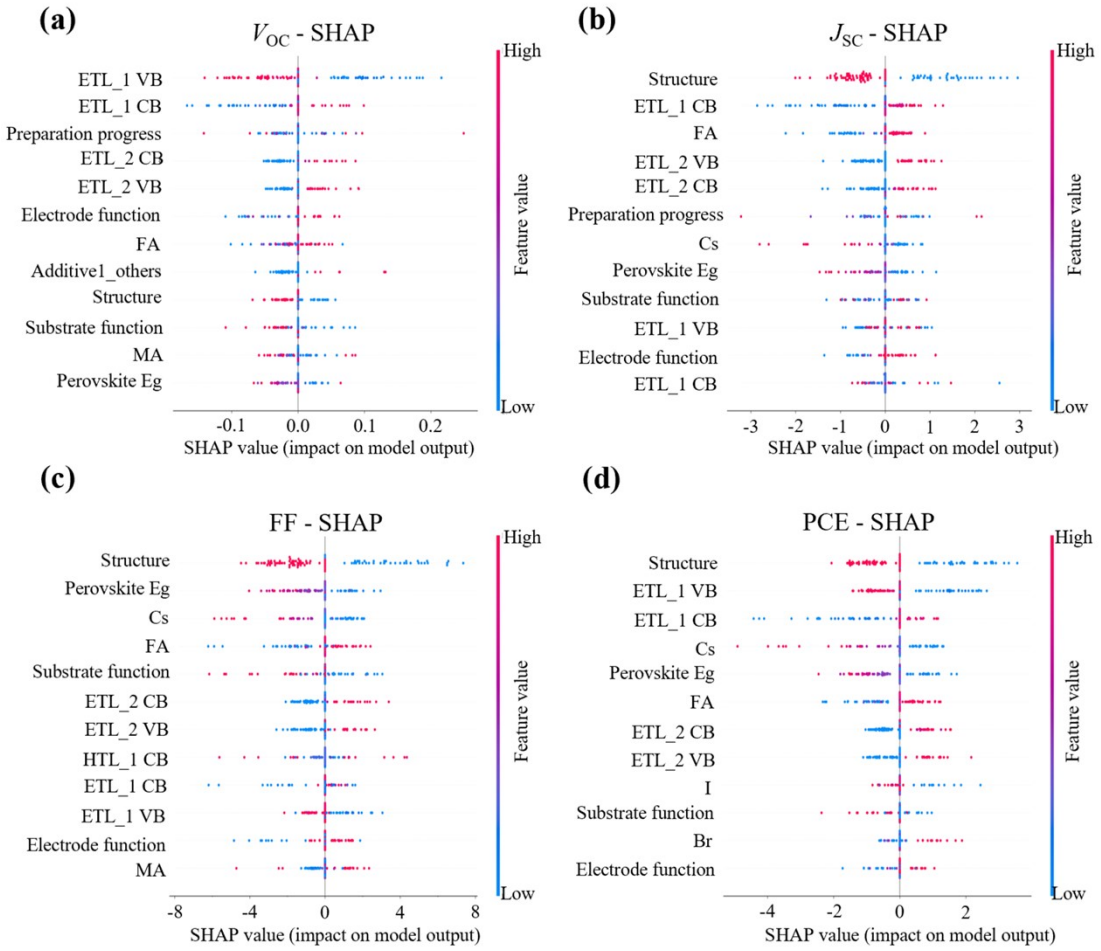



**Fig. S12** The SHAP analysis diagram of characteristic parameters $V_{OC}$, $J_{SC}$, FF, PCE affecting perovskite solar cells based on SVR model: **(a)** $V_{OC}$, **(b)** $J_{SC}$, **(c)** FF, and **(d)** PCE.

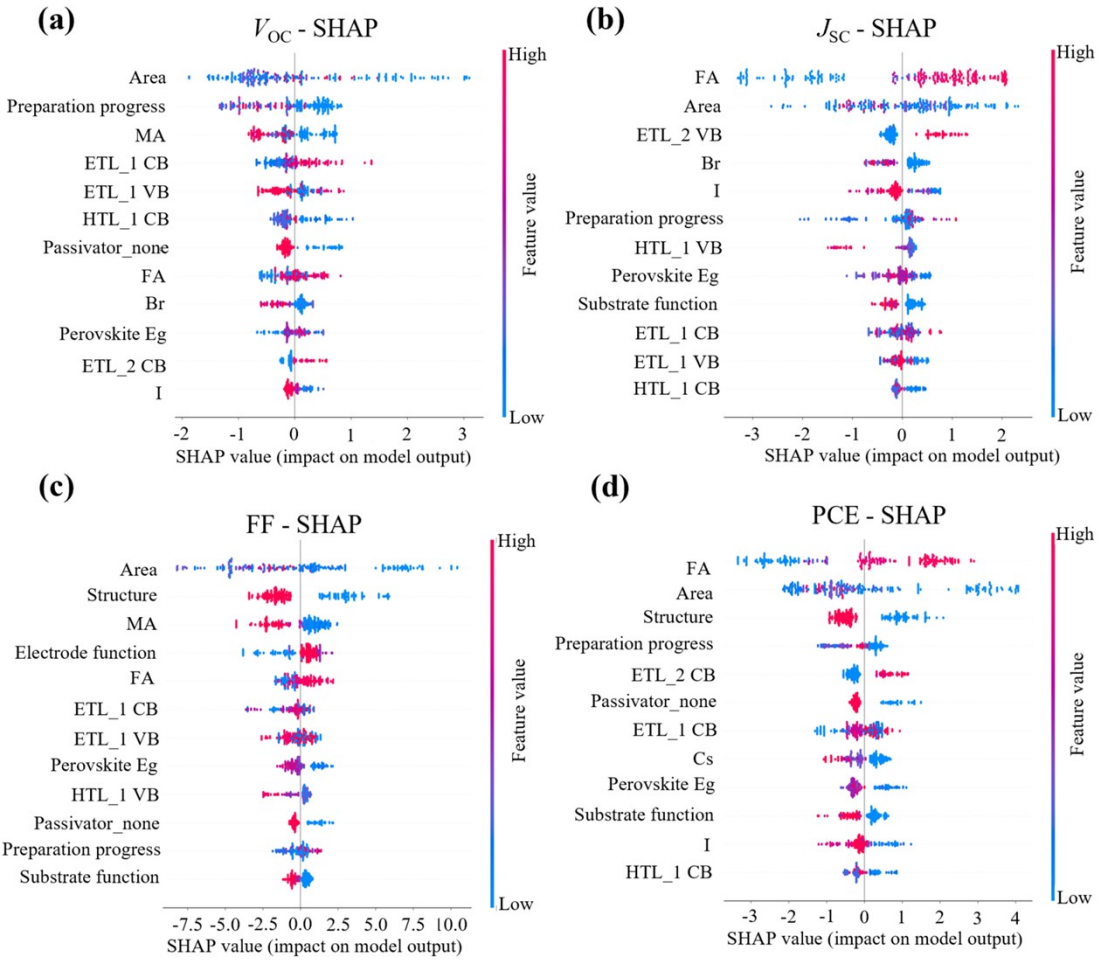



**Fig. S13** The SHAP analysis diagram of characteristic parameters $V_{OC}$, $J_{SC}$, FF, PCE affecting perovskite solar cells based on LightGBM model: **(a)** $V_{OC}$, **(b)** $J_{SC}$, **(c)** FF, and **(d)** PCE.

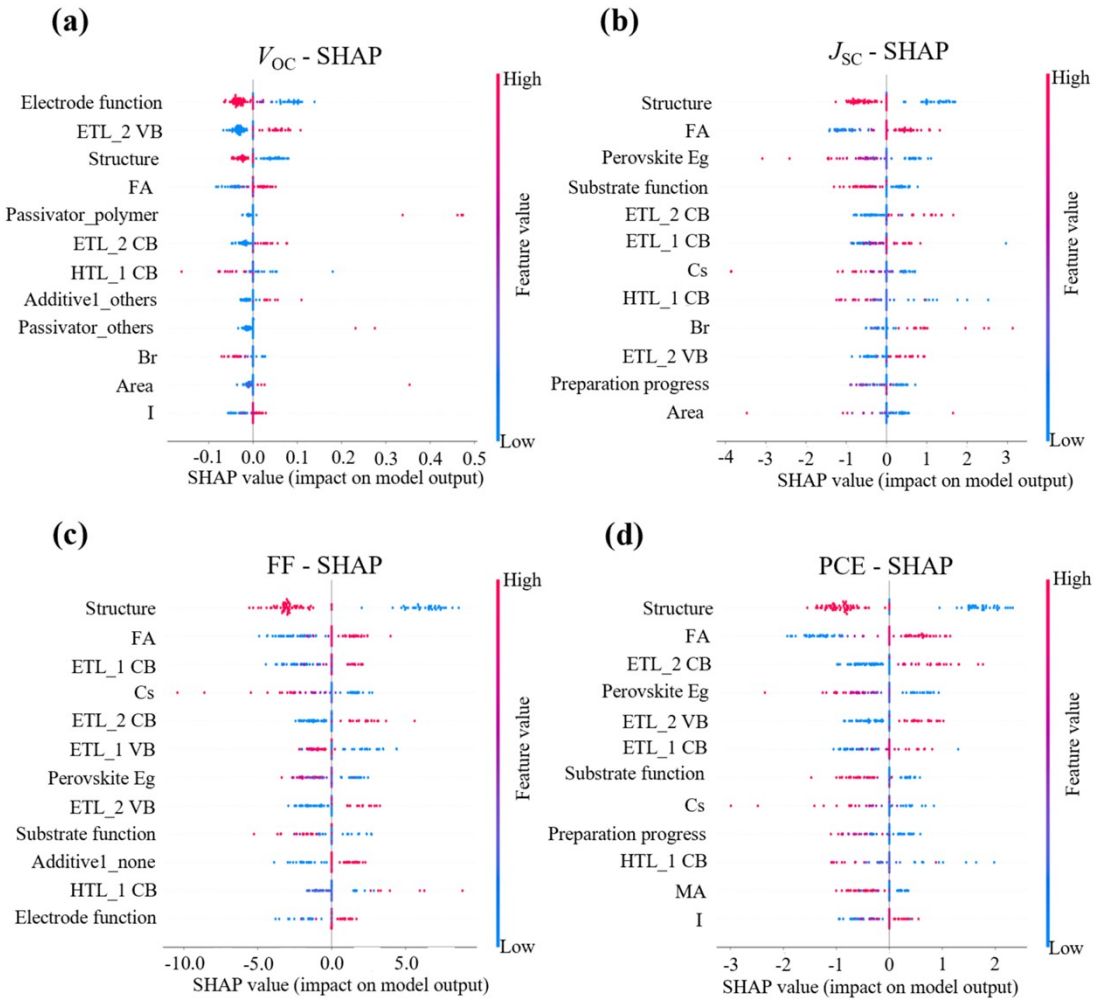



**Fig. S14** The SHAP analysis diagram of characteristic parameters $V_{OC}$, $J_{SC}$, FF, PCE affecting perovskite solar cells based on MLP model: **(a)** $V_{OC}$, **(b)** $J_{SC}$, **(c)** FF, and **(d)** PCE.

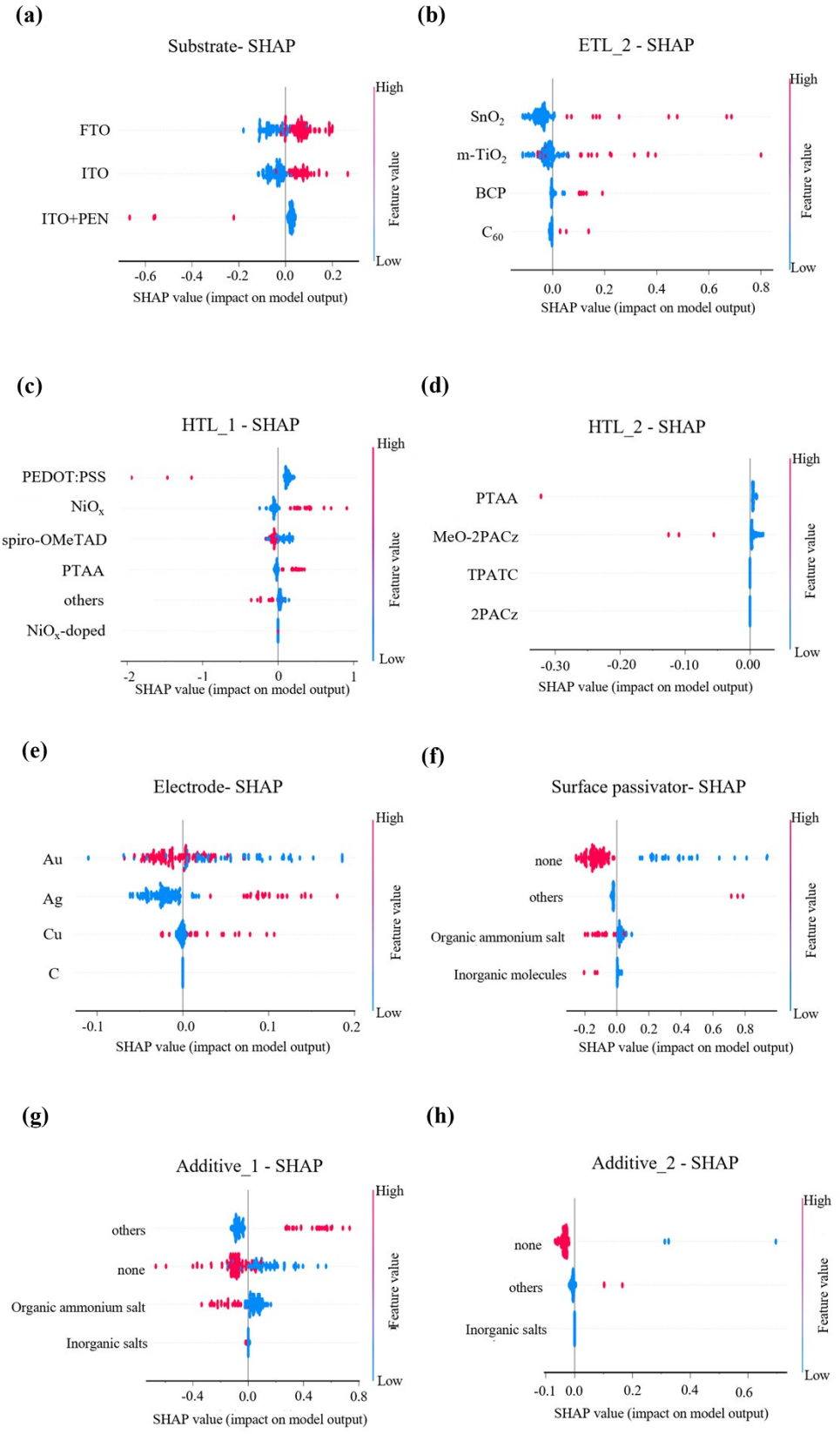



**Fig. S15** SHAP analysis diagram of functional layers in perovskite solar cells based on the optimal XGBoost model: **(a)** Substrate, **(b)** ETL_2, **(c)** HTL_1, **(d)** HTL_2, **(e)** Electrode, **(f)** Surface Passivator, **(g)** Additive_1, **(h)** Additive_2.

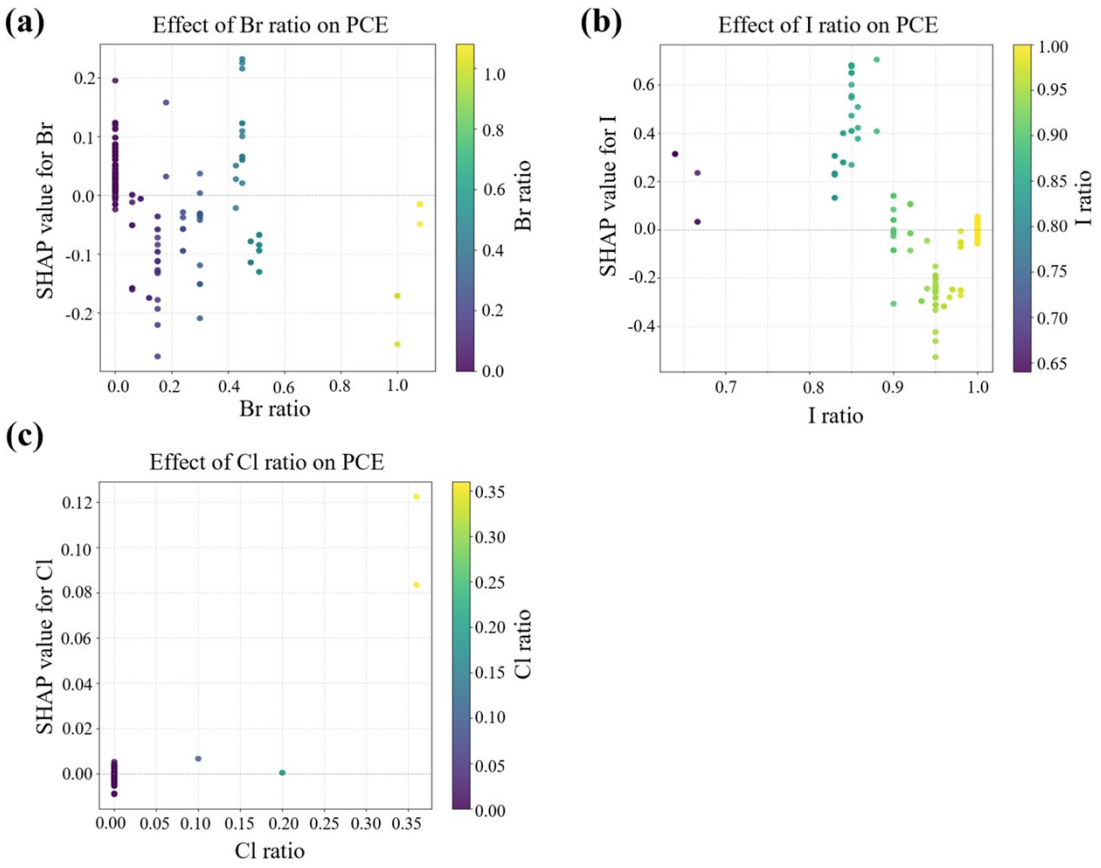



**Fig. S16** SHAP analysis diagram of the influence of different anion ratios on PCE based on the optimal XGBoost model: **(a)** Br anion, **(b)** I anion, and **(c)** Cl anion.

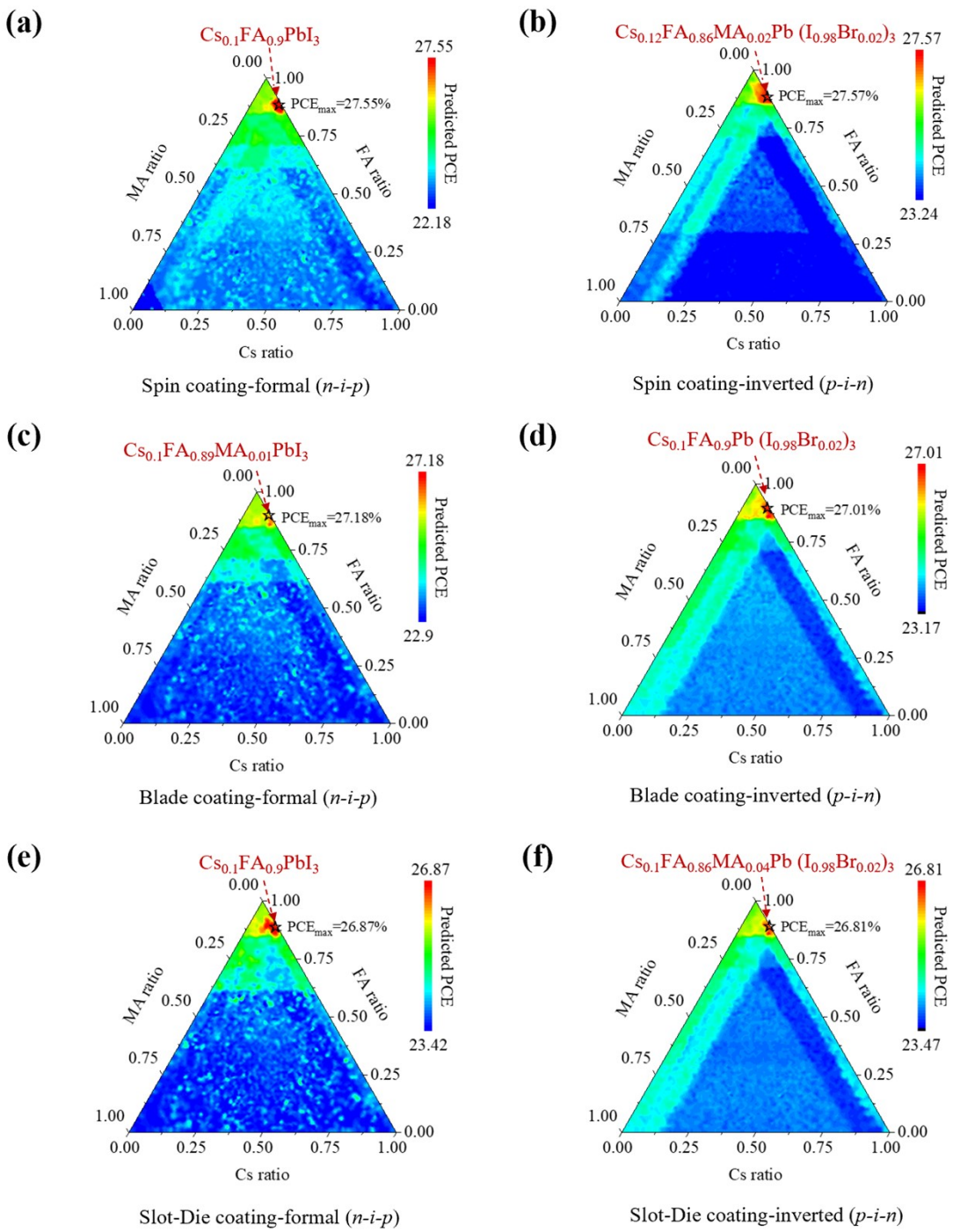



**Fig. S17** The triangular heatmap diagram of the effects of Cs, FA, and MA cations on PCE for three preparation processes of formal and inverted structures under a small area of 0.03 $cm^2$ based on the optimal XGBoost model: **(a)** Spin coating-formal, **(b)** Spin coating-inverted, **(c)** Blade coating-formal, **(d)** Blade coating-inverted, **(e)** Slot-Die coating-formal, and **(f)** Slot-Die coating-inverted.

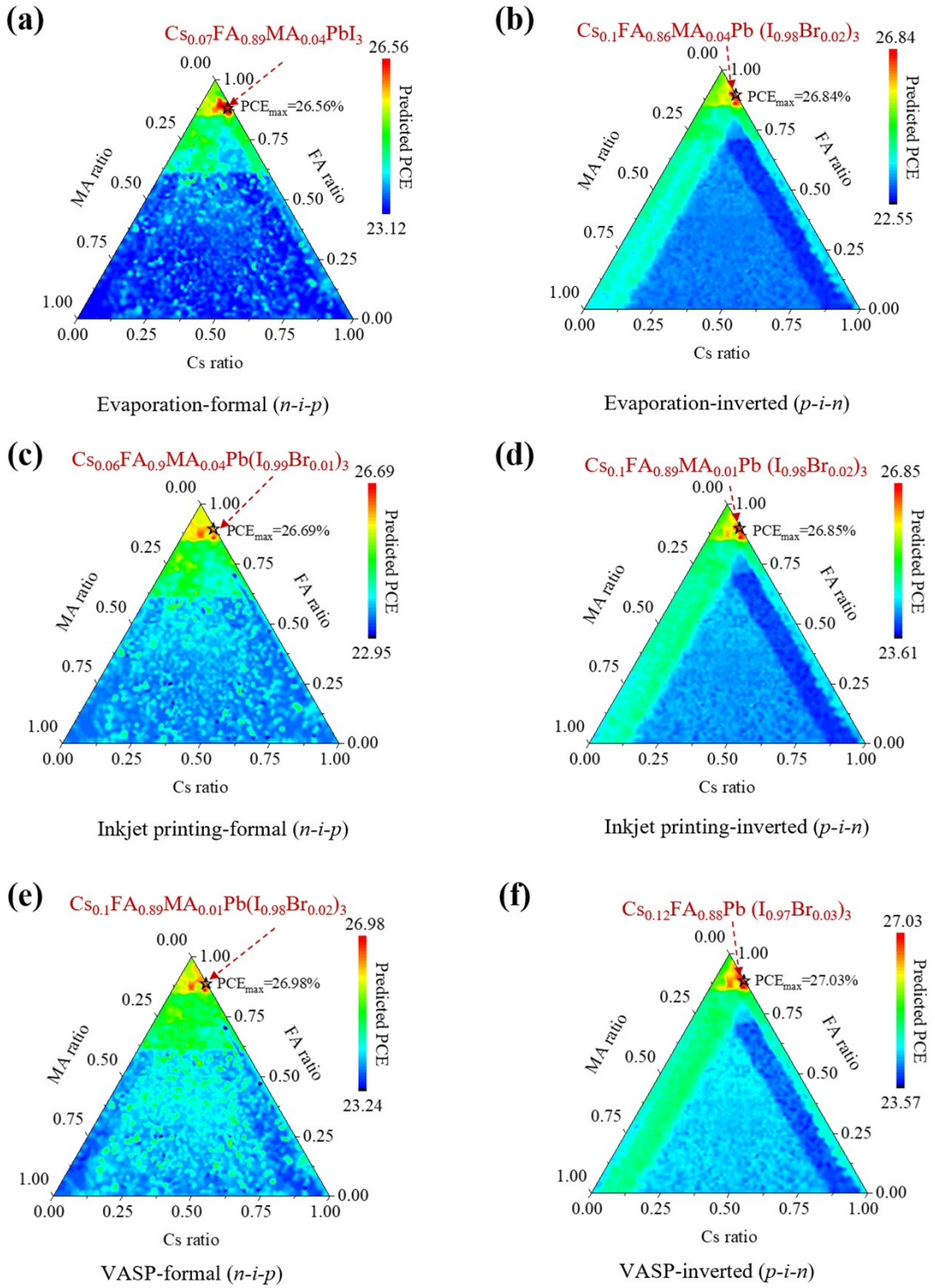



**Fig. S18** The triangular heatmap of the effects of Cs, FA, and MA cations on PCE for three preparation processes of formal structure and inverted structure under a small area of 0.03 $cm^2$ based on the optimal XGBoost model: **(a)** Evaporation-formal, **(b)** Evaporation-inverted, **(c)** Inkjet printing-formal, **(d)** Inkjet printing-inverted, **(e)** VASP-formal, and **(f)** VASP-inverted.

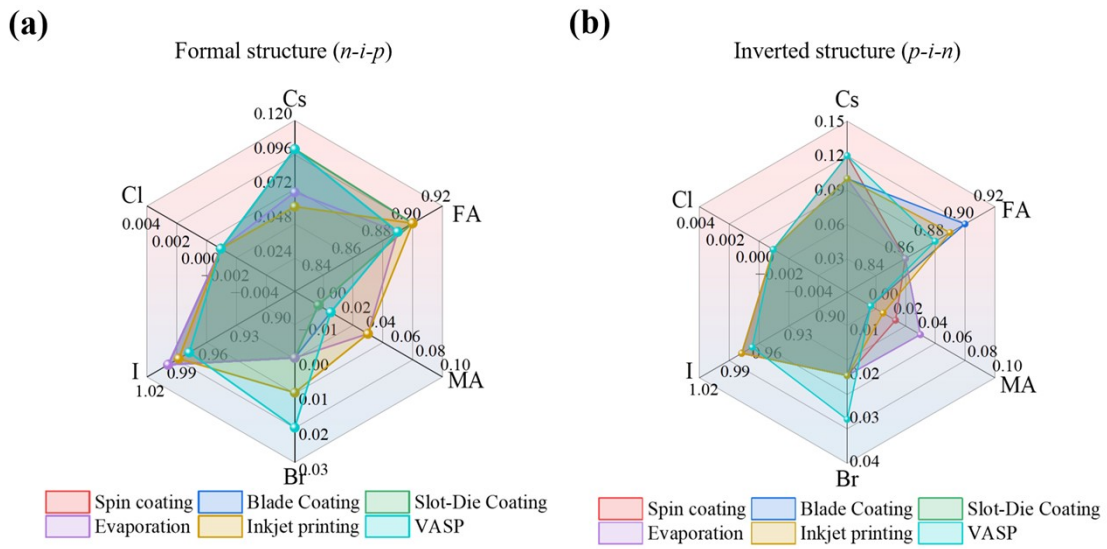



**Fig. S19** Radar spectra of the optimal perovskite components for formal structure and inverted structure prepared by six different processes with a small area of 0.03 $cm^2$: **(a)** Formal structure, **(b)** Inverted structure.

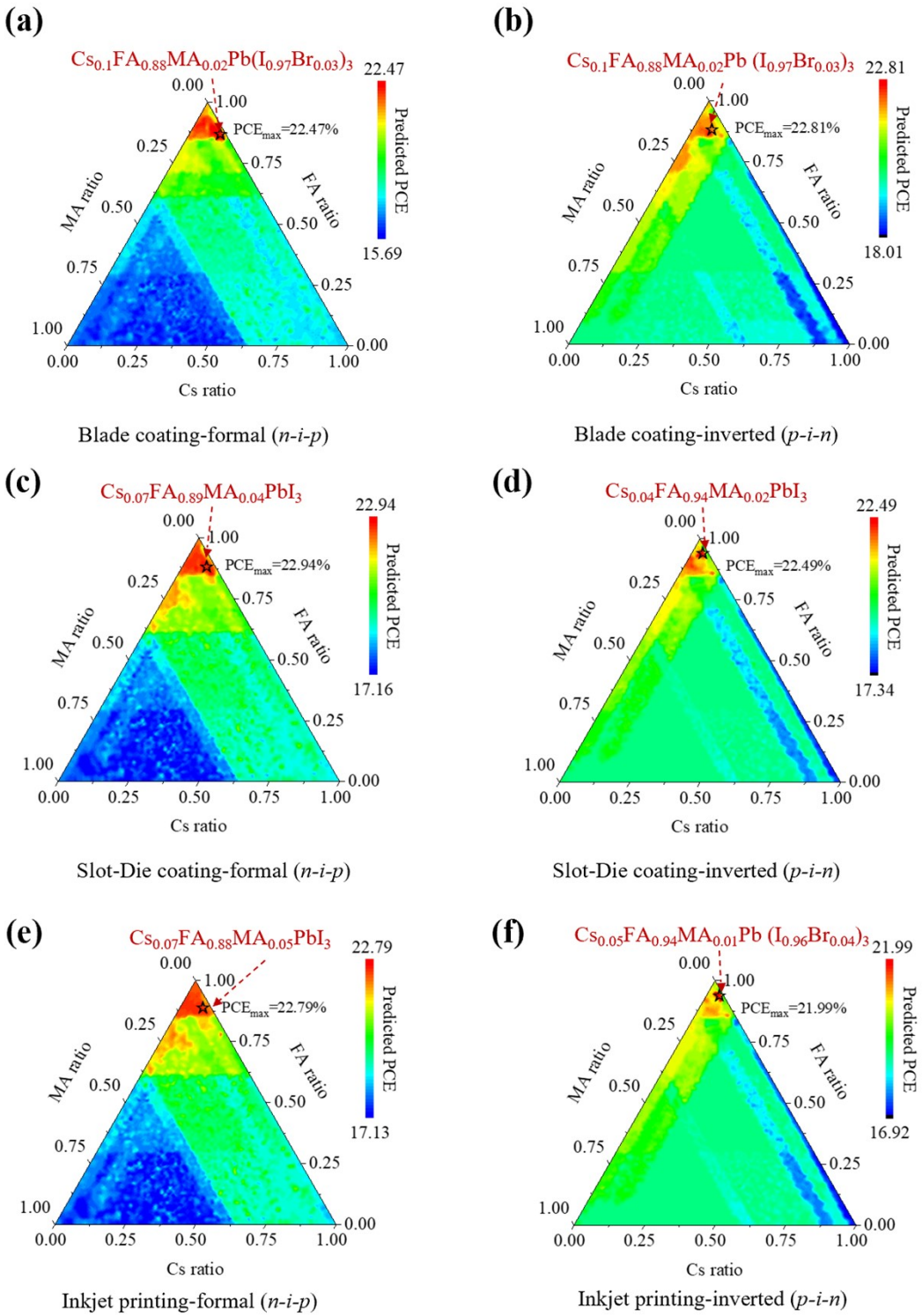



**Fig. S20** The triangular heatmap of the effects of Cs, FA, and MA cations on PCE for three preparation processes of formal and inverted structures under a large area of 10 $cm^2$ based on the optimal XGBoost model: **(a)** Blade coating-formal, **(b)** Blade coating-inverted, **(c)** Slot-Die coating-formal, **(d)** Slot-Die coating-inverted **(e)** Inkjet printing-formal, and **(f)** Inkjet printing-inverted.

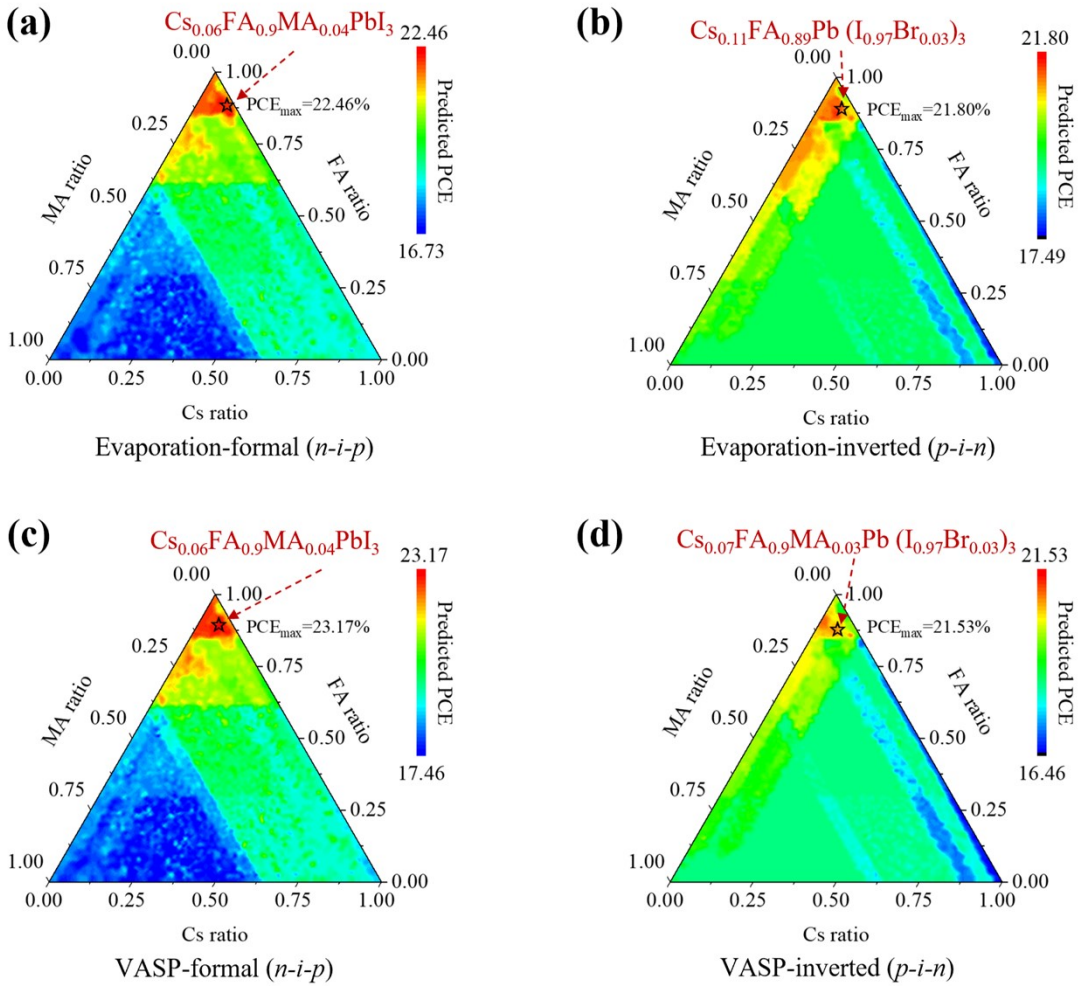



**Fig. S21** The ternary phase diagram of the effects of Cs, FA, and MA cations on PCE for two preparation processes of formal and inverted structures under a large area of 10 $cm^2$ based on the optimal XGBoost model: **(a)** Evaporation-formal, **(b)** Evaporation-inverted, **(c)** VASP-formal, and **(d)** VASP-inverted.

**Table. S1** The 23 input features and corresponding explanations of ML models.

| Factors | Description or alternatives for one-hot encoding |
|---|---|
| ETL_1 VB | VB value of the first ETL layer |
| ETL_1 CB | CB value of the first ETL layer |
| ETL_2 VB | VB value of the second ETL layer |
| ETL_2 CB | CB value of the second ETL layer |
| HTL_1 VB | VB value of the first HTL layer |
| HTL_1 CB | CB value of the first HTL layer |
| HTL_2 VB | VB value of the second HTL layer |
| HTL_2 CB | CB value of the second HTL layer |
| Electrode function | Work functions of different types of electrodes |
| Substrate function | Work functions of different types of substrates |
| Area | The mask area of the J-V test of the device |
| Perovskite Eg | Perovskiteb Eg is given based on empirical formulas |
| MA | The proportion of MA cations to all cations in perovskite |
| FA | The proportion of FA cations to all cations in perovskite |
| Cs | The proportion of Cs cations to all cations in perovskite |
| Br | The proportion of Br anions to all anions in perovskite |
| I | The proportion of I anions to all anions in perovskite |
| Cl | The proportion of Cl anions to all anions in perovskite |
| Additive_1 | Inorganic salts, Organic ammonium salt, others, none |
| Additive_2 | Inorganic salts, Organic ammonium salt, others, none |
| Surface passivator | Organic ammonium salts, polymer, Inorganic molecules, others, none |
| Structure | n-i-p, p-i-n |
| Preparation process | Spin coating, Blade Coating, Slot-Die Coating, Evaporation, Inkjet printing, VASP |

**Table S2.** Data distribution of the dataset across different device areas and preparation processes.

| Preparation process | Sample size | | | | | |
|---|---|---|---|---|---|---|
| | <1 | 1 to 15 | 15 to 60 | 60 to 100 | >100 | total |
| Spin Coating | 32 | 44 | 97 | 7 | 17 | 197 |
| Blade Coating | 4 | 9 | 23 | 4 | 5 | 45 |
| Slot-Die Coating | 8 | 4 | 22 | 2 | 1 | 37 |
| Evaporation | 4 | 2 | 7 | 1 | 0 | 14 |
| Inkjet Printing | 5 | 3 | 2 | 1 | 11 | 22 |
| VASP | 7 | 5 | 6 | 2 | 2 | 22 |
| total | 60 | 67 | 157 | 17 | 36 | 337 |

**Table S3.** Data distribution ratio before and after weighted processing of different preparation processes.

| Preparation process | Spin coating | Blade Coating | Slot-Die Coating | Evaporation | Inkjet printing | VASP |
|---|---|---|---|---|---|---|
| Unweighted processing | 58.5% | 13.4% | 11.0% | 4.2% | 6.5% | 6.5% |
| Weighted processing | 18.0% | 18.0% | 16.5% | 12.9% | 16.5% | 18.0% |

**Table. S4** Training parameters for 6 models used for PCE prediction.

| ML algorithms | Training parameters |
|---|---|
| LightGBM | 'regressor__n_estimators': 300,<br>'regressor__max_depth': 6,<br>'regressor__ learning_rate': 0.05,<br>'regressor__ subsample': 0.8,<br>'regressor__ colsample_bytree ': 0.8 |
| SVR | 'regressor__ C ': 100,<br>'regressor__ gamma ': 'scale',<br>'regressor__ kernel ': 'rbf' |
| RF | 'regressor__n_estimators': 300,<br>'regressor__max_depth': 15,<br>'regressor__min_samples_split': 5,<br>'regressor__min_samples_leaf': 2,<br>'regressor__max_features': 'sqrt',<br>'regressor__bootstrap': True,<br>'regressor__random_state':42, |
| MLP | 'regressor__activation': ' relu ',<br>'regressor__alpha': 0.001,<br>'regressor__ max_iter': 1000,<br>'regressor__ learning_rate_init ': '0.001',<br>'regressor__solver': 'adam',<br>'regressor__ early_stopping ': True |
| AdaBoost | 'regressor__ max_depth ': 4,<br>'regressor__ min_samples_split ': 5,<br>'regressor__ min_samples_leaf ': 2 |
| XGBoost | 'regressor__colsample_bytree': 0.8,<br>'regressor__gamma': 0.1,<br>'regressor__learning_rate': 0.05,<br>'regressor__max_depth': 6,<br>'regressor__n_estimators': 300,<br>'regressor__subsample': 0.8,<br>'regressor__ min_child_weight ': 3 |

**Table. S5** Training metrics of 6 ML models on training and testing sets.

| ML algorithms | Train set | | | Test set | | |
|---|---|---|---|---|---|---|
| | $R^2$ | RMSE(%) | MAPE(%) | $R^2$ | RMSE(%) | MAPE(%) |
| XGBoost | 0.9475 | 1.0678 | 5.4055 | 0.8151 | 1.9699 | 9.787 |
| RF | 0.8667 | 1.5102 | 8.3044 | 0.63734 | 2.5585 | 14.3297 |
| AdaBoost | 0.8483 | 1.8154 | 10.9258 | 0.7661 | 2.2157 | 13.2769 |
| SVR | 0.882 | 1.6007 | 11.4056 | 0.7303 | 2.3789 | 14.5416 |
| LightGBM | 0.9235 | 1.2892 | 6.0133 | 0.7867 | 2.1158 | 11.069 |
| MLP | 0.6834 | 2.6226 | 15.1249 | 0.5436 | 3.0949 | 18.2154 |

**Table. S6** The optimal perovskite components and predicted PCE for six preparation processes under small area (0.03 $cm^2$) formal and inverted structures obtained from the triangle heatmap.

| Structure | Preparation process | Cs | FA | MA | Br | I | Cl | predicted PCE(%) |
|---|---|---|---|---|---|---|---|---|
| Formal (*n-i-p*) | Spin coating | 0.1 | 0.9 | 0.0 | 0 | 1 | 0 | 27.55 |
| | Blade Coating | 0.1 | 0.89 | 0.01 | 0 | 1 | 0 | 27.18 |
| | Slot-Die Coating | 0.1 | 0.9 | 0.0 | 0 | 1 | 0 | 26.87 |
| | Evaporation | 0.07 | 0.89 | 0.04 | 0 | 1 | 0 | 26.56 |
| | Inkjet printing | 0.06 | 0.9 | 0.04 | 0.01 | 0.99 | 0 | 26.69 |
| | VASP | 0.1 | 0.89 | 0.01 | 0.02 | 0.98 | 0 | 26.98 |
| Inverted (*p-i-n*) | Spin coating | 0.12 | 0.86 | 0.02 | 0.02 | 0.98 | 0 | 27.57 |
| | Blade Coating | 0.1 | 0.9 | 0 | 0.02 | 0.98 | 0 | 27.01 |
| | Slot-Die Coating | 0.1 | 0.86 | 0.04 | 0.02 | 0.98 | 0 | 26.81 |
| | Evaporation | 0.1 | 0.86 | 0.04 | 0.02 | 0.98 | 0 | 26.84 |
| | Inkjet printing | 0.1 | 0.89 | 0.01 | 0.02 | 0.98 | 0 | 26.85 |
| | VASP | 0.12 | 0.88 | 0 | 0.03 | 0.97 | 0 | 27.03 |

**Table. S7** The optimal perovskite components and PCE for six preparation processes under large area (10 $cm^2$) formal and inverted structures obtained from the triangle heatmap.

| Structure | Preparation process | Cs | FA | MA | Br | I | Cl | PCE(%) |
|---|---|---|---|---|---|---|---|---|
| Formal (*n-i-p*) | Spin coating | 0.08 | 0.89 | 0.03 | 0 | 1 | 0 | 22.99 |
| | Blade Coating | 0.1 | 0.88 | 0.02 | 0.03 | 0.97 | 0 | 22.47 |
| | Slot-Die Coating | 0.07 | 0.89 | 0.04 | 0 | 1 | 0 | 22.94 |
| | Evaporation | 0.06 | 0.9 | 0.04 | 0 | 1 | 0 | 22.46 |
| | Inkjet printing | 0.07 | 0.88 | 0.05 | 0 | 1 | 0 | 22.79 |
| | VASP | 0.06 | 0.9 | 0.04 | 0 | 1 | 0 | 23.17 |
| Inverted (*p-i-n*) | Spin coating | 0.07 | 0.89 | 0.04 | 0 | 1 | 0 | 22.70 |
| | Blade Coating | 0.1 | 0.88 | 0.02 | 0.03 | 0.97 | 0 | 22.81 |
| | Slot-Die Coating | 0.04 | 0.94 | 0.02 | 0.04 | 0.96 | 0 | 22.49 |
| | Evaporation | 0.11 | 0.89 | 0 | 0.03 | 0.97 | 0 | 21.80 |
| | Inkjet printing | 0.05 | 0.94 | 0.01 | 0.04 | 0.96 | 0 | 21.99 |
| | VASP | 0.07 | 0.9 | 0.03 | 0.01 | 0.99 | 0 | 21.53 |

**Table S8.** Ten additional data points were collected to verify the stability of the model predictions.

| Number / Factor | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Voc (V) | 1.112 | 1.0725 | 1.064 | 1.14 | 1.154 | 1.16 | 0.993 | 1.08 | 1.137 | 1.172 |
| Jsc (mA/cm$^2$) | 24.63 | 19.6 | 20.8 | 25.85 | 25.09 | 25.02 | 26.8 | 20.54 | 25.39 | 25.53 |
| FF (%) | 77.82 | 78.31 | 76 | 85 | 77.7 | 84.69 | 72.1 | 78.49 | 80.75 | 84.44 |
| Area (cm$^2$) | 25 | 16 | 12.5 | 0.07 | 6.25 | 0.07 | 14 | 16 | 0.09 | 0.09 |
| Substrate | FTO | ITO | FTO | ITO | ITO | ITO | FTO | ITO | ITO | ITO |
| Substrate function (eV) | 4.5 | 4.7 | 4.5 | 4.7 | 4.7 | 4.7 | 4.5 | 4.7 | 4.7 | 4.7 |
| Cs | 0 | 0.05 | 0.05 | 0 | 0 | 0.07 | 0.05 | 0.05 | 0.05 | 0.05 |
| FA | 1 | 0.85 | 0.95 | 0.92 | 1 | 0.93 | 0.93 | 0.95 | 0.95 | 0.95 |
| MA | 0 | 0.05 | 0 | 0.08 | 0 | 0 | 0.02 | 0 | 0 | 0 |
| Br | 0 | 0.15 | 0 | 0 | 0 | 0 | 0.15 | 0 | 0 | 0 |
| I | 3 | 2.85 | 3 | 3 | 3 | 3 | 2.585 | 3 | 3 | 3 |
| Cl | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| perovskite Eg (eV) | 1.48 | 1.518 | 1.492 | 1.486 | 1.48 | 1.498 | 1.518 | 1.492 | 1.492 | 1.492 |
| ETL_1 | PCBM | $SnO_2$ | $SnO_2$ | $C_{60}$ | $C_{60}$ | $C_{60}$ | $C_{60}$ | $C_{60}$ | $C_{60}$ | $C_{60}$ |
| ETL_1 CB (eV) | 3.8 | 4.5 | 4.5 | 3.9 | 3.9 | 3.9 | 3.9 | 3.9 | 3.9 | 3.9 |
| ETL_1 VB (eV) | 6.1 | 7.5 | 7.5 | 5.8 | 5.8 | 5.8 | 5.8 | 5.8 | 5.8 | 5.8 |
| ETL_2 | BCP | none | none | BCP | BCP | BCP | BCP | BCP | BCP | BCP |
| ETL_2 CB | 2.8 | 0 | 0 | 2.8 | 2.8 | 2.8 | 2.8 | 2.8 | 2.8 | 2.8 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| (eV) | | | | | | | | | | |
| ETL_2 VB (eV) | 6.7 | 0 | 0 | 6.7 | 6.7 | 6.7 | 6.7 | 6.7 | 6.7 | 6.7 |
| HTL_1 | $NiO_x$ | $NiO_x$ | Spiro-OMeTAD | TP-TPA | $NiO_x$ | PTAA | $NiO_x$ | $NiO_x$ | $NiO_x$ | $NiO_x$ |
| HTL_1 CB (eV) | 1.75 | 1.75 | 2.2 | 2.1 | 1.75 | 2.1 | 1.75 | 1.75 | 1.75 | 1.75 |
| HTL_1 VB (eV) | 5.1 | 5.1 | 5.22 | 5 | 5.1 | 5.3 | 5.1 | 5.1 | 5.1 | 5.1 |
| HTL_2 | MeO-2PACz | $NiO_x$ | none | none | MeO-4PACz | none | none | none | MeO-4PACz | MeO-4PACz |
| HTL_2 CB (eV) | 2.3 | 1.75 | 0 | 0 | 2.35 | 0 | 0 | 0 | 2.35 | 2.35 |
| HTL_2 VB (eV) | 5.3 | 5.1 | 0 | 0 | 5.35 | 0 | 0 | 0 | 5.35 | 5.35 |
| Electrode | Ag | Ag | Au | Ag | Ag | Cu | Cu | Cu | Ag | Ag |
| Electrode function (eV) | 4.3 | 4.3 | 5.28 | 4.3 | 4.3 | 4.8 | 4.8 | 4.8 | 4.3 | 4.3 |
| additive_1 | others | Organic ammonium salt | Organic ammonium salt | none | none | others | Organic ammonium salt | Organic ammonium salt | none | Organic ammonium salt |
| additive_2 | none | none | none | none | none | none | none | none | none | none |
| Surface passivator | Organic ammonium salts | Organic ammonium salts | Organic ammonium salts | Organic ammonium salts | Organic ammonium salts | others | polymer | none | Organic ammonium salts | Organic ammonium salts |
| Structure | p-i-n | p-i-n | n-i-p | p-i-n | p-i-n | p-i-n | p-i-n | p-i-n | p-i-n | p-i-n |
| Preparation | Spin coating | Spin coating | Blade Coating | Spin coating | Blade Coating | Slot-Die | Slot-Die | Evaporation | Spin coating | Spin coating |

| process | | | | | | Coating | Coating | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| DOI | 10.1002/anie.202516464 | 10.1002/smtd.202501325 | 10.1039/d5el00032g | 10.1016/j.cej.2025.168738 | 10.1039/d5ee04570c | 10.1016/j.jechem.2025.08.028 | 10.1016/j.solmat.2025.113858 | 10.1016/j.solmat.2025.113793 | 10.1016/j.jechem.2025.05.039 | 10.1016/j.jechem.2025.05.039 |
| Experimental efficiency (%) | 21.31 | 16.46 | 18.46 | 24.8 | 21.3 | 24.58 | 19.19 | 17.58 | 23.32 | 25.27 |
| Predicted efficiency (%) | 20.67 | 17.17 | 17.25 | 23.41 | 21.22 | 23.97 | 19.96 | 17.97 | 23.62 | 23.63 |
| **relative error (%)** | **2.988** | **4.333** | **6.578** | **5.625** | **0.368** | **2.470** | **3.992** | **2.237** | **1.299** | **6.500** |

**Table. S9** Decay data of PCE with area variation under the optimal perovskite composition of six preparation processes for formal structure and inverted structure obtained in a small area.

| Structure | Area ($cm^2$) | Predicted PCE (%) | | | | | |
|---|---|---|---|---|---|---|---|
| | | Spin coating | Blade Coating | Slot-Die Coating | Evaporation | Inkjet printing | VASP |
| Formal (*n-i-p*) | 0.03 | 27.55 | 27.18 | 26.87 | 26.56 | 26.69 | 26.98 |
| | 10.03 | 22.44 | 22.39 | 22.51 | 21.81 | 22.09 | 22.49 |
| | 20.03 | 19.08 | 18.87 | 18.68 | 18.71 | 18.91 | 19.14 |
| | 30.03 | 18.09 | 17.76 | 18.49 | 18.37 | 18.73 | 18.77 |
| | 40.03 | 17.43 | 17.67 | 18.17 | 18.04 | 18.62 | 18.36 |
| | 50.03 | 17.35 | 17.58 | 17.83 | 18 | 18.47 | 18.01 |
| | 60.03 | 17.18 | 17.5 | 17.61 | 17.53 | 18.21 | 17.73 |
| | 70.03 | 17.14 | 17.42 | 17.49 | 17.39 | 18.05 | 17.54 |
| | 80.03 | 17.09 | 17.34 | 17.37 | 17.26 | 17.69 | 17.47 |
| | 90.03 | 17.06 | 17.27 | 17.33 | 17.23 | 17.45 | 17.42 |
| | 100.03 | 17.04 | 17.2 | 17.32 | 17.21 | 17.41 | 17.38 |
| Inverted (*p-i-n*) | 0.03 | 27.57 | 27.01 | 26.81 | 26.84 | 26.85 | 27.03 |
| | 10.03 | 22.61 | 22.79 | 22.49 | 21.79 | 21.99 | 21.49 |
| | 20.03 | 20.09 | 20.68 | 20.46 | 19.57 | 19.78 | 19.19 |
| | 30.03 | 18.62 | 19.76 | 19.51 | 18.59 | 18.76 | 18.25 |
| | 40.03 | 17.89 | 19.29 | 19.06 | 18.16 | 18.3 | 17.86 |
| | 50.03 | 17.52 | 19.06 | 18.84 | 17.98 | 18.09 | 17.75 |
| | 60.03 | 17.32 | 18.94 | 18.75 | 17.88 | 17.99 | 17.67 |
| | 70.03 | 17.21 | 18.89 | 18.71 | 17.85 | 17.95 | 17.64 |
| | 80.03 | 17.15 | 18.86 | 18.69 | 17.84 | 17.93 | 17.62 |
| | 90.03 | 17.13 | 18.85 | 18.68 | 17.83 | 17.92 | 17.6 |
| | 100.03 | 17.11 | 18.84 | 18.67 | 17.82 | 17.91 | 17.59 |