Highly accurate prediction of reaction performance and enantioselectivity with a uniform machine learning protocol

Xiaofei Sun[#], Jingyuan Zhu[#], Hengzhi You[*], Bin Chen[*], Fen-Er Chen[*]

Supplementary Information

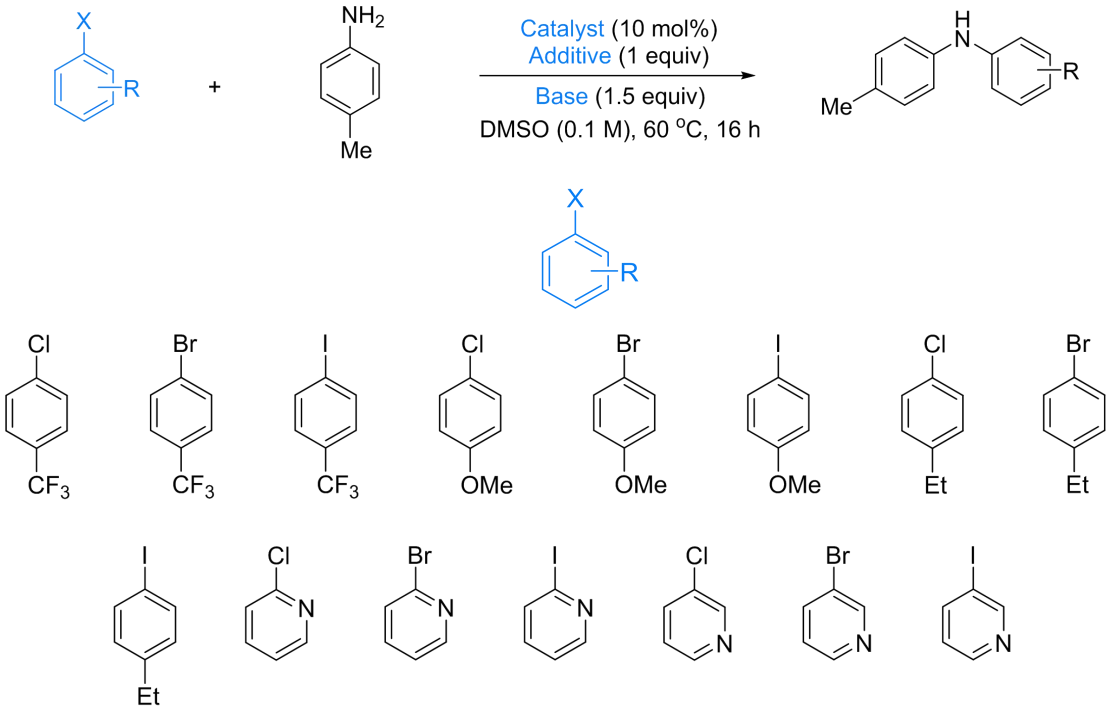
Table of contents

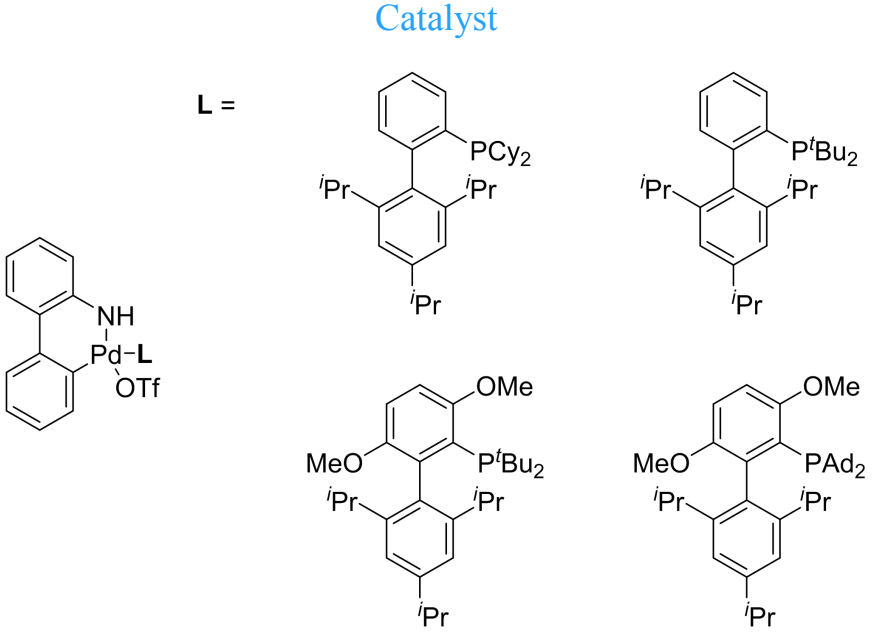
Supplementary Note 1: Data mining

1.1 Data availability

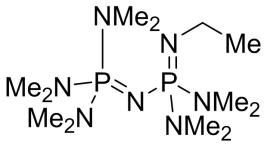
Buchwald-Hartwig reactions[1], Palladium-catalyzed cross coupling data for Suzuki-Miyaura[2], asymmetric N, S-acetal formation[3] and asymmetric hydrogenation reactions[4] were used as objectives for machine learning.

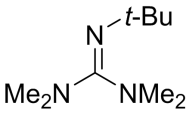
1.1.1 Buchwald-Hartwig amination (BHA)

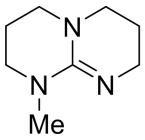




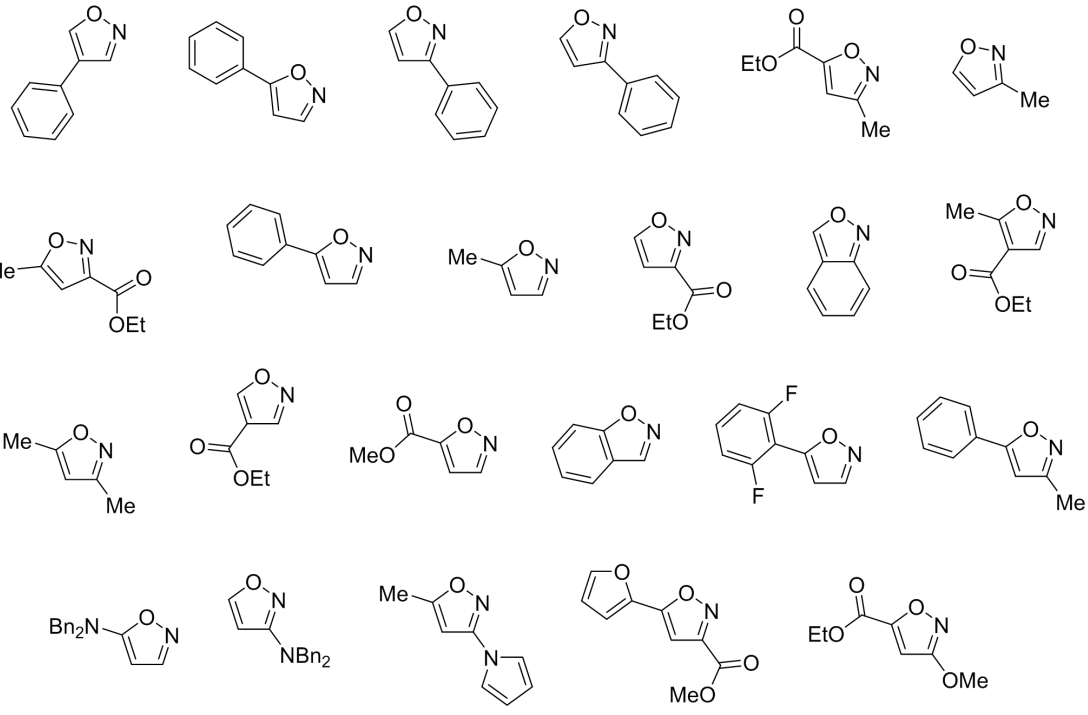
Base



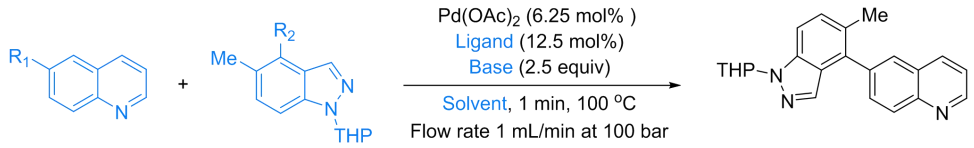


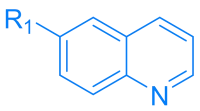


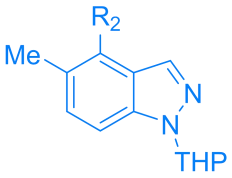
Additive



1.1.2 Suzuki-Miyaura coupling (SMC)







R_1 = Cl, Br, OTf, I, $B(OH)_2$, Bpin, BF_3k R_2 = Br, $B(OH)_2$, Bpin, BF_3k

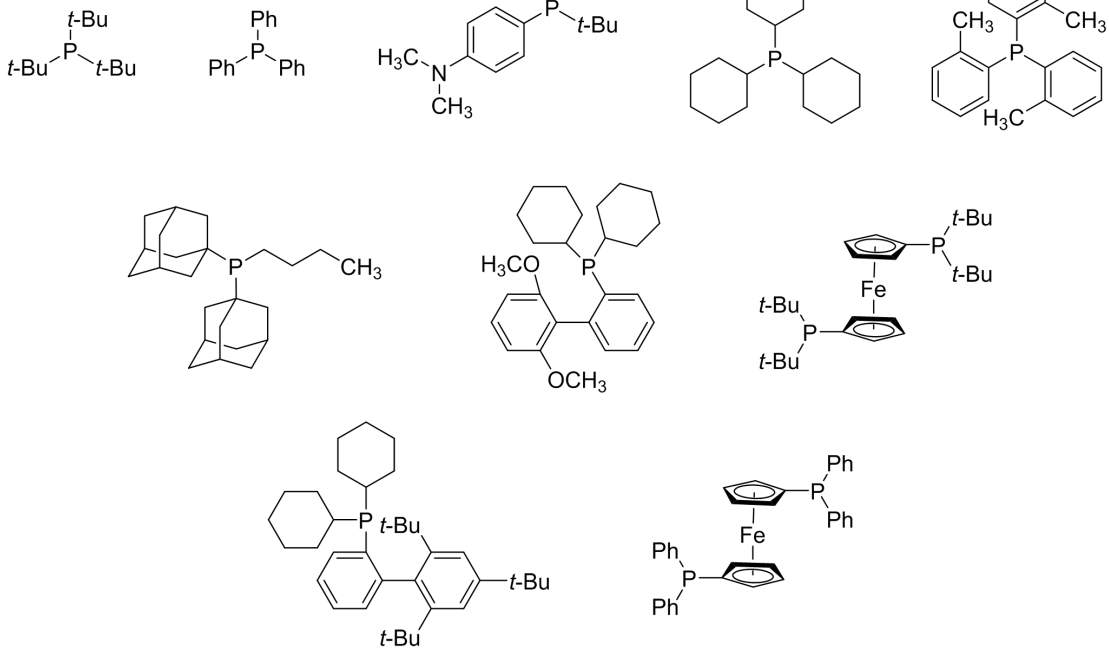
Base

Et_3N, LiOtBu, K_3PO_4, KOH, $NaHCO_3$, NaOH

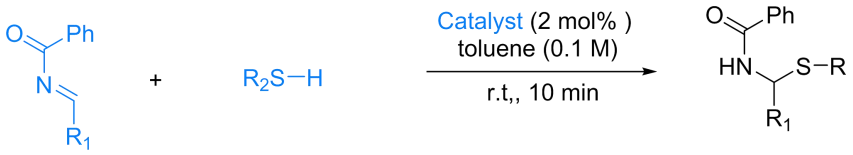
Solvent

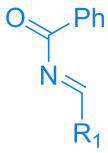
MeOH, THF, MeCN, DMF

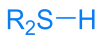
Ligand



1.1.3 Asymmetric N, S-acetal formation (ANSA)

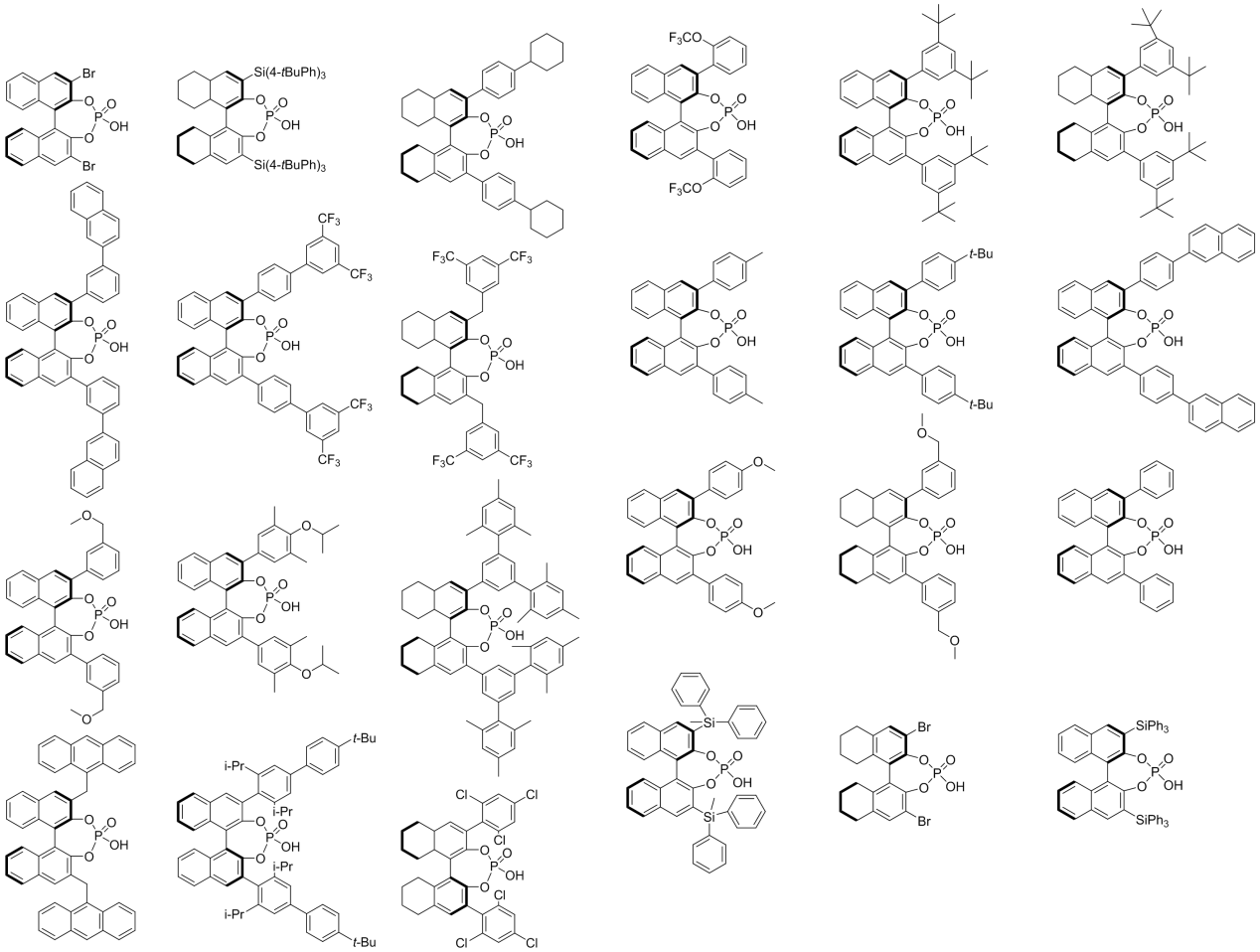


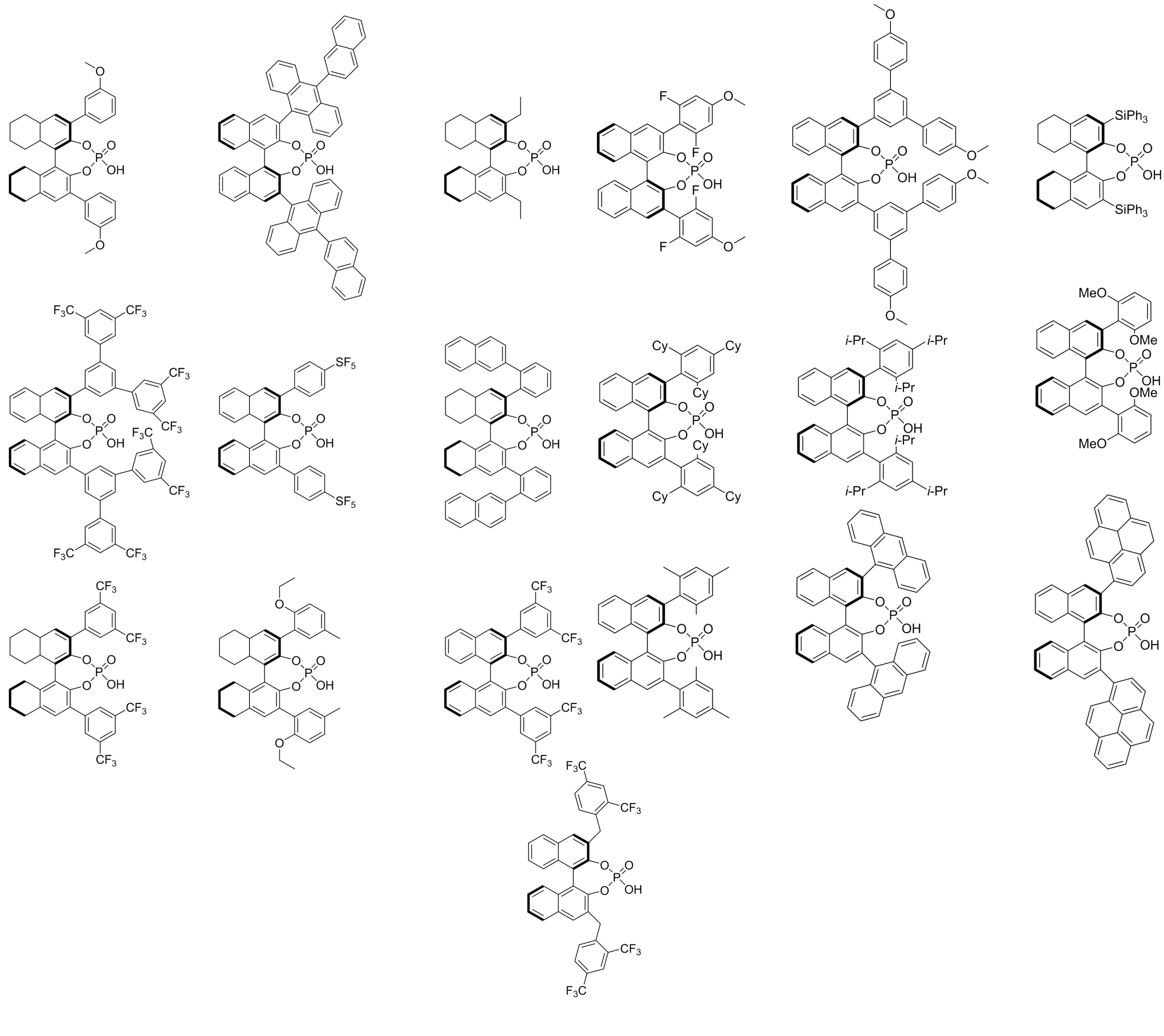




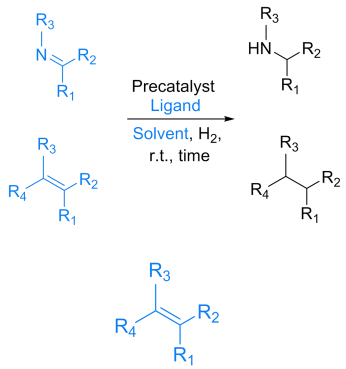
R_1 = Ph,	R_2 = Ph,
4-CF_3Ph,	Cy,
4-OMePh,	Et,
1-naphthyl,	4-OMePh,
2,4-dichlorophenyl	2-MePh

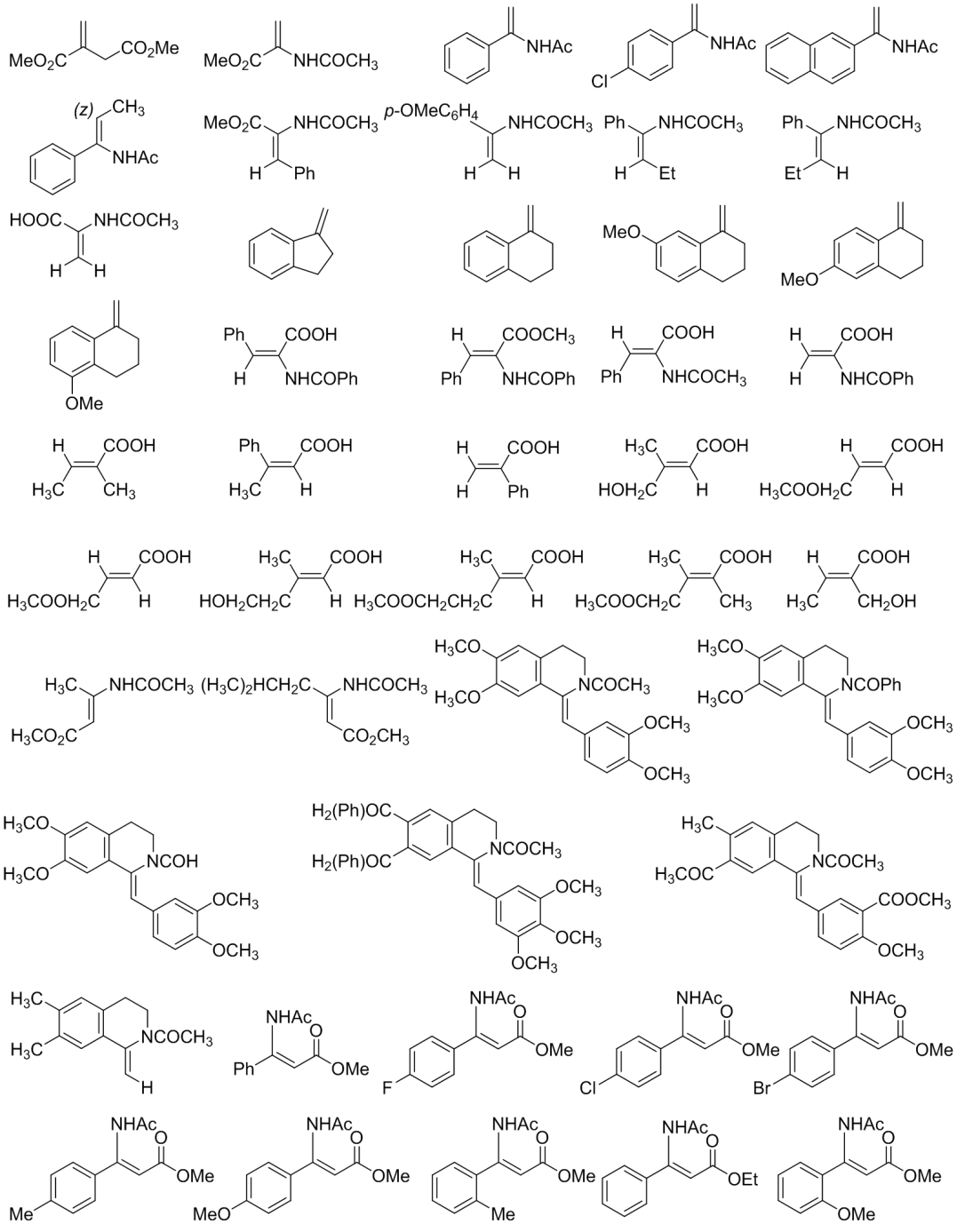
Catalyst

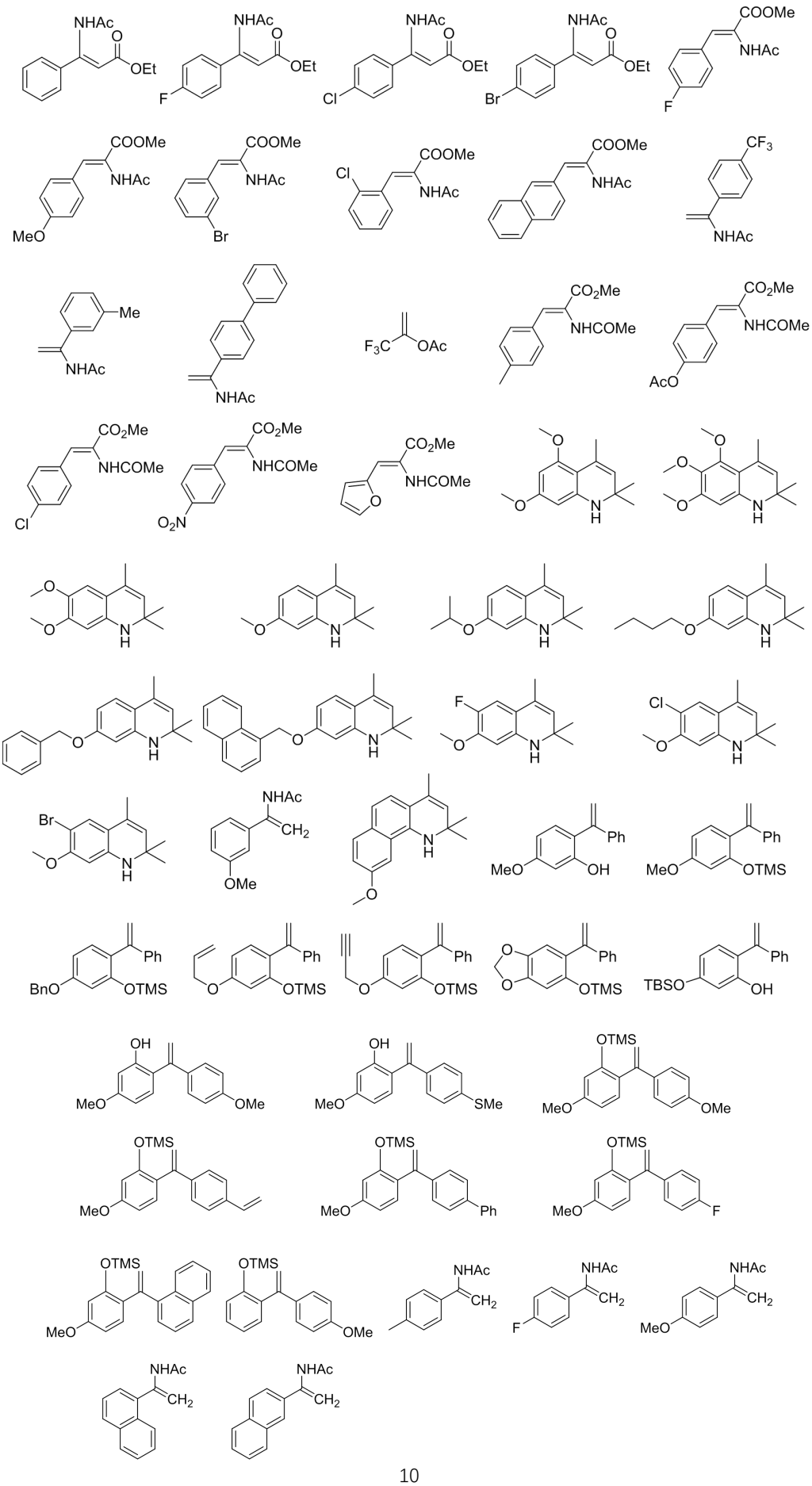


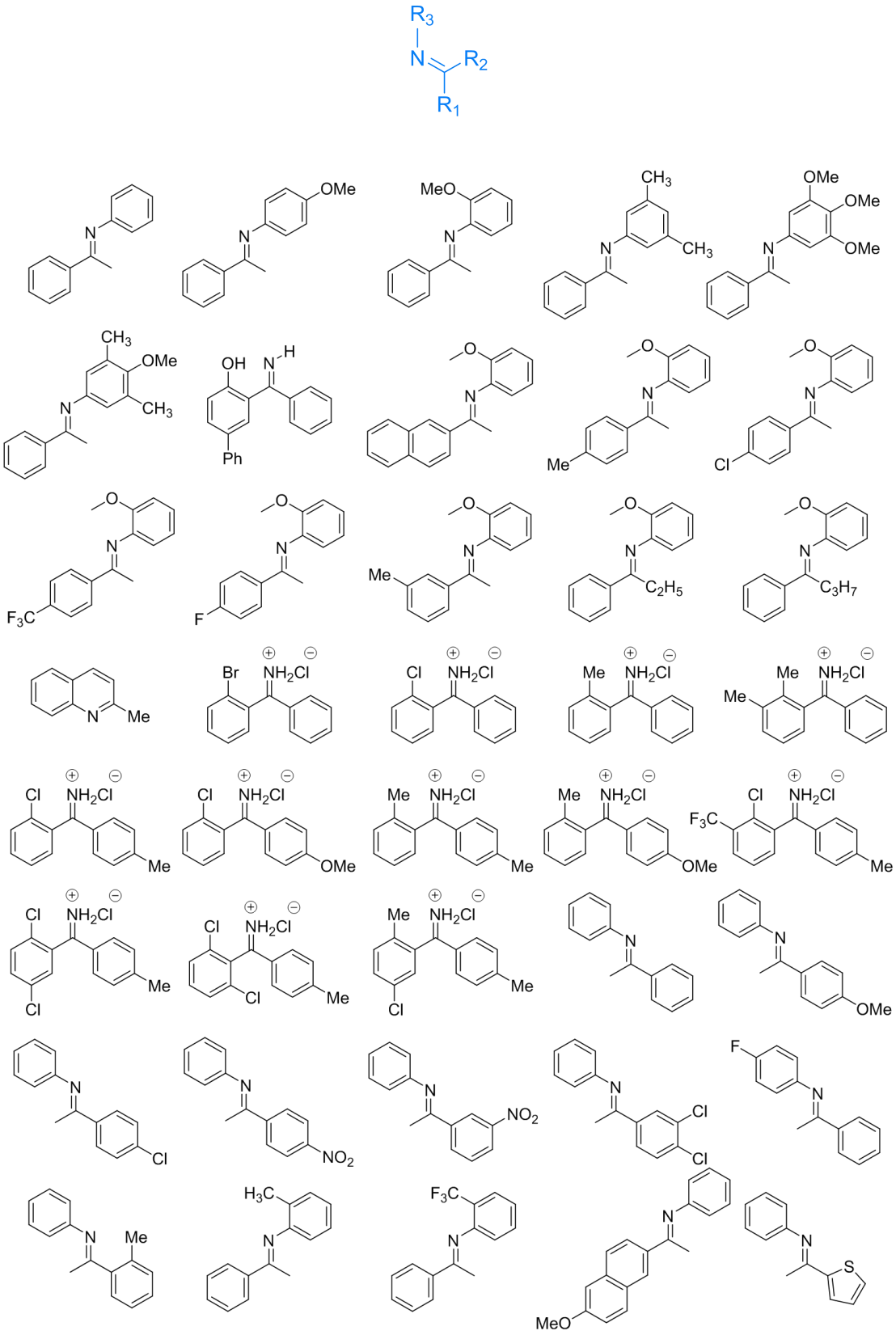


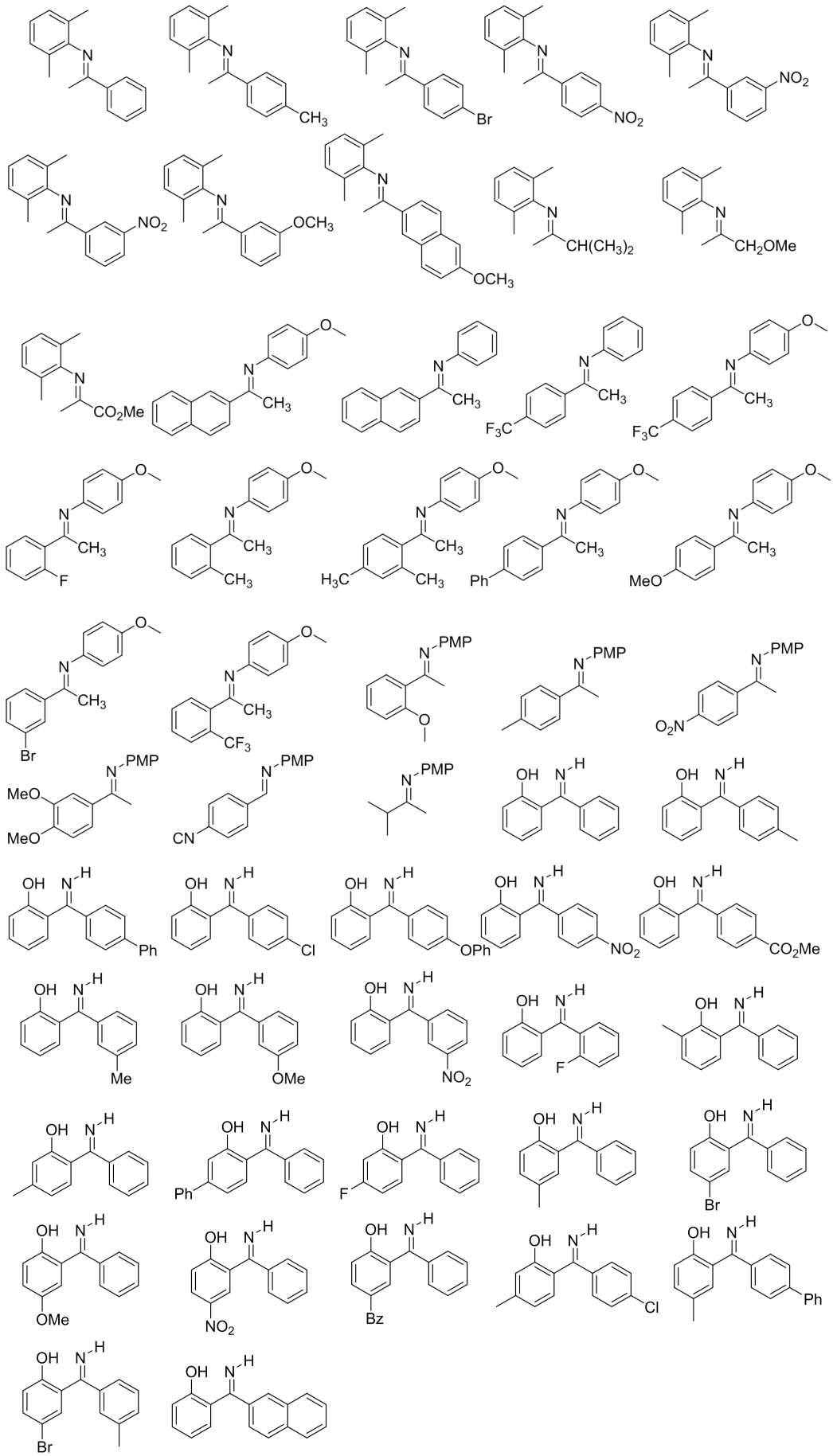
1.1.4 Asymmetric hydrogenation reactions (AHR)



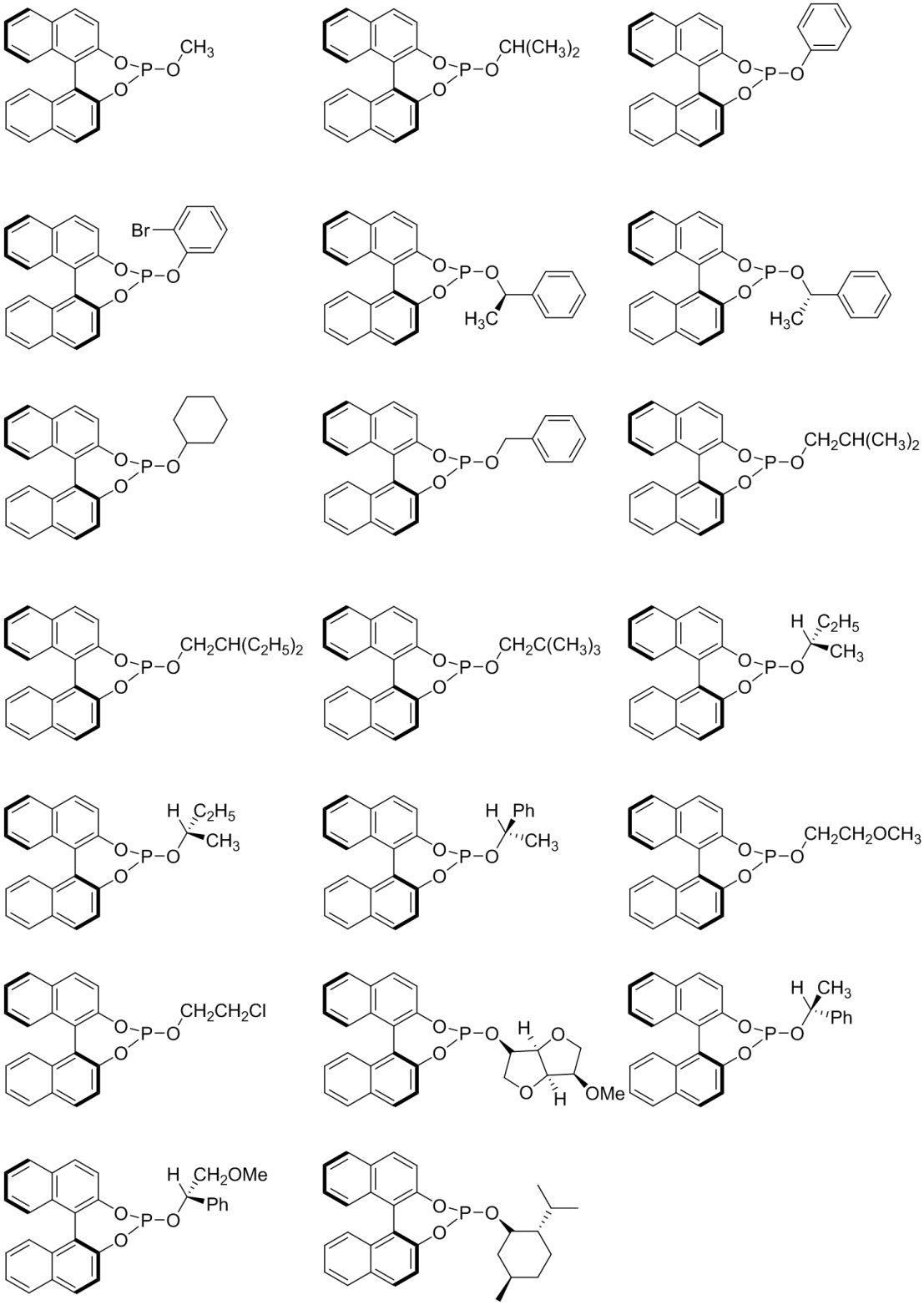


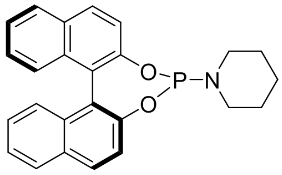
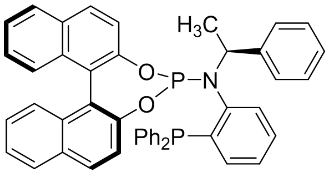
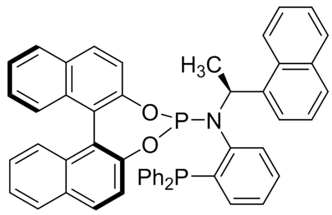
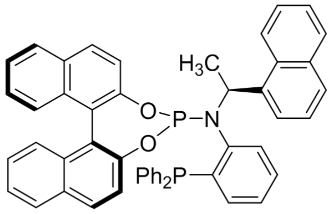
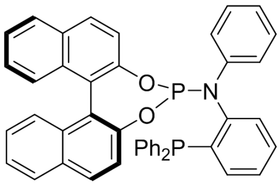
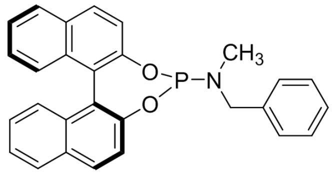
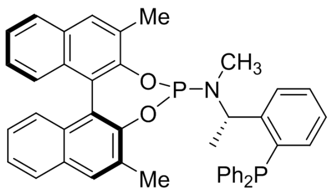
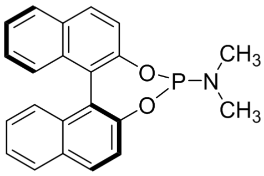
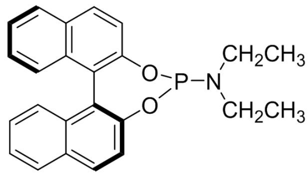
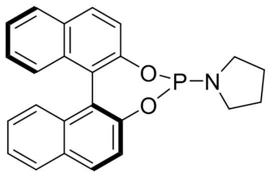
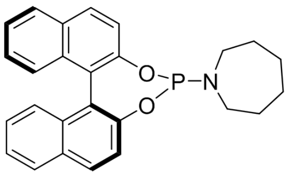
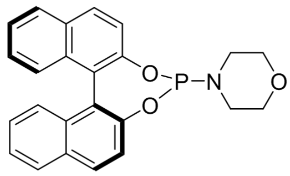
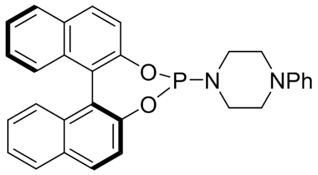
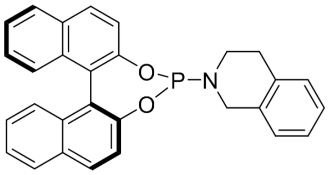
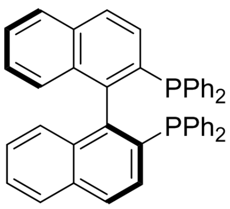
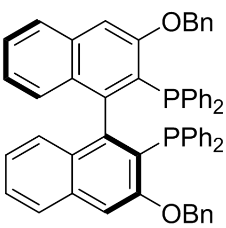
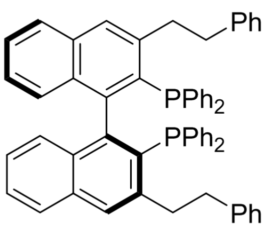
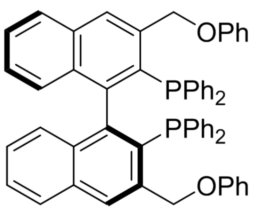
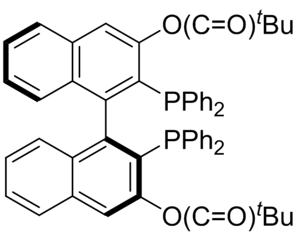
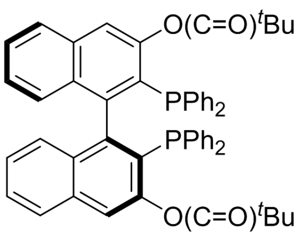

NHAc
O
OEt
F
Cl
Br
COOMe
NHAc
MeO
CF3
Me
F3C
OAc
CO2Me
NHCOMe
AcO
O2N
N
H
CH2
OMe
OH
OTMS
Ph
BnO
TBSO
SMe

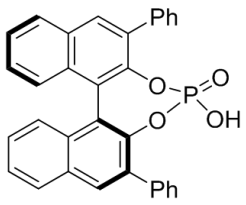

R3
N
R2
R1
OMe
MeO
CH3
OMe
OMe
OMe
CH3
CH3
OH
N
H
Ph
O
N
Me
O
N
Cl
O
N
F3C
O
N
F
O
N
Me
O
N
C2H5
O
N
C3H7
N
Me
Br
NH2Cl
Cl
NH2Cl
Me
NH2Cl
Me
Me
NH2Cl
Cl
NH2Cl
Me
Cl
NH2Cl
OMe
Me
NH2Cl
Me
Me
NH2Cl
OMe
F3C
Cl
NH2Cl
Me
Cl
NH2Cl
Cl
Me
Cl
NH2Cl
Cl
Me
Me
NH2Cl
Cl
Me
N
N
OMe
N
Cl
N
NO2
N
Cl
Cl
F
N
N
Me
H3C
N
F3C
N
N
MeO
N
S

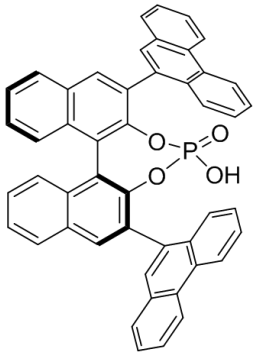


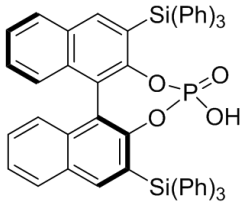
Ligand

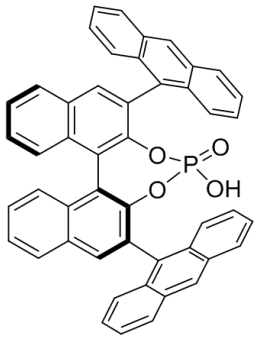

CH3
CH(CH3)2
Br
H3C
H3C
CH2CH(CH3)2
CH2CH(C2H5)2
CH2C(CH3)3
H C2H5 CH3
H C2H5 CH3
H Ph CH3
CH2CH2OCH3
CH2CH2Cl
H CH3 Ph
H
O
H
OMe
H CH2OMe Ph

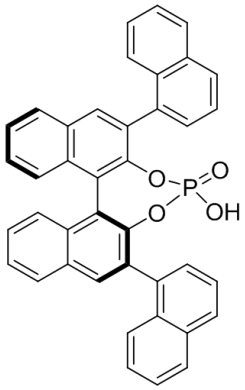





















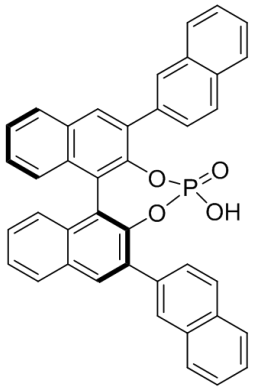

Ph
O
P
O
OH
O
Ph

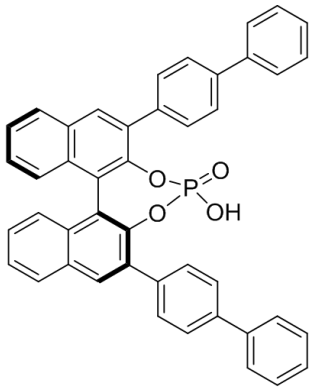

O
P
O
OH
O

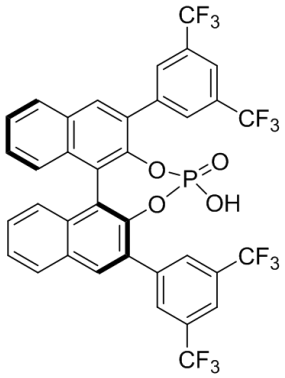

Si(Ph)3
O
P
O
OH
O
Si(Ph)3

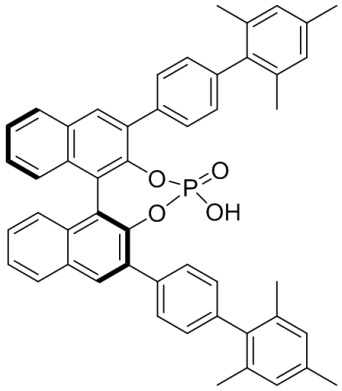

O
P
O
OH
O

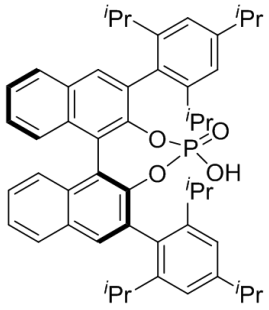

O
P
O
OH
O

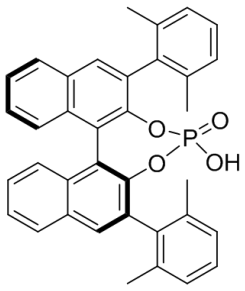

O
P
O
OH
O

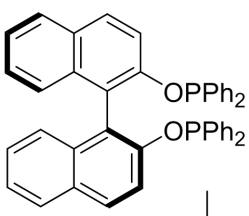

O
P
O
OH
O

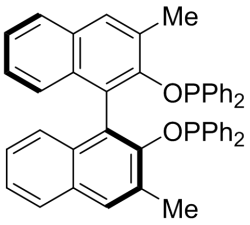

CF3
CF3
O
P
O
OH
O
CF3
CF3

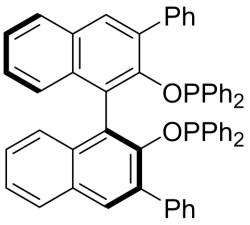

O
P
O
OH
O

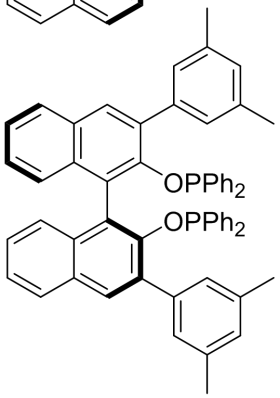

iPr
iPr
iPr
O
P
O
OH
O
iPr
iPr
iPr

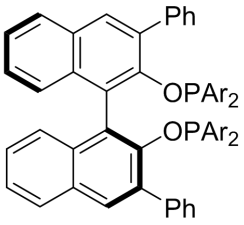

O
P
O
OH
O



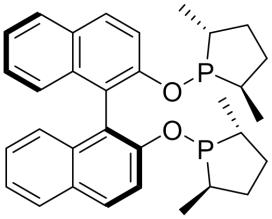


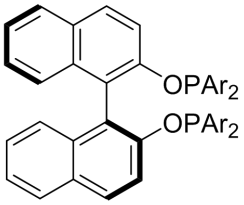






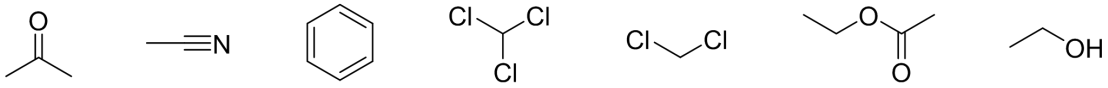
Ar = 3,5-$Me_2C_6H_3$

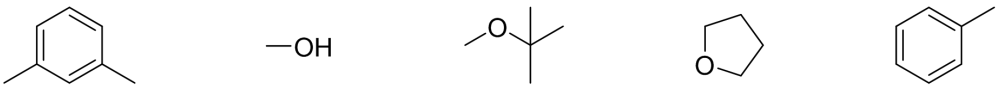




Ar = 3,5-$Me_2C_6H_3$

Solvent





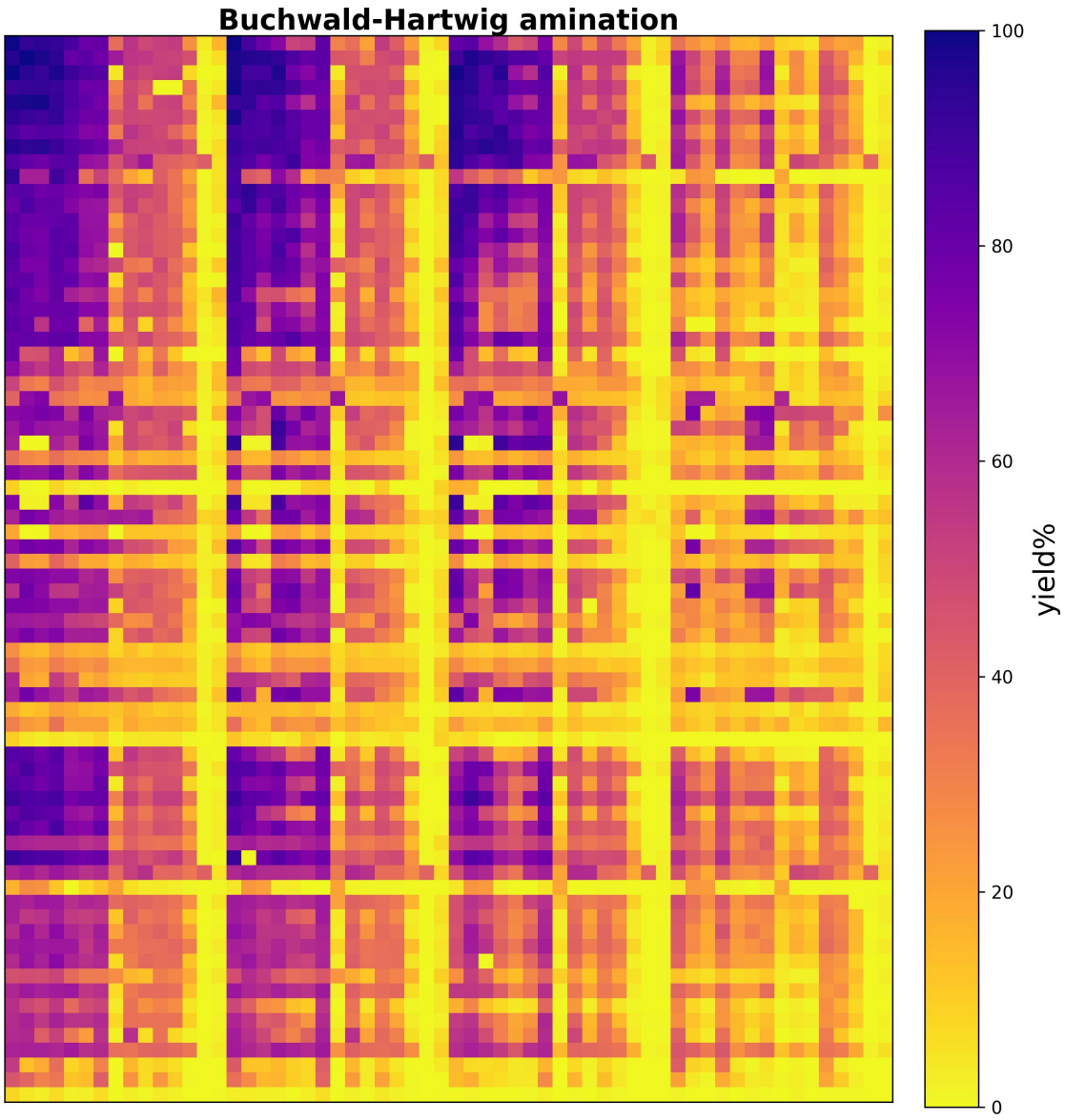


Fig .S1 Heatmaps for yields distribution of BHA.

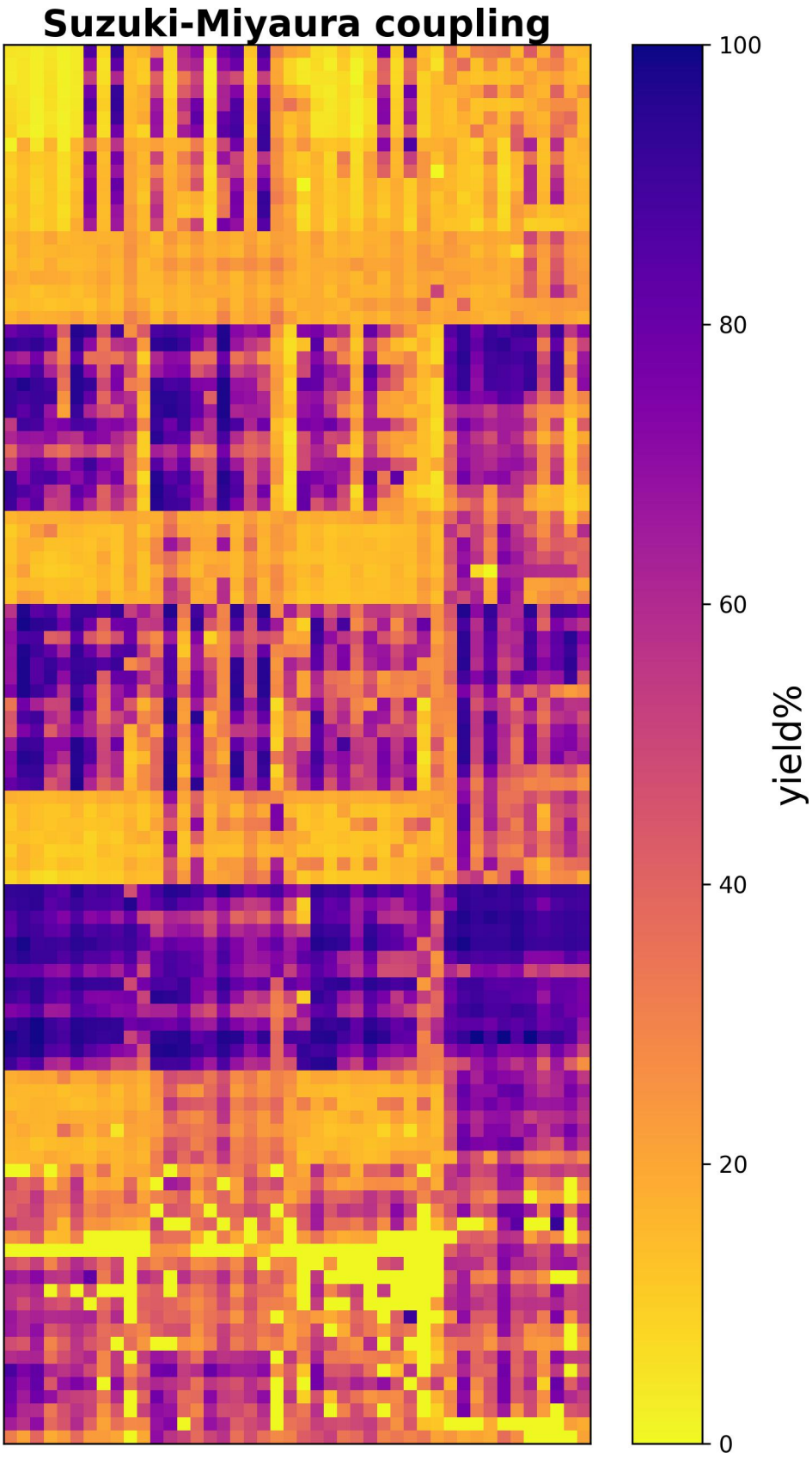


Fig .S2 Heatmaps for yields distribution of SMC.

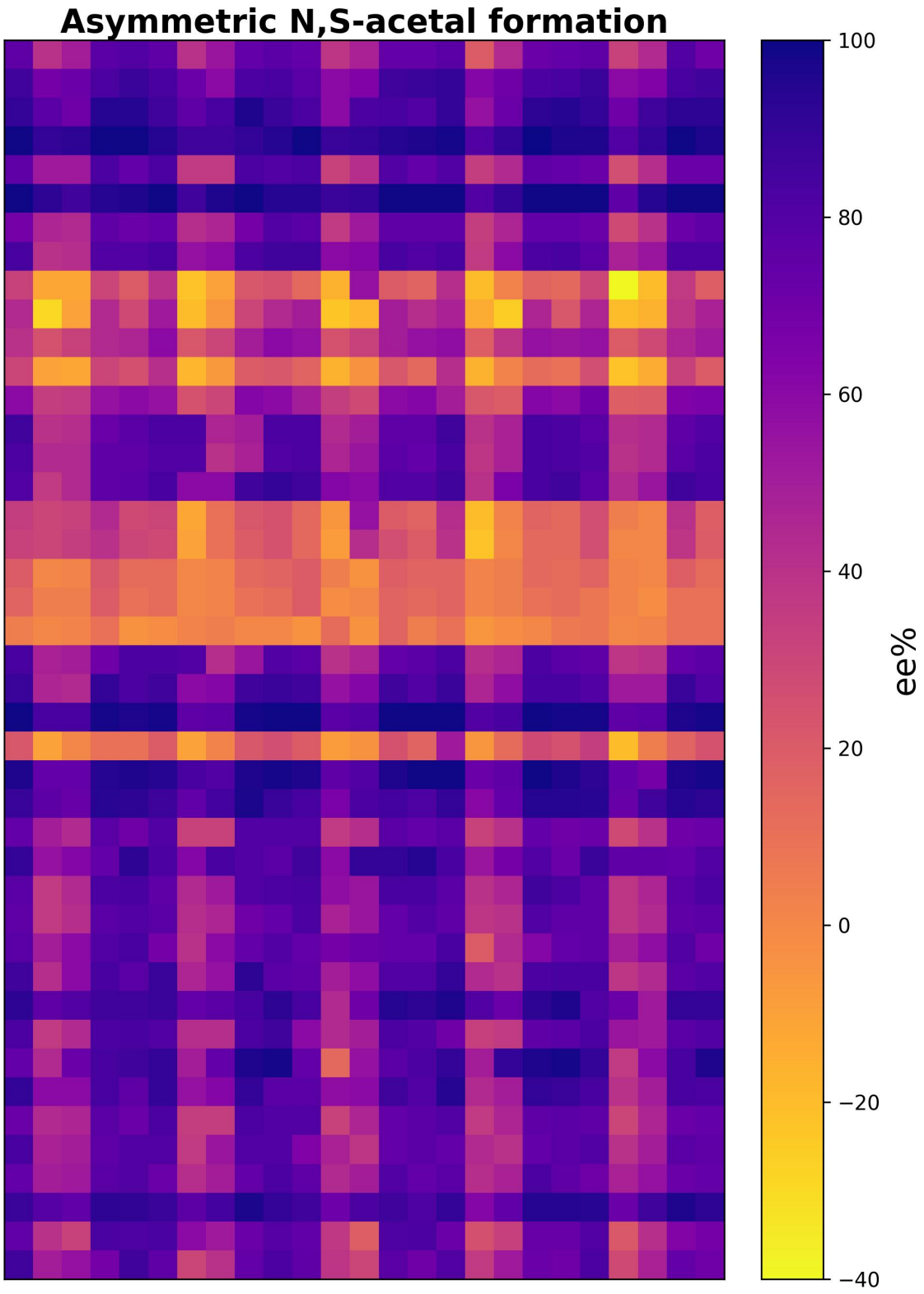


Fig .S3 Heatmaps for ee distribution of ANSA.

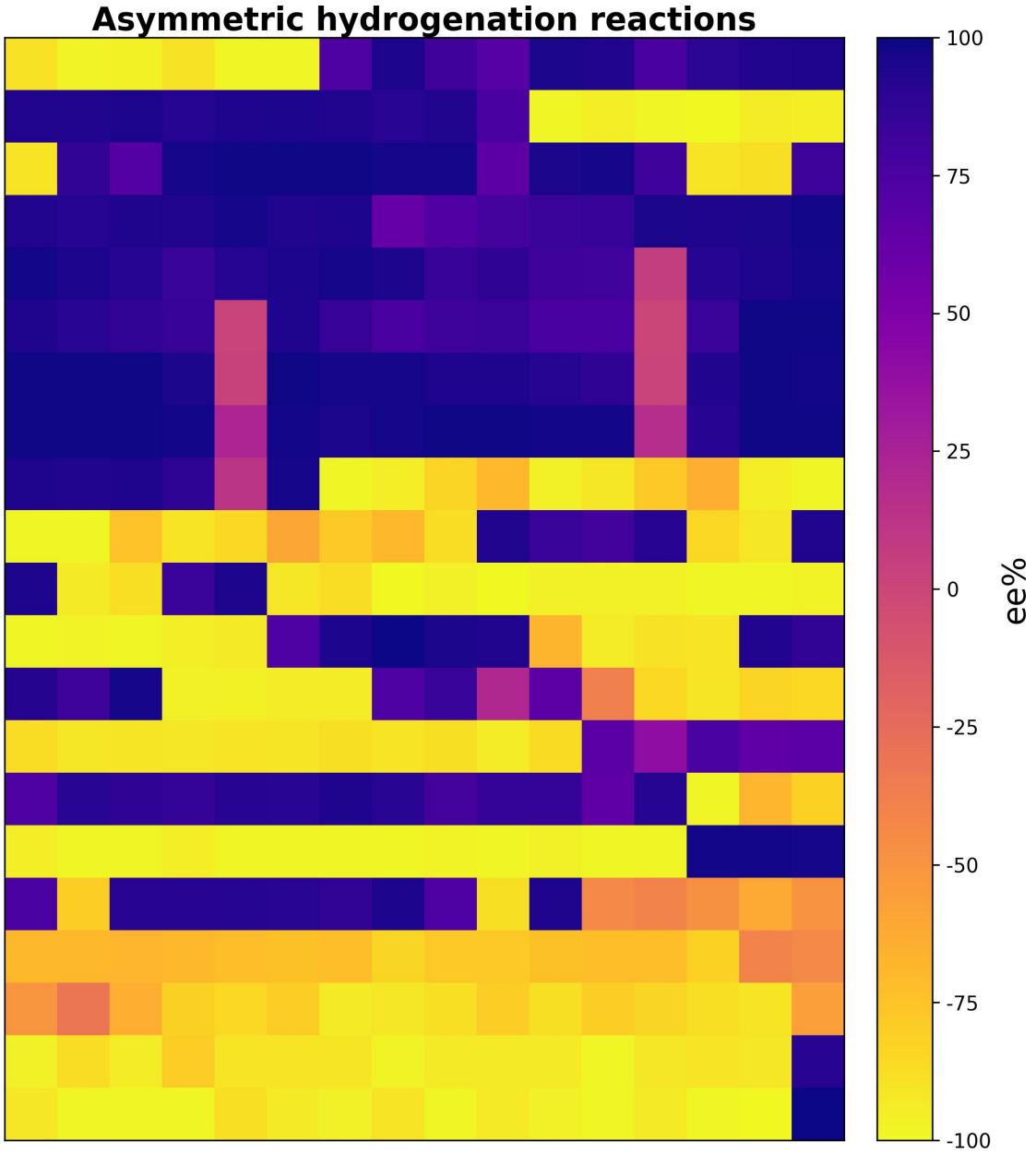


Fig .S4 Heatmaps for ee distribution of AHR.

1.2 Analysis of data

Fig 5-8 show the similarity score matrix of each category in each data set. Color of each grid reflects the Tanimoto coefficient[5] between molecule in row and molecule in column.

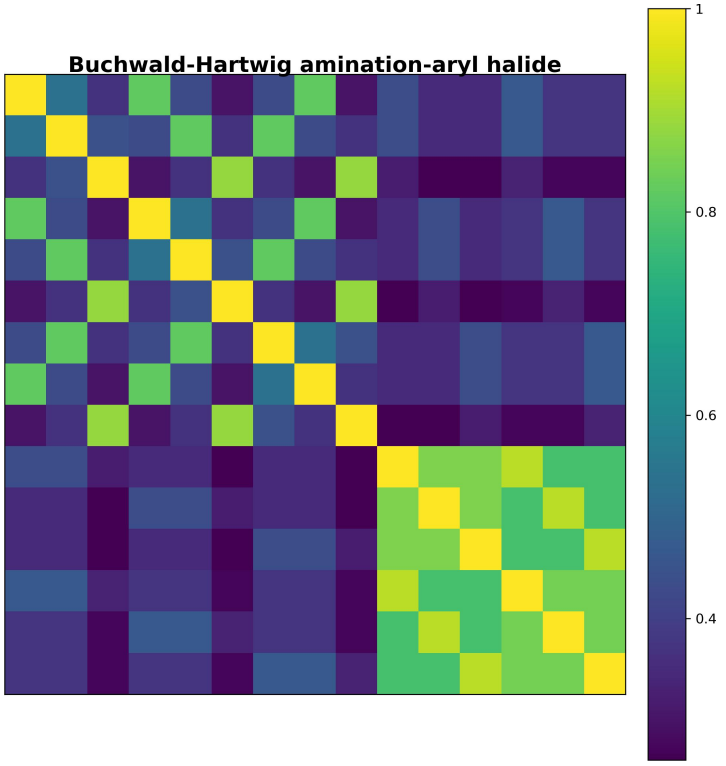


Fig .S5 Similarity score matrix of aryl halides in BHA.

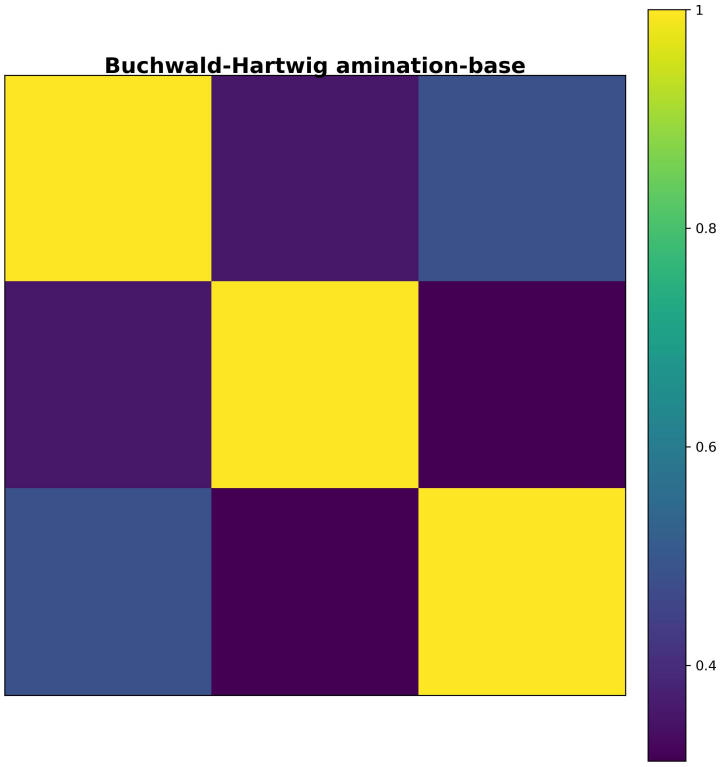


Fig .S6 Similarity score matrix of bases in BHA.

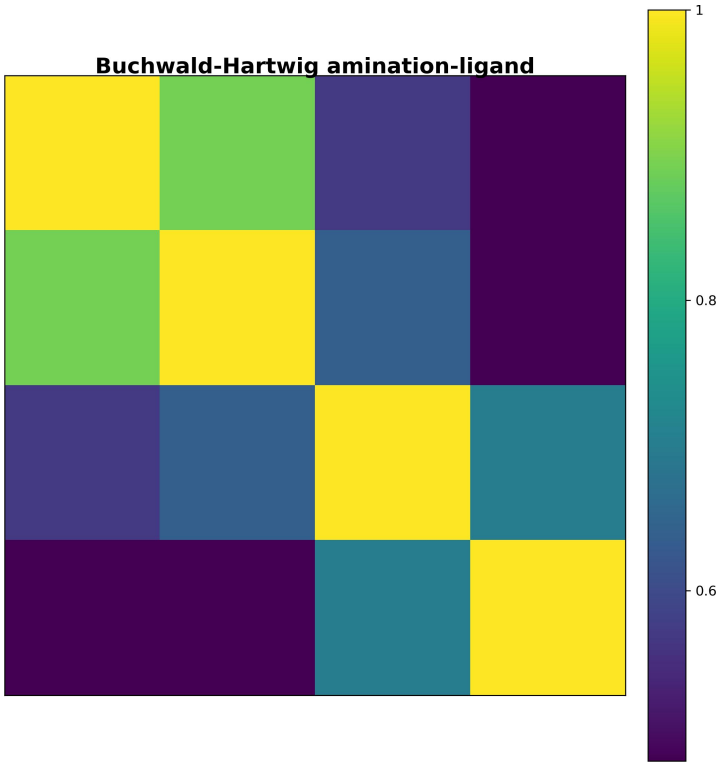


Fig .S7 Similarity score matrix of ligands in BHA.

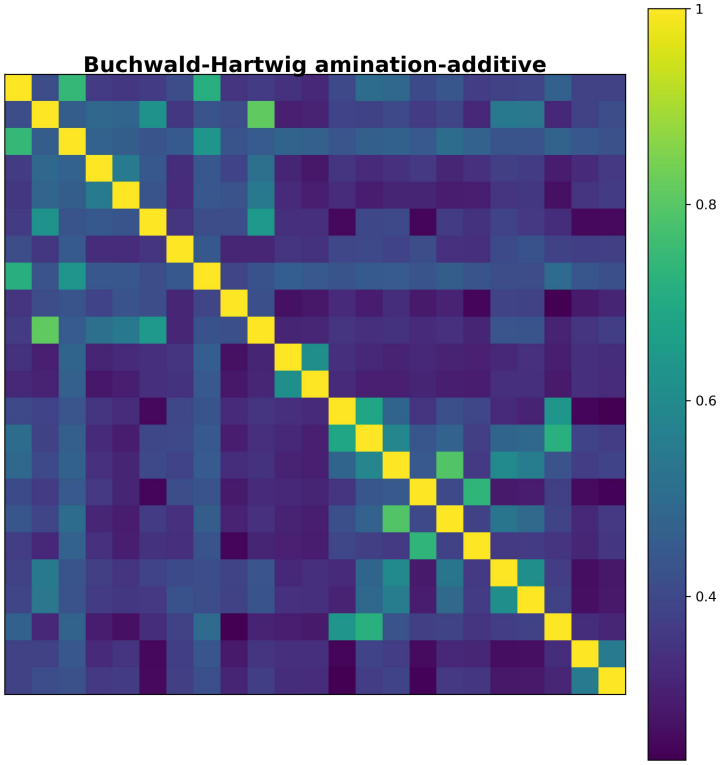


Fig .S8 Similarity score matrix of additives in BHA.

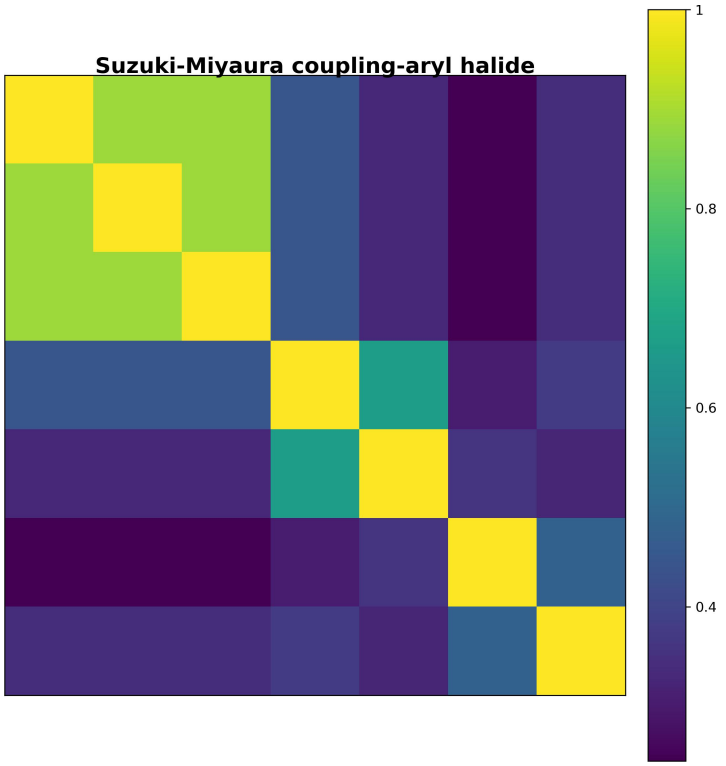


Fig .S9 Similarity score matrix of aryl halides in SMC.

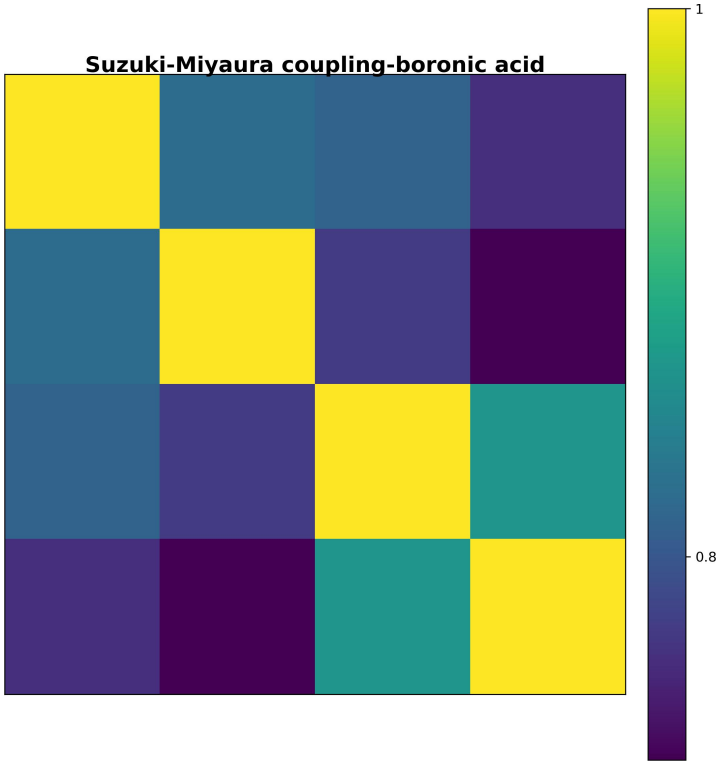


Fig .S10 Similarity score matrix of boronic acids in SMC.

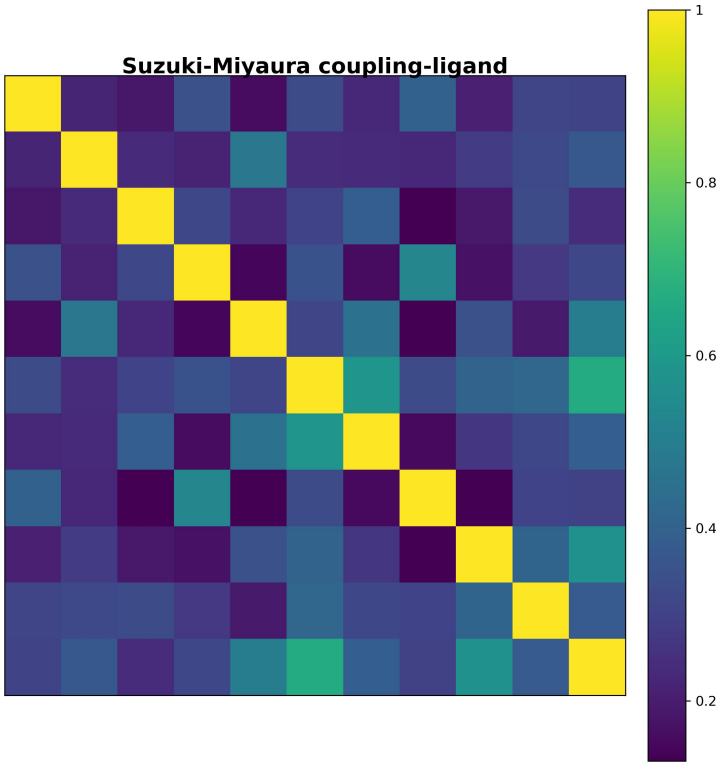


Fig .S11 Similarity score matrix of ligands in SMC.

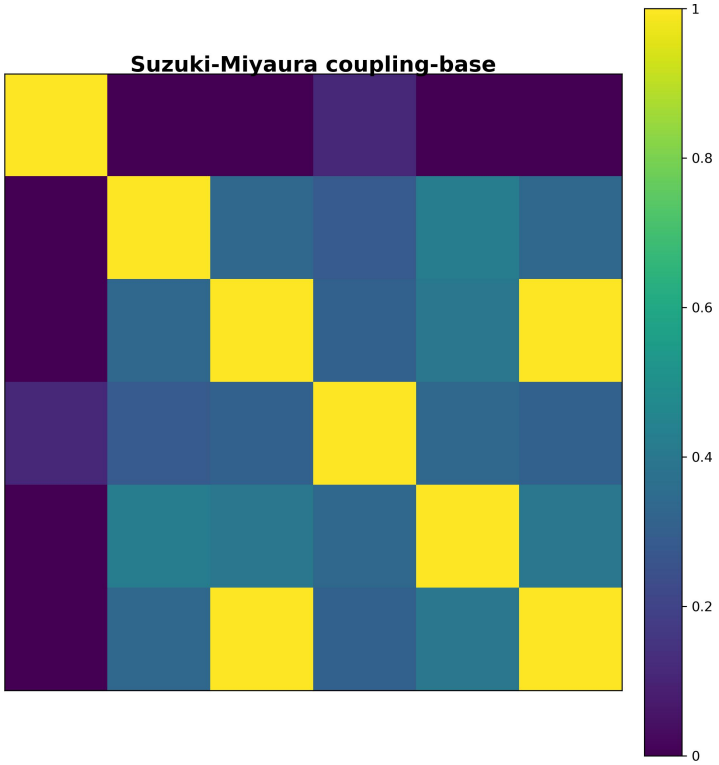


Fig .S12 Similarity score matrix of bases in SMC.

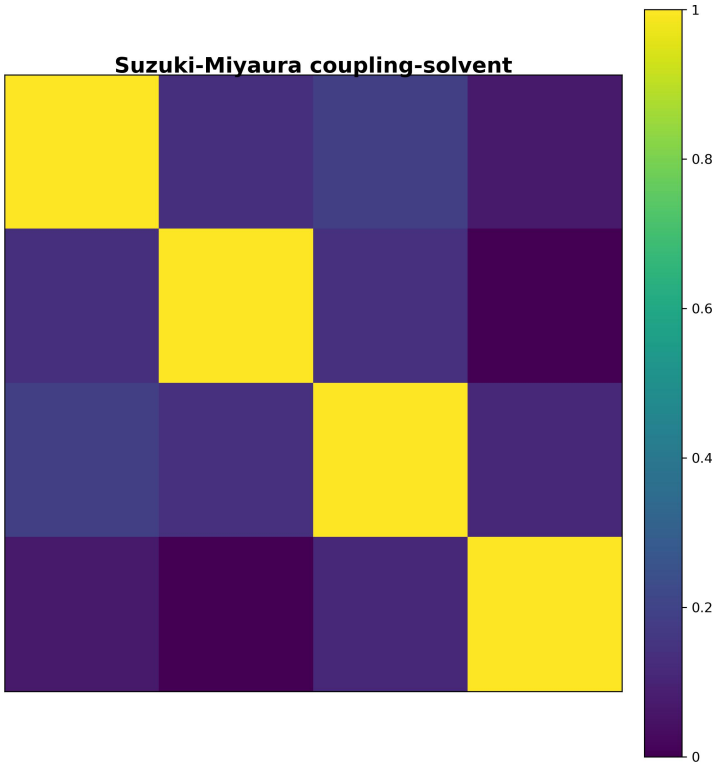


Fig .S13 Similarity score matrix of solvents in SMC.

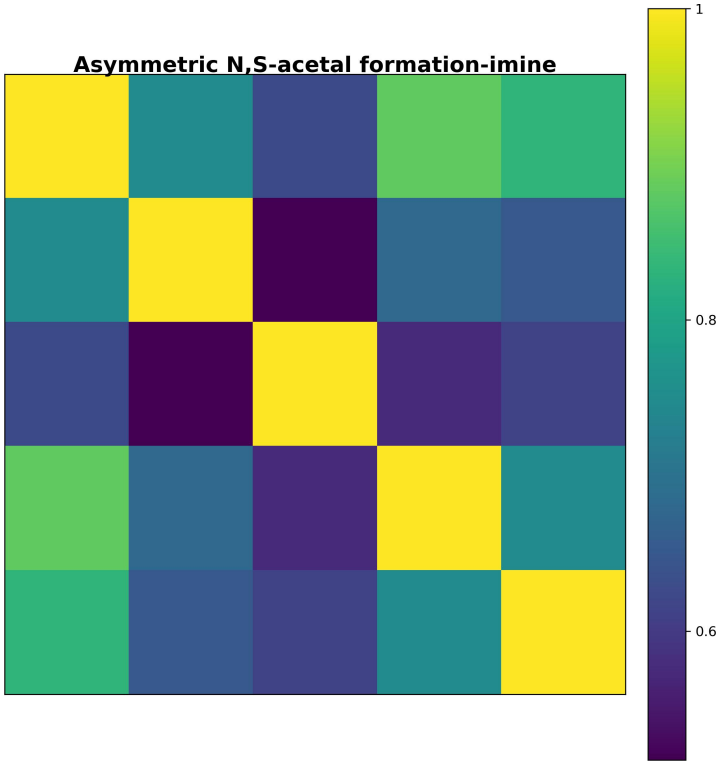


Fig .S14 Similarity score matrix of imines in ANSA.

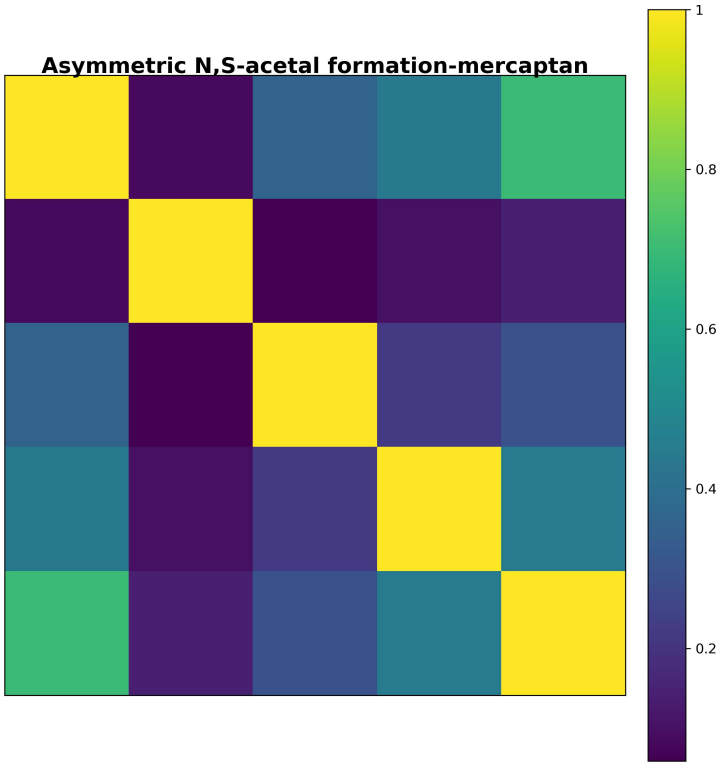


Fig .S15 Similarity score matrix of mercaptans in ANSA.

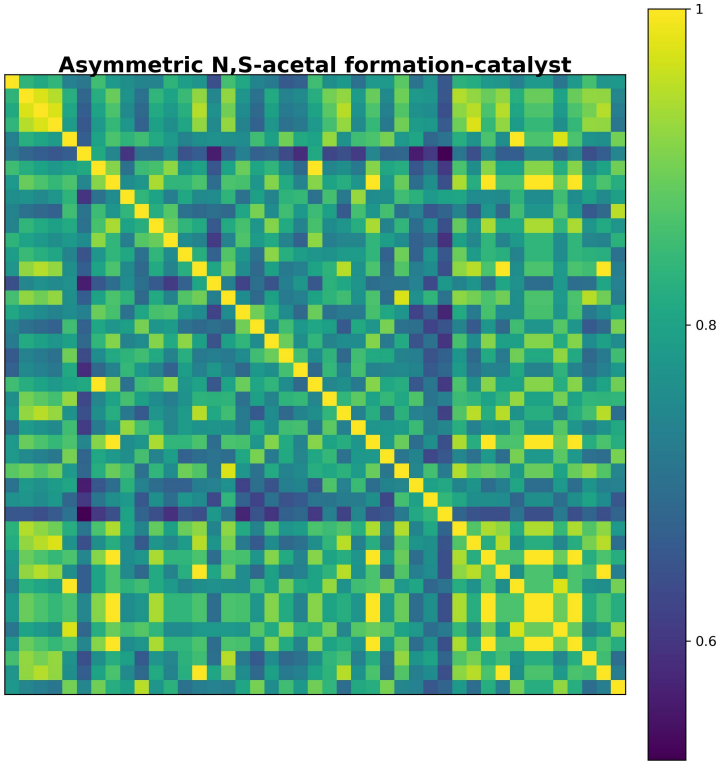


Fig .S16 Similarity score matrix of catalysts in ANSA.

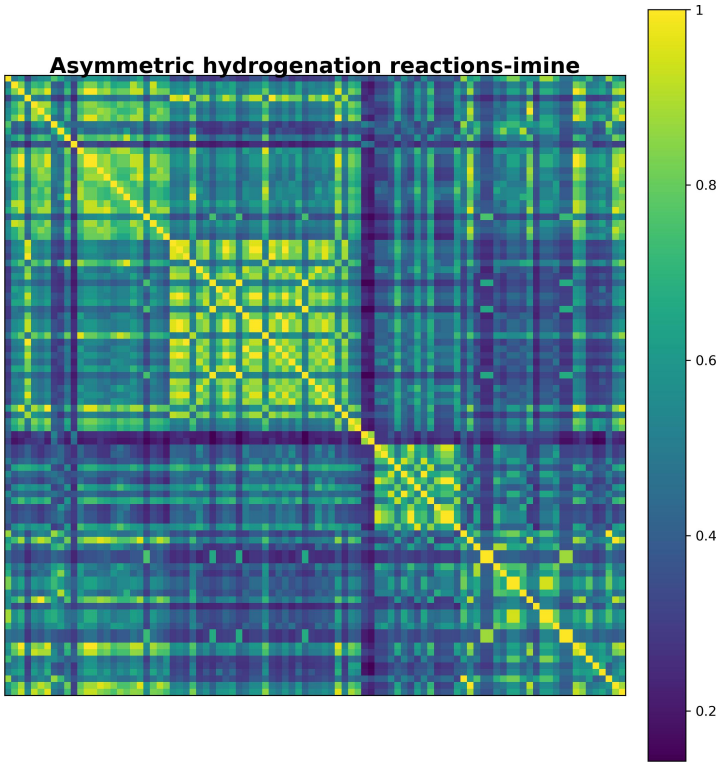


Fig .S17 Similarity score matrix of imines in AHR.

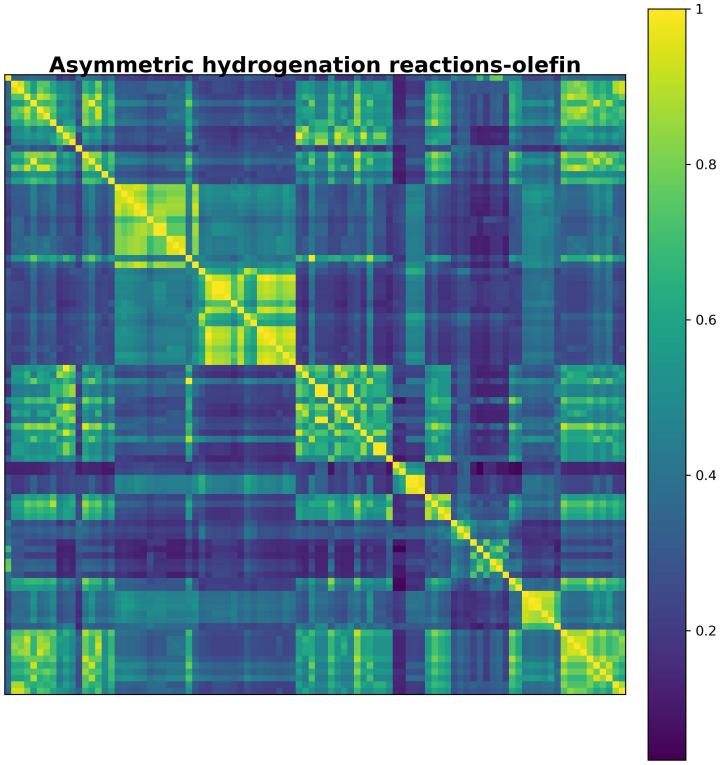


Fig .S18 Similarity score matrix of olefins in AHR.

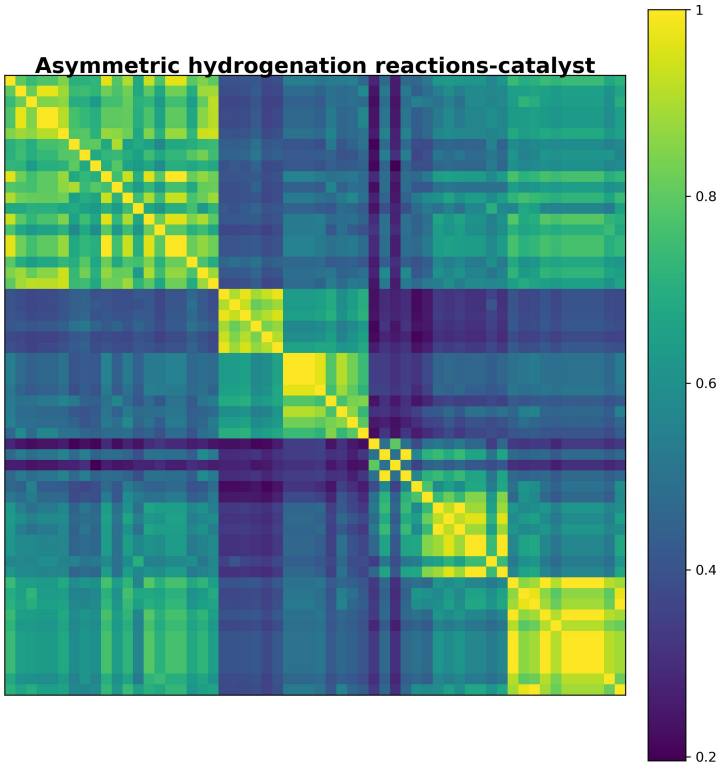


Fig .S19 Similarity score matrix of catalysts in AHR.

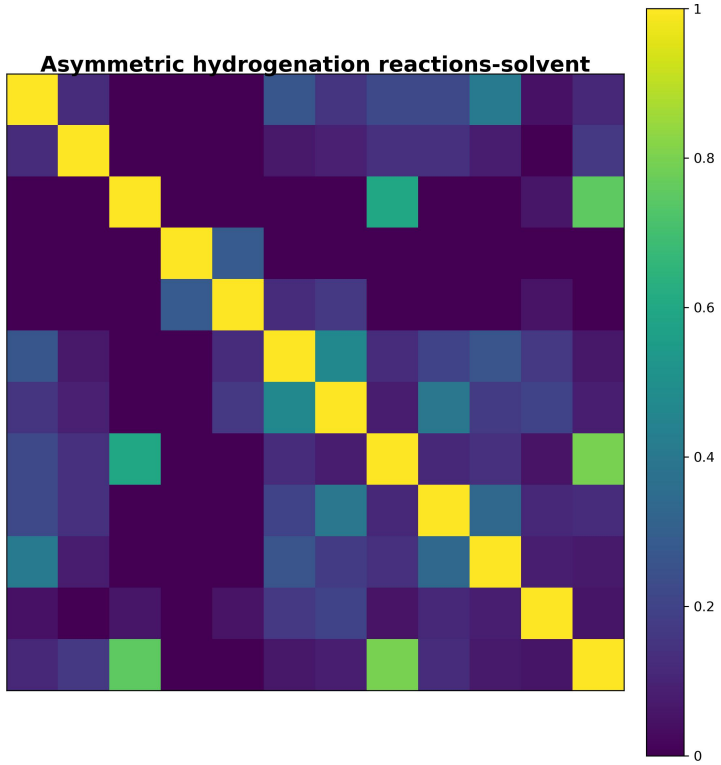


Fig .S20 Similarity score matrix of solvents in AHR.

Supplementary Note 2: Calculation of descriptors

2.1 Computational Methods

Full molecule structures were optimized with density functional B3LYP [6], and the 6-311G* basis set[7]. Descriptors were calculated from these structures using density functional m062x[8] and def2tzvp basis set[9].

Table .S1 Descriptors used for ML models

No.	Description	Source	atom level or molecule level
1	Atom type	Gaussian 09	atom level
2	Coordinate of the optimized structure	Gaussian 09	atom level
3	Mulliken charges of the optimized structure	Gaussian 09	atom level
4	NPA charges of the optimized structure	Gaussian 09	atom level
5	NMR shielding of the optimized structure	Gaussian 09	atom level
6	HOMO of the optimized structure	Gaussian 09	molecule level
7	LUMO of the optimized structure	Gaussian 09	molecule level
8	Zero-point energy of the optimized structure	Gaussian 09	molecule level
9	Volumn of the optimized structure	Gaussian 09	molecule level
10	Dipole of the optimized structure	Gaussian 09	molecule level

2.2 Three-dimensional information reconstruction for machine learning

Unlike existing methods that use key substructures in molecules to describe molecular structure characteristics, we use atomic-level descriptors to fully extract chemical information based on 3d structures required for reactions. Our idea in extracting chemical information is that the 3D structure of molecules can be determined by the order of atoms and atomic coordinates, while other atomic features such as NPA charge, NMR Shielding, etc. are extracted in the same order as atoms. Since this transformation from 3D space to 2D vector is based on linear structure formula, it can ensure that the chemical features correspond to the atom to which it belongs.

We get linear structure formula of compounds from Reaxys database, extract the chemical symbols in sequence, and convert the chemical symbols into numeric identifiers to produce atomic sequential features. We used Guassian 09 to optimize the structure of a molecule, and the resulting 3D molecular coordinates were also arranged in the atomic order of the linear structure formula, producing 3D coordinate features for the atomic descriptor. The properties of atoms such as NPA charge and NMR shielding are also treated in the same way. Finally, all these atomic level features are normalized.

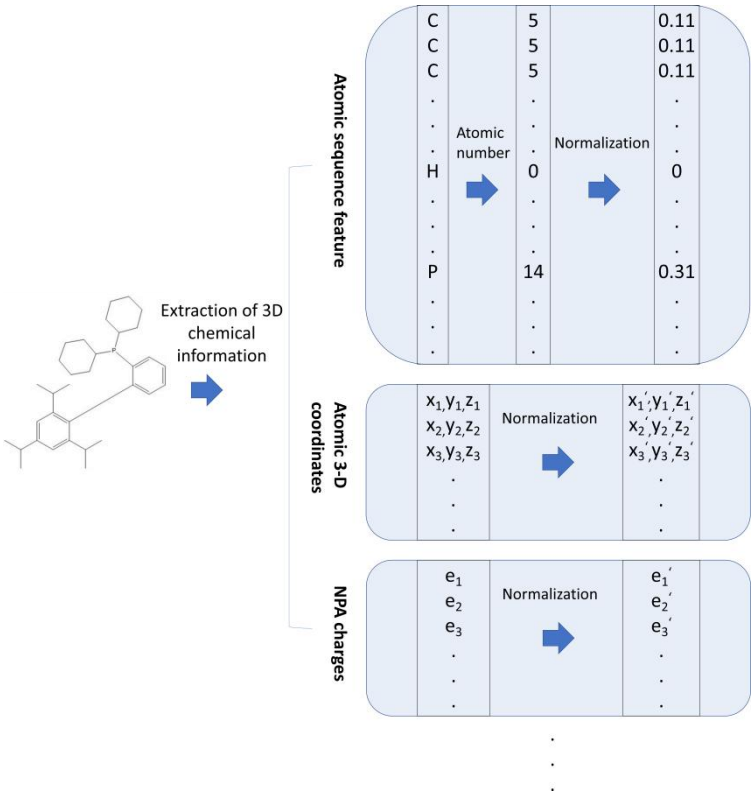


Fig. S21 General representation for three-dimensional information reconstruction using Atomic level features. The linear structure formula for the substance XPhos is $(C_6H_{11})_2P(C_6H_4C_6H_2(CH(CH_3)_2)_3)$, and the extraction of atomic features is carried out in this order.

Supplementary Note 3: Model generation

3.1 Data set split

5-fold cross-validation splits the data into 5 groups. Each of these groups are test set in turn, utilising the remaining data as training. Average metrics can then be calculated. K-fold cross validation is good for datasets as the data is used efficiently (every data point is part of test set at some point) and avoids chance biases in a static train/test split.

3.2 Machine learning hyperparameters

All regression models and classification models except neural networks were implemented with Python scripts using SciKitLearn[10]. Neural networks were built with Pytorch[11].

Table .S2 Parameters of ML models (except neural networks)

	Models	Parameters
Regression	AdaBoost	random_state=42, n_estimators=100
	LinearRegression	default
	Support Vector Machine	max_iter=10000
	K-Nearest Neighbors	n_neighbors=7
	Random Forest	n_estimators=100, random_state=42
	Gradient Boosting	default
Classification	AdaBoost	random_state=42, n_estimators=100
	K-Nearest Neighbors	n_neighbors=7
	LinearRegression	penalty='l2'

For regression analysis, all neural networks used the 'SGD' optimizer and sigmoid activation function, had a learning rate of 0.02, batch size of training data size and weight decay of 0.01. In BHA, SMC, ANSA, the neural networks had two layers with 1000, 500 nodes, respectively, and batch normalization. In AHR, the neural networks had two layers with 200, 50 nodes, respectively, and batch normalization. When leaving out one molecule, all neural networks had two layers with 0.5 and 0.5 dropouts.

For classification analysis, the neural network used the 'SGD' optimizer, sigmoid activation function and 'BCEWithLogits' loss function, had two layers with 200, 50 nodes, respectively, and batch normalization.

3.3 Metrics

When predicting yields and enantiomeric excess, we used R^2 and RMSE to evaluate machine learning models. R^2 and RMSE are defined in equations (1) and (2), where n is the number of samples, $\widehat{y_i}$ is the predicted value of y_i, $\bar{y}$ is the mean value of y.

$$R^2 = 1 - \left(\frac{\sum_{i=1}^{n}(\widehat{y_i}-y_i)^2}{\sum_{i=1}^{n}(y_i-\bar{y})^2}\right) \quad (1)$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\widehat{y_i}-y_i)^2} \quad (2)$$

When predicting the *R*/*S* configurations in AHR, we use accuracy to evaluate machine

learning models. Accuracy is defined in equations (3), where TP is the number of true positives, TN is the number of true negatives, P is the number of positives and N is the number of negatives.

$$\mathrm{Acc} = \frac{TP+TN}{P+N} \tag{3}$$

3.4 Comparison of different descriptors on sparse data sets

AHR is a typical sparse reaction data set: it has 260 compounds, but only 334 reactions. The number of reactions is so small that the breadth of the feature space of the chemical reaction is not spanned. We found that short molecular-level descriptors perform better in small, simple chemical spaces, while information-rich long descriptors such as atomic-level descriptors are better at handling complex data sets. The comparison of different types of descriptors on this sparse reaction data set is still to be expected.

When only the molecular level descriptor (MLD) is used (Fig. 7), most models have low performance, and the short descriptor's low dimensionality has the most severe impact on SVM performance, with R^2 0.19. But we can see that as model complexity and presentation increase, so does performance. The contrast is obvious when using atomic level descriptor (ALD) (Fig. 8), the performance of these models has improved significantly, indicating that this complete chemical information expression based on 3D structural information is useful for sparse chemical reaction data sets. As a combination of the two descriptors above, MALD achieves optimal performance (Fig. 9). The Neural Network model using this descriptor achieves a staggering 0.99 R^2. It can be seen that on sparse reaction datasets, these two levels of descriptors can complement each other. It should not be overlooked that the SMOTE algorithm also plays an important role in this data set.

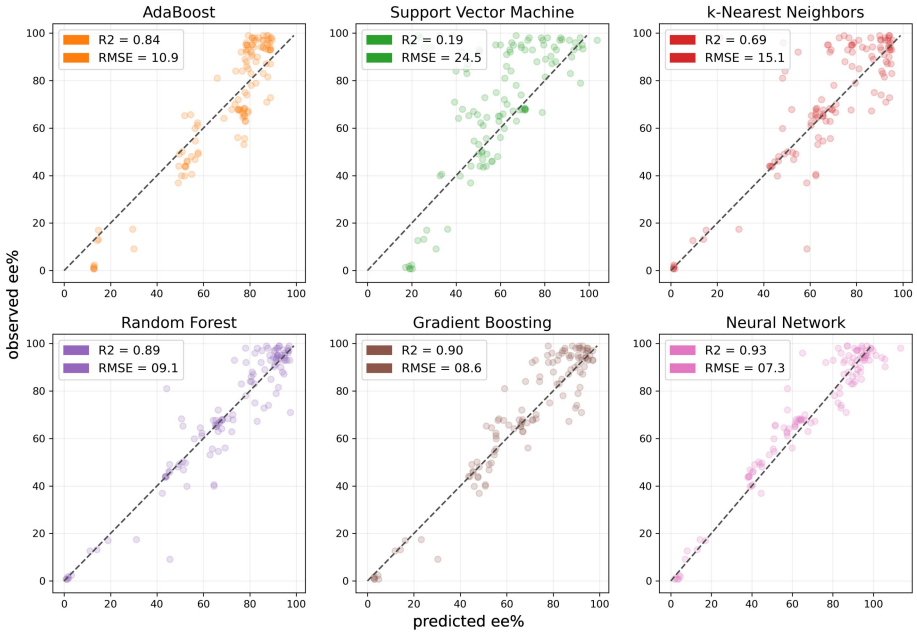


Fig. S22 Prediction results with MLD on AHR data set (5-fold cross-validation).

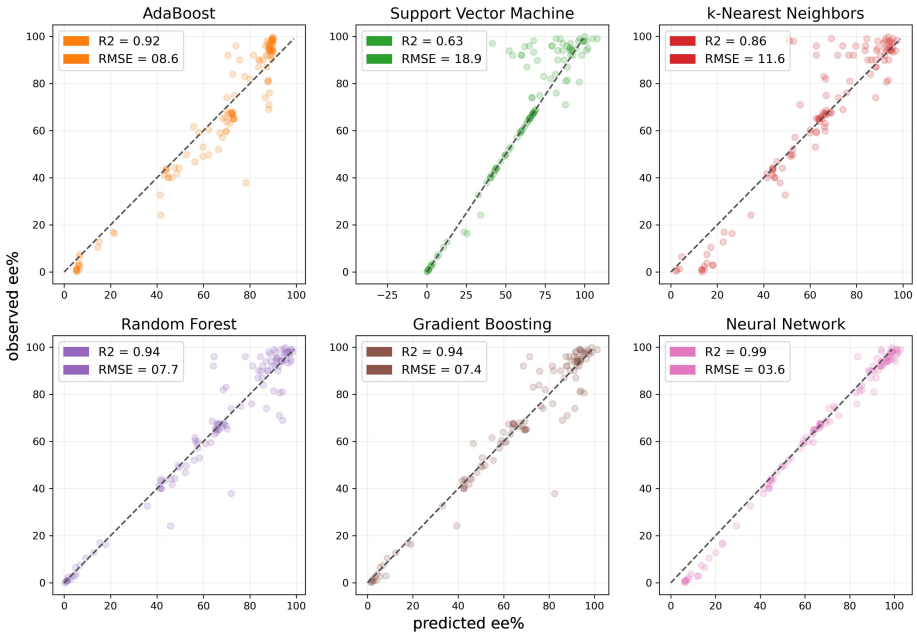


Fig. S23 Prediction results with ALD on AHR data set (5-fold cross-validation).

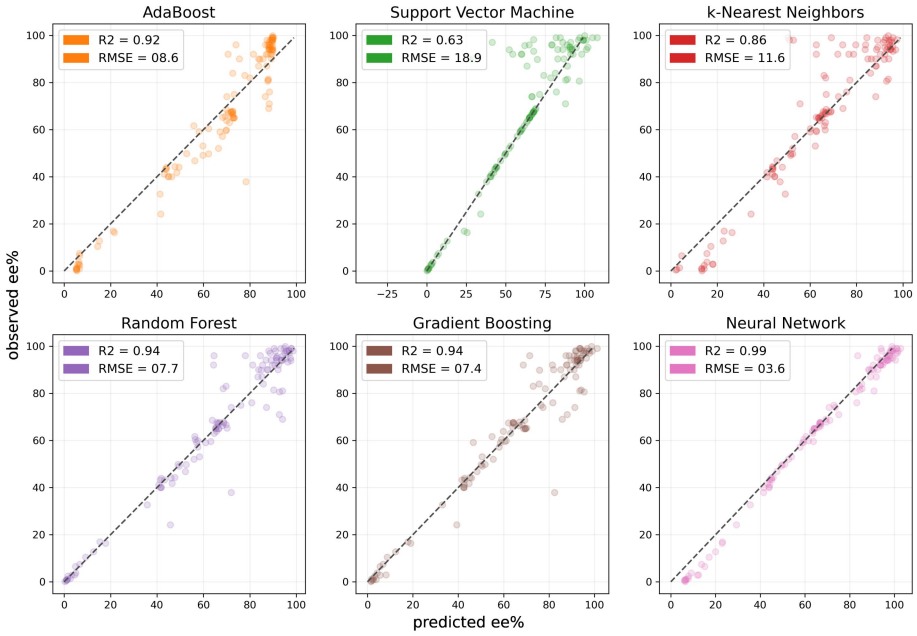


Fig. S24 Prediction results with MALD on AHR data set (5-fold cross-validation).

3.5 Leave one molecule out prediction (LOMOP)

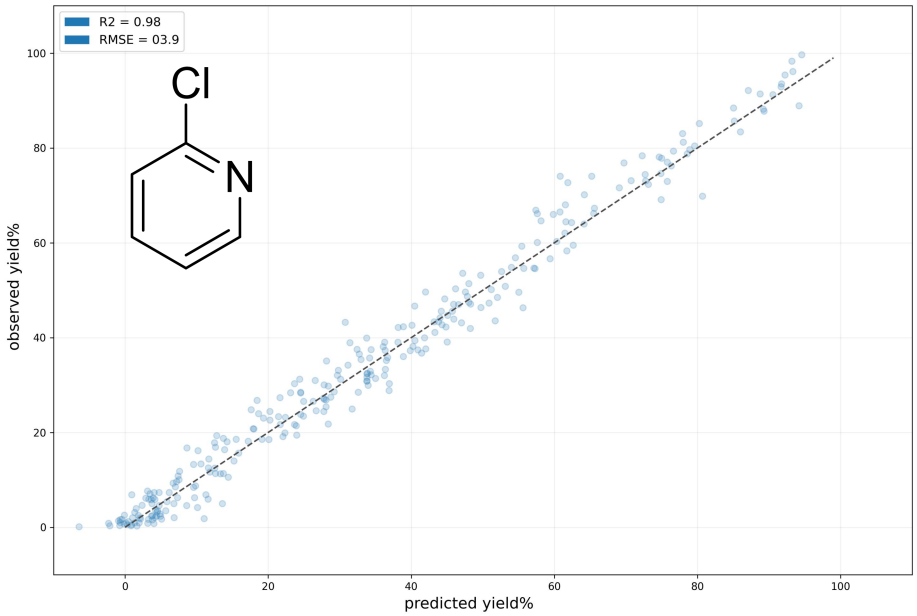


Fig .25 The best LOMOP in aryl halide category in BHA.

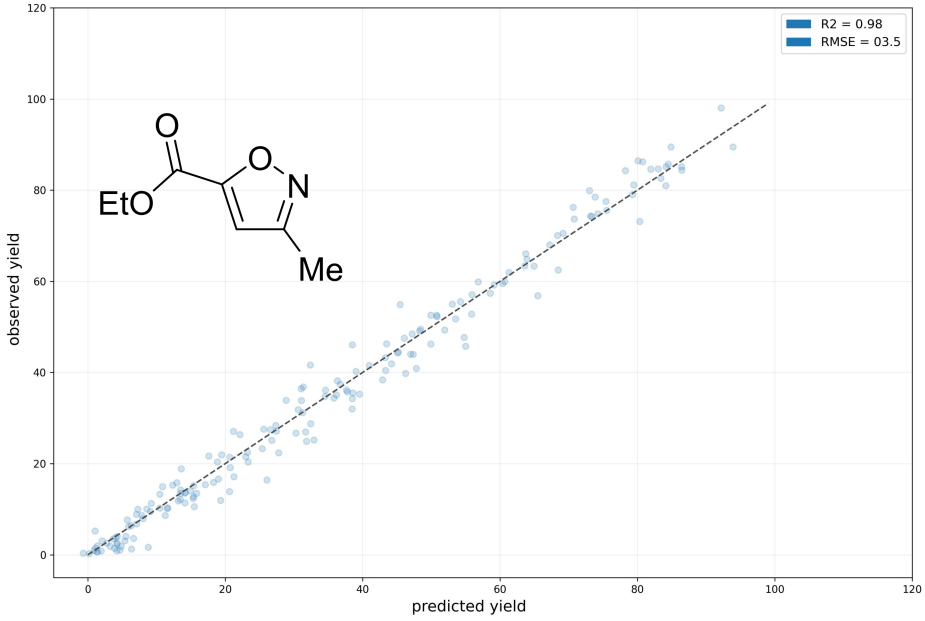


Fig .S26 The best LOMOP in additive category in BHA.

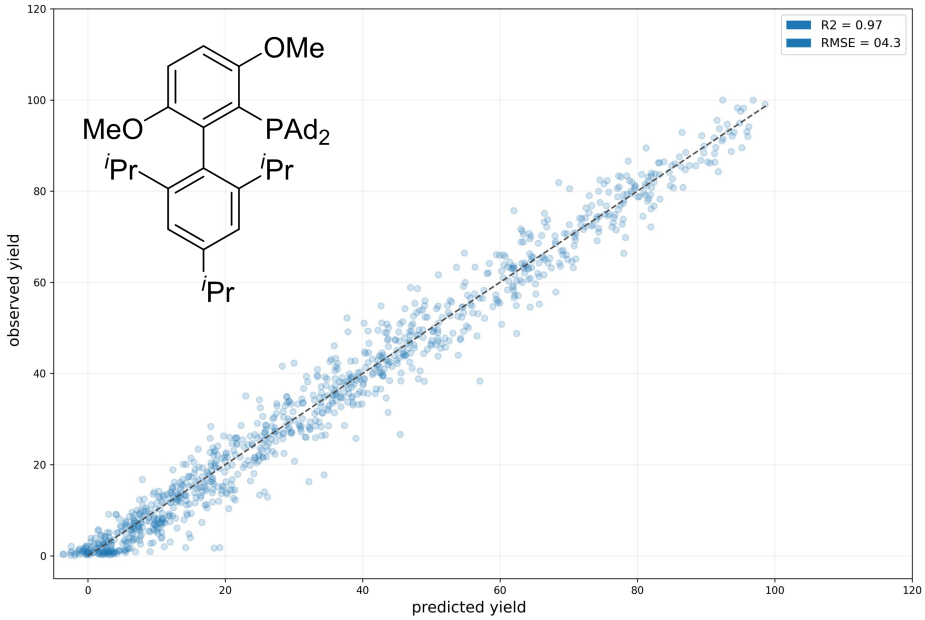


Fig .S27 The best LOMOP in ligand category in BHA.

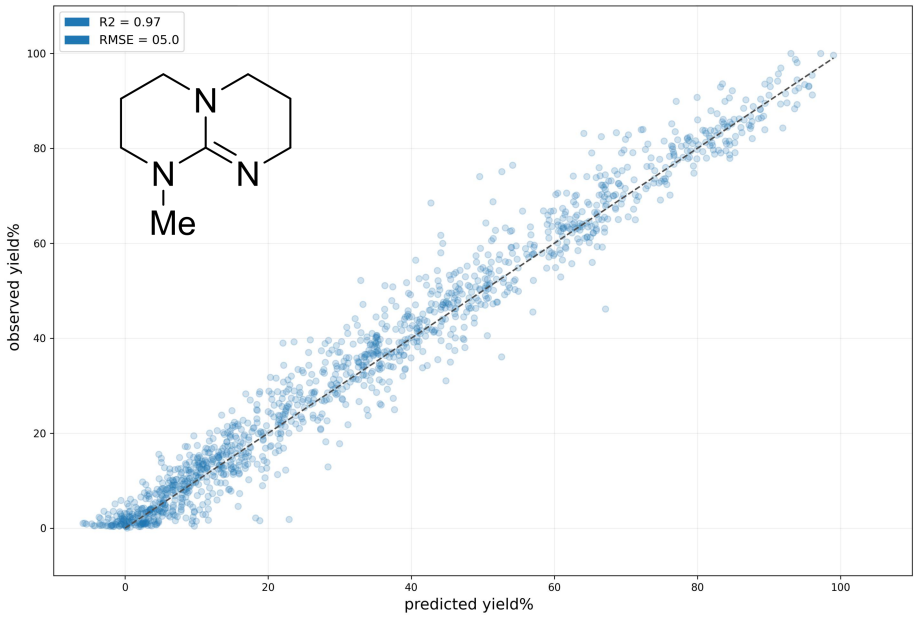


Fig .S28 The best LOMOP in base category in BHA.

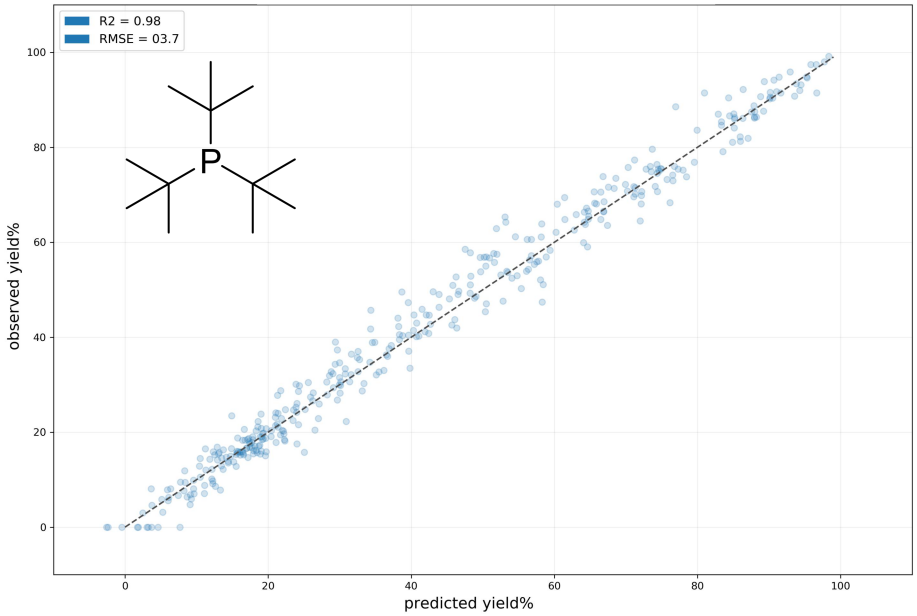


Fig .S29 The best LOMOP in ligand category in SMC.

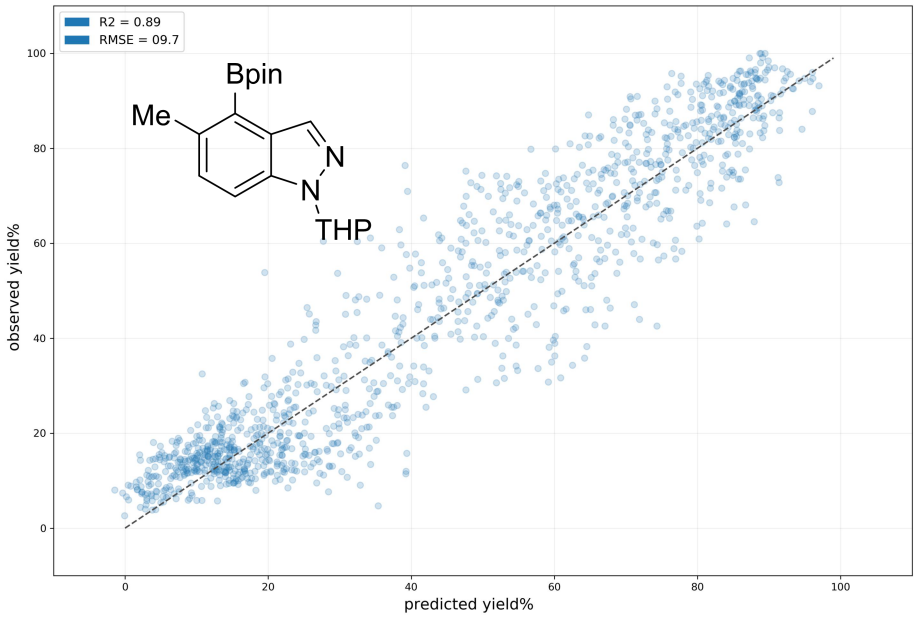


Fig .S30 The best LOMOP in boronic acid category in SMC.

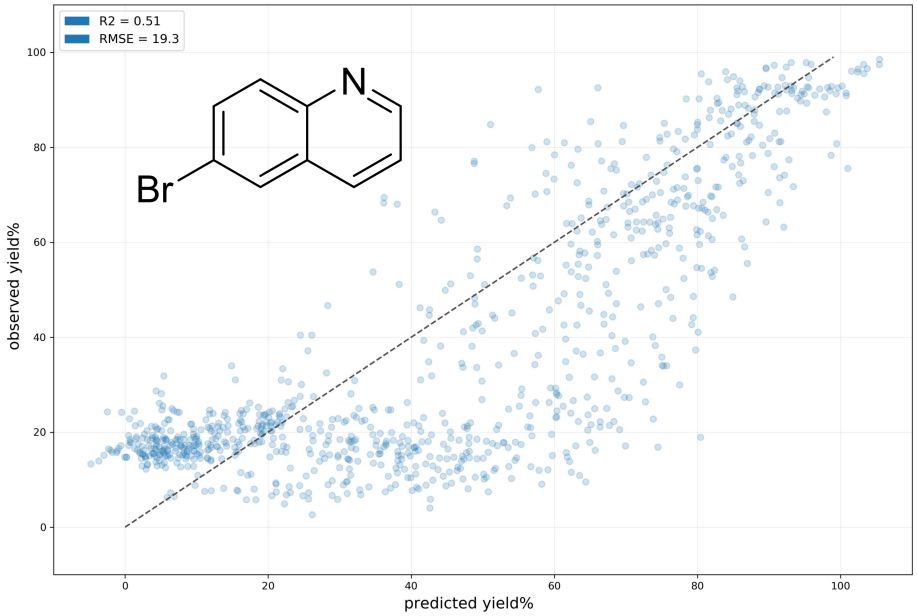


Fig .S30 The best LOMOP in aryl halide category in SMC.

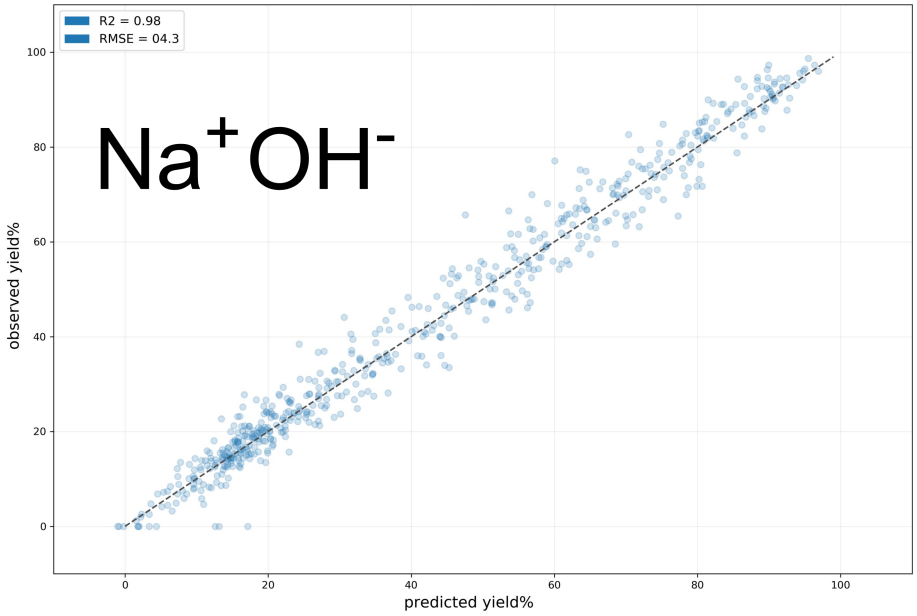


Fig .S31 The best LOMOP in base category in SMC.

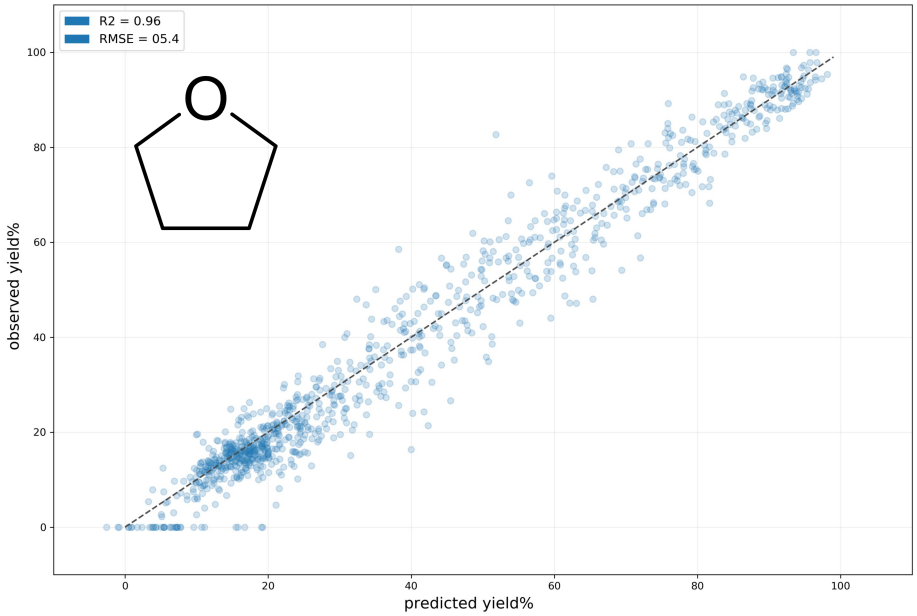


Fig .S32 The best LOMOP in solvent category in SMC.

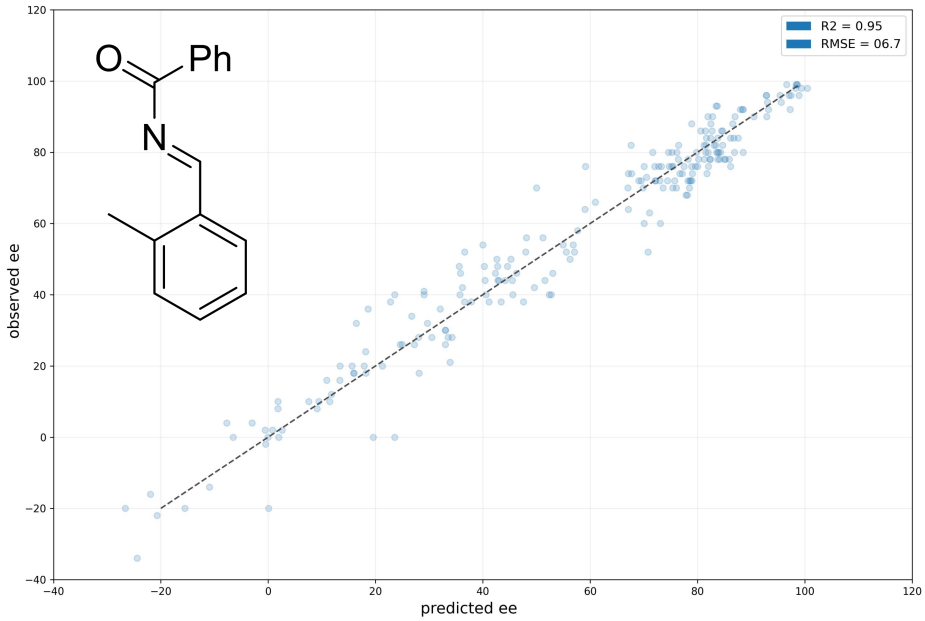


Fig .S33 The best LOMOP in imine category in ANSA.

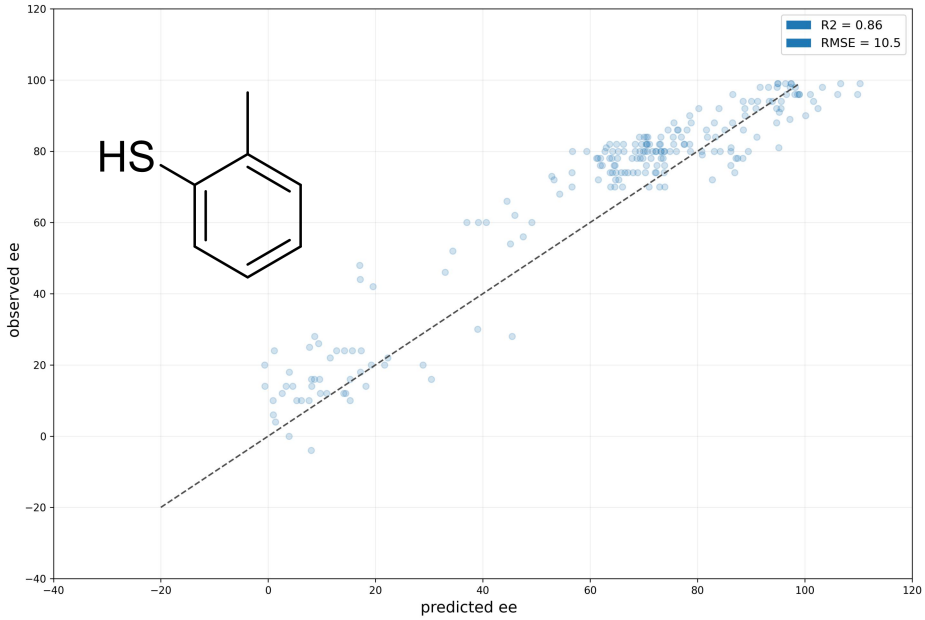


Fig .S34 The best LOMOP in mercaptan category in ANSA.

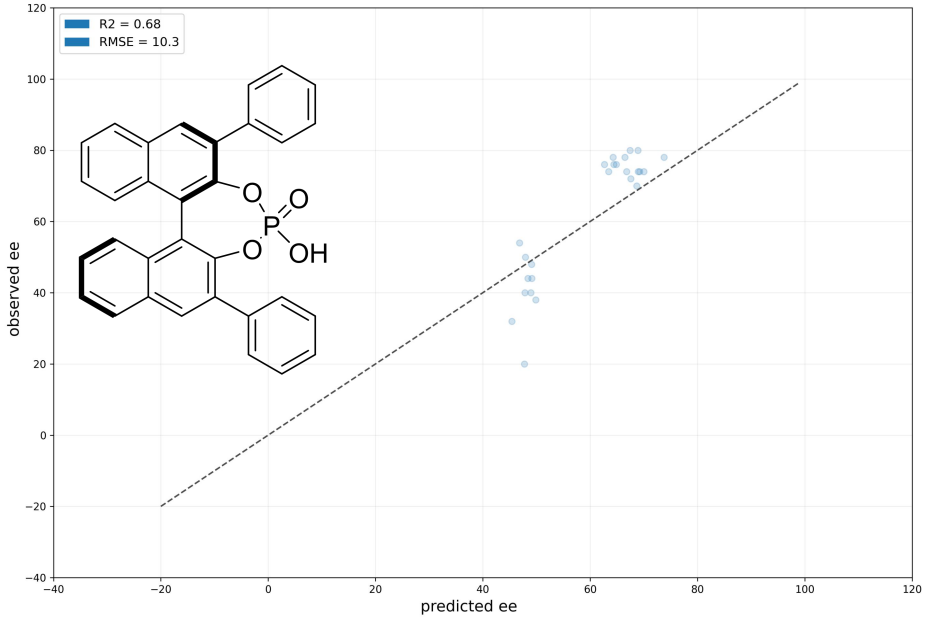


Fig .S35 The best LOMOP in catalyst category in ANSA.

3.6 Detailed Description of Our Machine Learning Methods

Next, we introduce how to use scikit-learn and pytorch code for various ML algorithms in this study. We will focus on demonstrating how to use, modify, and run code, with emphasis on key lines. Our code uses CUDA so that it can use GPU, which reduces computing time. Here, we take the neural network as a representative example.

We have installed eclipse IDE 2021, anaconda 3, python 3.6, python 1.1 and scikit-learn in Ubuntu system, and there is an NVIDIA 1070 graphics card on the computer. We configured the virtual environment built in Anaconda in eclipse, and then created a python type project to write the code of this study.

In the following script, we loaded the reaction data, created the Neural Network model, and completed the training and verification. We highlighted the key operations.

```
#!/usr/bin/env python
# coding: utf-8

import numpy as np
from matplotlib import pyplot as plt
import matplotlib.patches as mpatches
import pandas as pd
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from torch.autograd import Variable
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.svm import LinearSVR
import time
import os
from pandas.core.indexes.numeric import Int64Index
import suzuki18.lib_for_suzuki as suzuki
from torch.nn.modules import padding
import random
use_gpu = torch.cuda.is_available()

DATA_DIR = 'data/'
time_s = time.strftime("%Y%m%d%H%M%S", time.localtime())
OUT_DIR = 'out/models'+time_s+'/'
if not os.path.exists(OUT_DIR):
    os.mkdir(OUT_DIR)
INPUTS_HOMO_Energy = 'HOMO_Energy.csv'
```

```
INPUTS_LUMO_Energy = 'LUMO_Energy.csv'
INPUTS_Dipole = 'Dipole.csv'
INPUTS_Volumn = 'Volumn.csv'
INPUTS_Single_Energy = 'Single_Energy.csv'
INPUTS_Atom = 'Atom.csv'
INPUTS_Coord = 'Coord.csv'
INPUTS_Mulliken_charges = 'Mulliken_charges.csv'
INPUTS_NAO_charge = 'NAO_charge.csv'
INPUTS_NMR_Shielding = 'NMR_Shielding.csv'
YIELDS_DF = 'reac_yields.csv'

inputs_HOMO_Energy = pd.read_csv(DATA_DIR + INPUTS_HOMO_Energy)
inputs_HOMO_Energy = inputs_HOMO_Energy.fillna(0.)
inputs_LUMO_Energy = pd.read_csv(DATA_DIR + INPUTS_LUMO_Energy)
inputs_LUMO_Energy = inputs_LUMO_Energy.fillna(0.)
inputs_Dipole = pd.read_csv(DATA_DIR + INPUTS_Dipole)
inputs_Dipole = inputs_Dipole.fillna(0.)
inputs_Volumn = pd.read_csv(DATA_DIR + INPUTS_Volumn)
inputs_Volumn = inputs_Volumn.fillna(0.)
inputs_Single_Energy = pd.read_csv(DATA_DIR + INPUTS_Single_Energy)
inputs_Single_Energy = inputs_Single_Energy.fillna(0.)

inputs_Atom = pd.read_csv(DATA_DIR + INPUTS_Atom)[idex_thres]
inputs_Coord = pd.read_csv(DATA_DIR + INPUTS_Coord)[idex_thres]
inputs_Mulliken_charges = pd.read_csv(DATA_DIR + INPUTS_Mulliken_charges)[idex_thres]
inputs_NAO_charge = pd.read_csv(DATA_DIR + INPUTS_NAO_charge)[idex_thres]
inputs_NMR_Shielding = pd.read_csv(DATA_DIR + INPUTS_NMR_Shielding)[idex_thres]
inputs_occupy = pd.read_csv(DATA_DIR + INPUTS_OCCUPY)[idex_thres]
yields = pd.read_csv(DATA_DIR + YIELDS_DF)[idex_thres]

total_cleaned_len = len(inputs_HOMO_Energy)
train_index = random.sample(range(total_cleaned_len), int(total_cleaned_len*0.8))
train_indices = np.zeros((total_cleaned_len)).astype(np.bool)
for tcl in range(total_cleaned_len):
    if tcl in train_index:
        train_indices[tcl] = True
test_indices = ~ train_indices

np.savetxt(OUT_DIR + 'data_train_indices.csv', train_indices, delimiter = ',')
np.savetxt(OUT_DIR + 'data_test_indices.csv', test_indices, delimiter = ',')

input_atomlevel_dim = inputs_HOMO_Energy.shape[1] + inputs_LUMO_Energy.shape[1] +
inputs_Dipole.shape[1] + inputs_Volumn.shape[1] + inputs_Single_Energy.shape[1]#25
output_dim = 1
```

```
inputs_HOMO_Energy = suzuki.scale_inputs(inputs_HOMO_Energy)
inputs_LUMO_Energy = suzuki.scale_inputs(inputs_LUMO_Energy)
inputs_Dipole = suzuki.scale_inputs(inputs_Dipole)
inputs_Volumn = suzuki.scale_inputs(inputs_Volumn)
inputs_Single_Energy = suzuki.scale_inputs(inputs_Single_Energy)

inputs_Atom = suzuki.scale_inputs(inputs_Atom)
inputs_Coord = suzuki.scale_inputs(inputs_Coord)
inputs_Mulliken_charges = suzuki.scale_inputs(inputs_Mulliken_charges)
inputs_NAO_charge = suzuki.scale_inputs(inputs_NAO_charge)
inputs_NMR_Shielding = suzuki.scale_inputs(inputs_NMR_Shielding)

inputs_occupy_dim = inputs_occupy.shape[1]
inputs = np.concatenate((inputs_HOMO_Energy,inputs_LUMO_Energy),axis=1)
inputs = np.concatenate((inputs,inputs_Dipole),axis=1)
inputs = np.concatenate((inputs,inputs_Volumn),axis=1)
inputs = np.concatenate((inputs,inputs_Single_Energy),axis=1)

if USE_OCCUPY_ATOMLEVEL == 0:
    inputs = np.concatenate((inputs,inputs_occupy),axis=1)
    ann_input_dim = inputs.shape[1]
elif USE_OCCUPY_ATOMLEVEL == 1:
    inputs = np.concatenate((inputs,inputs_Atom),axis=1)
    inputs = np.concatenate((inputs,inputs_Coord),axis=1)
    inputs = np.concatenate((inputs,inputs_Mulliken_charges),axis=1)
    inputs = np.concatenate((inputs,inputs_NAO_charge),axis=1)
    inputs = np.concatenate((inputs,inputs_NMR_Shielding),axis=1)

yields = np.array(yields['x'])
yields = yields.flatten()
yields = np.nan_to_num(yields, nan=0)
inputs = inputs[:,15:]
input_dim = inputs.shape[1]
print('input_dim: ', input_dim)
# Use the indices to generate train/test sets
X_train = inputs[train_indices]
y_train = yields[train_indices]
featuresTrain = torch.from_numpy(X_train)
targetsTrain = torch.from_numpy(y_train)

X_test = inputs[test_indices]
y_test = yields[test_indices]
```

```
featuresTest = torch.from_numpy(X_test)
targetsTest = torch.from_numpy(y_test)

train = torch.utils.data.TensorDataset(featuresTrain,targetsTrain)
test = torch.utils.data.TensorDataset(featuresTest,targetsTest)

train_loader = torch.utils.data.DataLoader(train, batch_size = batch_size, shuffle = False)
test_loader = torch.utils.data.DataLoader(test, batch_size = batch_size_test, shuffle = False)

class NN_Chem(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(NN_Chem, self).__init__()
        self.fc_1 = nn.Linear(input_dim, 1000)
        self.bn_1 = nn.BatchNorm1d(1000)
        # Non-linearity 1
        self.relu_1 = nn.Sigmoid()
        self.fc_2 = nn.Linear(1000, 500)
        self.bn_2 = nn.BatchNorm1d(500)
        self.relu_2 = nn.Sigmoid()
        self.fc_3 = nn.Linear(500, output_dim)

    def forward(self, x):
        # Linear function 1
        out = self.fc_1(x)
        out = self.bn_1(out)
        # Non-linearity 1
        out = self.relu_1(out)
        out = self.fc_2(out)
        out = self.bn_2(out)
        out = self.relu_2(out)
        out = self.fc_3(out)
        return out

# Create NN
model = NN_Chem(input_dim, output_dim)

if use_gpu:
    model = model.cuda()

def squared_loss(y_true, y_pred):
    """Compute the squared loss for regression.

    Parameters
    ----------
```

```
    y_true : array-like or label indicator matrix
        Ground truth (correct) values.

    y_pred : array-like or label indicator matrix
        Predicted values, as returned by a regression estimator.

    Returns
    -------
    loss : float
        The degree to which the samples are correctly predicted.
    """
    return ((y_true - y_pred) ** 2).mean() / 2

# SGD Optimizer
learning_rate = 0.02
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, weight_decay=0.01)

# NN model training
count = 0
loss_list = []
iteration_list = []
rmse_list = []
best_rmse = 10000.#
for epoch in range(num_epochs):
    for i, (features_train, labels) in enumerate(train_loader)

        if use_gpu:
            features_train = features_train.cuda()
            labels = labels.cuda()
        features_train = Variable(features_train.to(torch.float))
        labels = Variable(labels.to(torch.float))

        # Clear gradients
        optimizer.zero_grad()
        # Forward propagation
        outputs = model(features_train)

        outputs = outputs.squeeze(-1)
        # Calculate softmax and ross entropy loss
        #loss = error(outputs, labels)
        loss = squared_loss(outputs, labels)

        # Calculating gradients
        loss.backward()
```

```
        # Update parameters
        optimizer.step()

        count += 1
        if count % 50 == 0:
            # Calculate Accuracy
            correct = 0
            total = 0
            rmse = 0
            # Predict test dataset
            for features_test, labels_test in test_loader:
                if use_gpu:
                    features_test = features_test.cuda()
                    labels_test = labels_test.cuda()
            features_test = Variable(features_test.to(torch.float))
            # Forward propagation
            outputs_test = model(features_test)
                total += 1
                rmse  +=  mean_squared_error((labels_test.to(torch.float)).data.cpu(),  outputs_test.data.cpu())
** 0.5

            rmse = rmse/float(total)

            if rmse < best_rmse:
                best_rmse = rmse
                for f in os.listdir(OUT_DIR):
                    if(os.path.isfile(OUT_DIR+f)) and 'best_model' in f:
                        os.remove(OUT_DIR+f)
                time_str = time.strftime("%Y%m%d%H%M%S", time.localtime())
                torch.save(model, OUT_DIR+'best_model_epoch_'+str(count-1)+'_'+time_str+'.pkl')
                r_squared = r2_score((labels_test.to(torch.float)).data.cpu(), outputs_test.data.cpu())
                predictions = []
                r2_values = []
                rmse_values = []
                r2_values.append(r_squared)
                rmse_values.append(rmse)
                predictions.append(outputs_test.data.cpu().squeeze(-1))
                fig_title = OUT_DIR+'model_epoch_'+str(count-1)+'_'+time_str
                utils.plot_models(predictions,
                  r2_values,
                  rmse_values,
                  (labels_test.to(torch.float)).data.cpu(),
                  save=True,
```

```
            fig_title = fig_title)

        loss_list.append(loss.data.cpu())
        iteration_list.append(count)
        rmse_list.append(rmse)
        if count % 500 == 0:
            # Print Loss
            print('Iteration: {}  Loss: {} ,  Rmse: {}'.format(count, loss.data.cpu(), rmse))
            time_str = time.strftime("%Y%m%d%H%M%S", time.localtime())
            torch.save(model, OUT_DIR+'model_epoch_'+str(count-1)+'_'+time_str+'.pkl')
```

References

1. Ahneman, D. T., Estrada, J. G., Lin, S., Dreher, S. D. & Doyle, A. G. Predicting reaction performance in C–N cross-coupling using machine learning. *Science* **360**, 186–190 (2018).
2. Perera, D. *et al.* A platform for automated nanomole-scale reaction screening and micromole-scale synthesis in flow. *Science* **359**, 429–434 (2018).
3. Zahrt, A. F. *et al.* Prediction of higher-selectivity catalysts by computer-driven workflow and machine learning. *Science* **363**, eaau5631 (2019).
4. Singh, S. *et al.* A unified machine-learning protocol for asymmetric catalysis as a proof of concept demonstration using asymmetric hydrogenation. *Proc Natl Acad Sci USA* **117**, 1339–1345 (2020).
5. Godden, J. W., Xue, L. & Bajorath, J. Combinatorial Preferences Affect Molecular Similarity/Diversity Calculations Using Binary Fingerprints and Tanimoto Coefficients. 4.
6. Yanai, T., Tew, D. P. & Handy, N. C. A new hybrid exchange–correlation functional using the Coulomb-attenuating method (CAM-B3LYP). *Chemical Physics Letters* **393**, 51–57 (2004).
7. Roy, L. E., Hay, P. J. & Martin, R. L. Revised Basis Sets for the LANL Effective Core

Potentials. *J. Chem. Theory Comput.* **4**, 1029–1031 (2008).

8. Hohenstein, E. G., Chill, S. T. & Sherrill, C. D. Assessment of the Performance of the M05−2X and M06−2X Exchange-Correlation Functionals for Noncovalent Interactions in Biomolecules. *J. Chem. Theory Comput.* **4**, 1996–2000 (2008).
9. Weigend, F. Accurate Coulomb-fitting basis sets for H to Rn. *Phys. Chem. Chem. Phys.* **8**, 1057 (2006).
10. scikit-learn. https://scikit-learn.org/stable/.
11. PyTorch. https://www.pytorch.org.
12. Gaussian 09, Revision D.01, M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, T. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, O. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski, and D. J. Fox, Gaussian, Inc., Wallingford CT, 2013.