Supporting Information for "High throughput discovery of ternary Cu-Fe-Ru alloy catalysts for photo-driven hydrogen production"

Maya Bhat, Zoe C. Simon, Savannah L. Talledo, Riti Sen, Jacob H. Smith, Stefan Bernhard, Jill E. Millstone, and John R. Kitchin

March 29, 2023

Contents

1 TEM Images and Nanoparticle Size Distributions

The sizes and size distributions measured by TEM varied in all two monometallic and four bi and trimetallic samples. Size distributions were generated from at least 200 nanoparticles per sample. In all cases, distinct nanoparticles were formed. After 15 hours of irradiation, the nanoparticles were imaged and sized as shown in Figure 1 - 5.

All samples analyzed by TEM contained compositions near, but not equal to, the most active composition. Though these compositions and concentrations varied slightly from the most active wells, particle size is not expected to be impacted by these small synthetic changes. The compositions and concentrations of the wells, as well as particle diameter and standard deviations, can be found in Table 1.

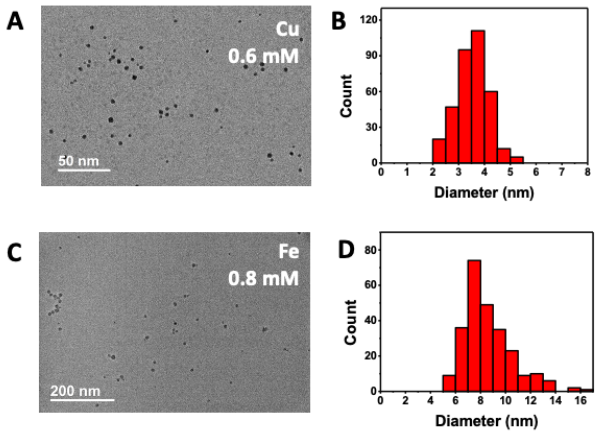


Figure 1: TEM images and nanoparticle size distributions of monometallic Cu and Fe. A. Cu TEM micrograph with nanoparticples at 0.6 mM metal concentration. B. Cu nanoparticle size distribution from the 0.6 mM metal concentration. C. Fe TEM micrograph with nanoparticles at 0.8 mM metal concentration. D. Fe nanoparticle size distribution from the 0.8 mM metal concentration.

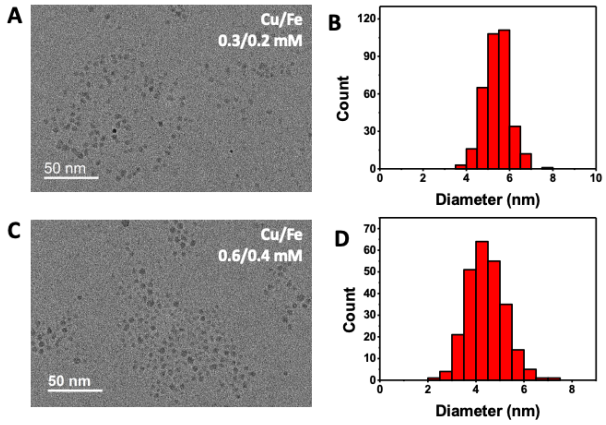


Figure 2: TEM images and nanoparticle size distributions of 2 compositions of Cu/Fe. A. Cu/Fe TEM micrograph with nanoparticles at 0.3/0.2 mM metal concentrations. B. Cu/Fe nanoparticle size distribution from the 0.3/0.2 mM sample. C. Cu/Ru TEM micrograph with nanoparticles at 0.6/0.4 mM metal concentrations. D. Cu/Ru nanoparticle size distribution from the 0.6/0.4 mM sample.

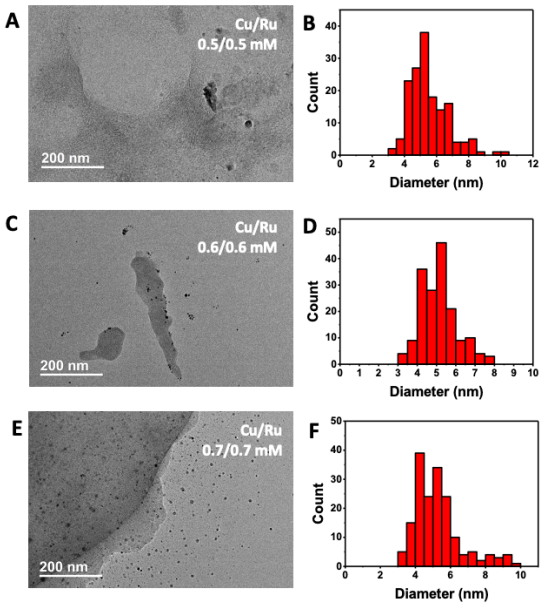


Figure 3: TEM images and nanoparticle size distributions of 3 compositions of Cu/Ru. A. Cu/Ru TEM micrograph with nanoparticles at 0.5/0.5 mM metal concentrations. B. Cu/Ru nanoparticle size distribution from the 0.5/0.5 mM sample. C. Cu/Ru TEM micrograph with nanoparticles at 0.6/0.6 mM metal concentrations. D. Cu/Ru nanoparticle size distribution from the 0.6/0.6 mM sample. E. Cu/Ru TEM micrograph with nanoparticles at 0.7/0.7 mM metal concentrations. D. Cu/Ru nanoparticle size distribution from the 0.7/0.7 mM sample.

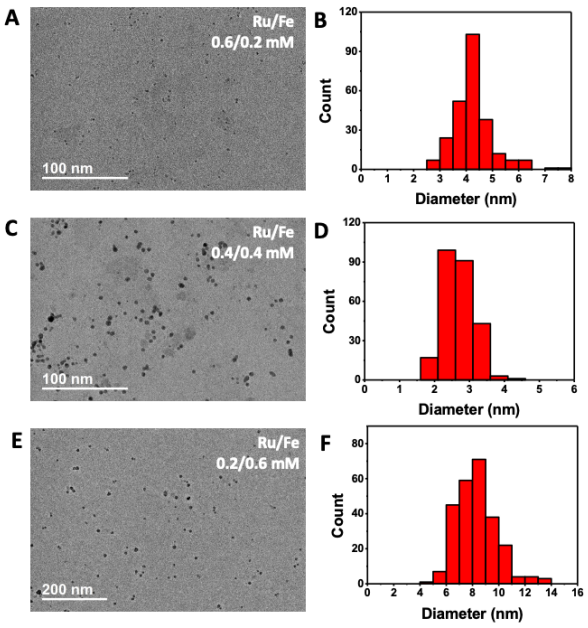


Figure 4: TEM images and nanoparticle size distributions of 3 compositions of Ru/Fe. A. Ru/Fe TEM micrograph with nanoparticles at 0.6/0.2 mM metal concentrations. B. Ru/Fe nanoparticle size distribution from the 0.6/0.2 mM sample. C. Ru/Fe TEM micrograph with nanoparticles at 0.4/0.4 mM metal concentrations. D. Ru/Fe nanoparticle size distribution from the 0.4/0.4 mM sample. E. Ru/Fe TEM micrograph with nanoparticles at 0.2/0.6 mM metal concentrations. D. Ru/Fe nanoparticle size distribution from the 0.2/0.6 mM sample.

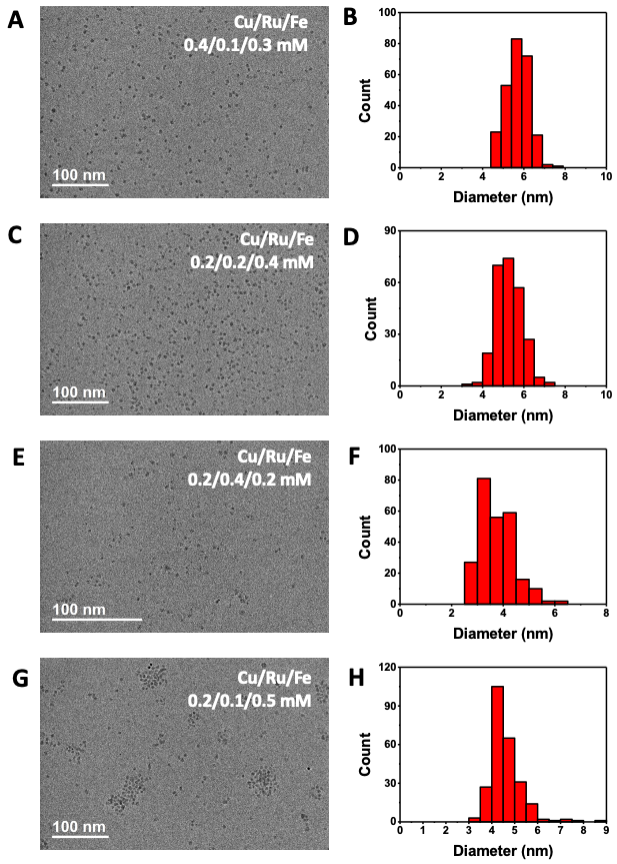


Figure 5: TEM images and nanoparticle size distributions of 4 compositions of Cu/Ru/Fe. A. Cu/Ru/Fe TEM micrograph with nanoparticles at 0.4/0.1/0.3 mM metal concentrations. B. Cu/Ru/Fe nanoparticle size distribution from the 0.4/0.1/0.3 mM sample. C. Cu/Ru/Fe TEM micrograph with nanoparticles at 0.2/0.2/0.4 mM metal concentrations. D. Cu/Ru/Fe nanoparticle size distribution from the 0.2/0.2/0.4 mM sample. E. Cu/Ru/Fe TEM micrograph with nanoparticles at 0.2/0.4/0.2 mM metal concentrations. F. Cu/Ru nanoparticle size distribution from the 0.2/0.4/0.2 mM sample. G. Cu/Ru/Fe TEM micrograph with nanoparticles at 0.2/0.1/0.5 mM metal concentrations. H. Cu/Ru nanoparticle size distribution from the 0.2/0.1/0.5 mM sample.

2 STEM-EDS Maps and Linescans

STEM-EDS mappping and linescan analysis were conducted to identify the metals present in each multimetallic sample. While the TEM micrographs determined the presence of particles, the elemental maps provide insight into the particle composition and chemical ordering. STEM-EDS maps and corresponding linescans characterization were conducted on the highest performing samples for each bi and trimetallic combination of Cu, Ru, and Fe.

Figure 6 A shows Fe and Cu containing nanoparticles. The linescans in figure 6 B are taken from the two arrows in the first map in A. The first linescan is from the top arrow, and the second linescan is from the bottom arrow. The STEM-EDS map shows two particle populations: Cu monometallic particles and bimetallic CuFe particles.

Figure 7 A shows Cu and Ru containing nanoparticles. The linescans in figure 7 B are taken from

Table 1: The metal concentrations, particle diameter, and corresponding activity of the wells that were taken for TEM imaging.

Metal	Cu	Fe	Ru	Avg Diameter (nm)	Avg Exp umolH
Cu	0.6			8.6 +/- 1.9	1.7
Fe		0.8		3.5 +/- 0.6	9.96
CuFe	0.3	0.2		5.4 +/- 0.6	10.86
CuFe	0.6	0.4		4.4 +/- 0.7	6.06
CuRu	0.5		0.5	5.0 +/- 0.9	8.88
CuRu	0.6		0.6	5.2 +/- 1.0	7.13
CuRu	0.7		0.7	5.3 +/- 1.0	7.09
RuFe		0.2	0.6	4.2 +/- 0.8	10.27
RuFe		0.4	0.4	2.7 +/- 0.4	12.13
RuFe		0.6	0.2	8.3 +/- 1.6	9.96
CuRuFe	0.4	0.3	0.1	5.8 +/- 0.6	14.56
CuRuFe	0.2	0.4	0.2	5.3 +/- 0.6	13.66
CuRuFe	0.2	0.2	0.4	3.7 +/- 0.7	14.20
CuRuFe	0.2	0.5	0.1	4.6 +/- 0.7	13.57

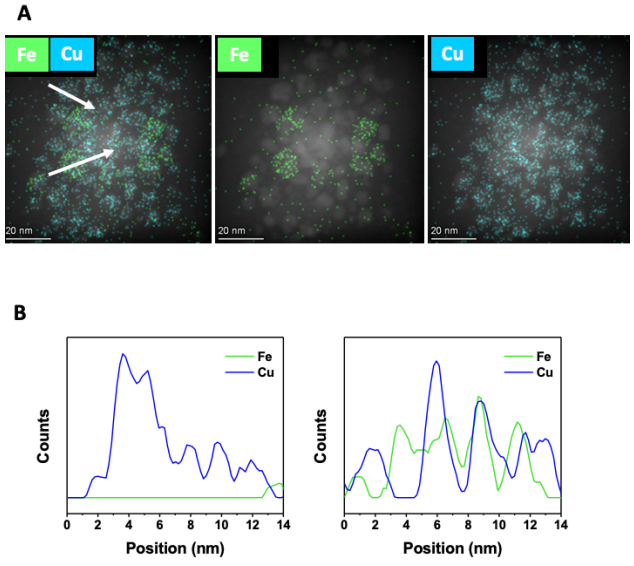


Figure 6: STEM-EDS mapping and linescans of a Cu/Fe sample at 0.6/0.4 mM metal concentration. A. HAADF images with overlayed STEM-EDS maps of CuFe sample. The first map is an overlay of Fe and Cu. The second map is the Fe signal, and the third map is the Cu signal. B. Linescans for two regions of particles for the Cu/Fe sample at 0.6/0.4 mM metal concentration.

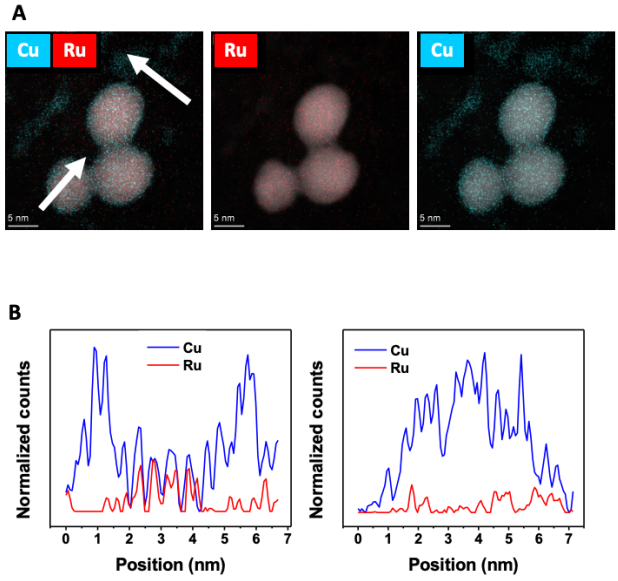


Figure 7: STEM-EDS mapping and linescans of a Cu/Ru sample at 0.5/0.5 mM metal concentration. A. HAADF images with overlayed STEM-EDS maps of CuRu sample. The first map is an overlay of Cu and Ru. The second map is the Ru signal, and the third map is the Cu signal. B. Linescans for two regions of particles for the Cu/Ru sample at 0.5/0.5 mM metal concentration.

the two arrows in the first map in A. The first linescan is from the bottom arrow, and the second linescan is from the top arrow. The second linescan is of a Cu containing organic species. The STEM-EDS map shows two particle populations: Cu containing species and bimetallic core-shell CuRu-Cu particles.

Figure 8 A shows Ru and Fe containing nanoparticles. The linescans in figure 8 B are taken from various particles in the sample. The STEM-EDS map shows two particle populations: Ru monometallic particles and bimetallic RuFe particles.

Figure 9 A shows Cu, Ru, and Fe containing nanoparticles. The linescans in figure 9 B are taken from various particles in the sample.

3 Normalization

3.1 Internal Standard

The internal standard was used to normalize experiments. On every plate, six wells were filled with the exact same composition of a molecular internal standard solution. The average activity and standard error of these wells for each experiment is plotted in Figure 10.

Given the same reaction conditions, the internal standard should perform identically. The variation seen within a plate is a feature of the experimental error. The variation seen from plate to plate is due to variability in the external conditions. These conditions could be due to temperature, humidity, light exposure, or others, and they affect the entire plate. As a result, the performance of the rest of the plate is compared to the internal standard performance to account for external variation.

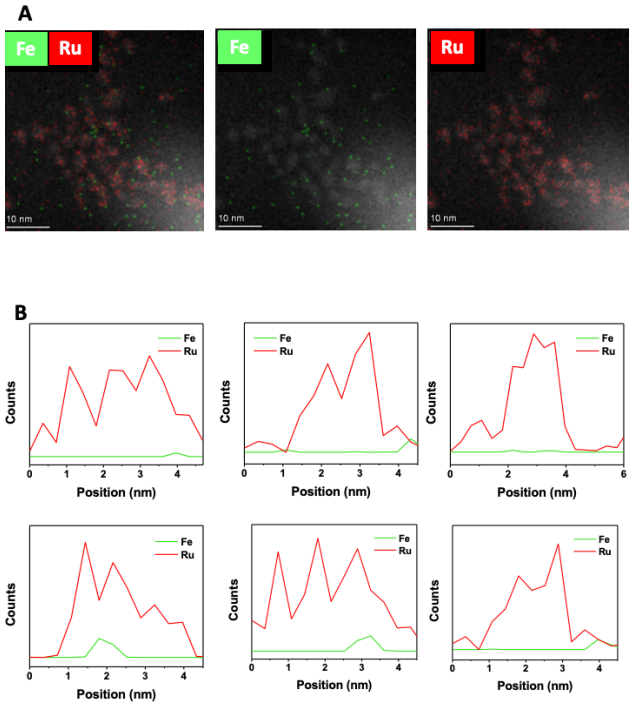


Figure 8: STEM-EDS mapping and linescans of a Ru/Fe sample at 0.4/0.4 mM. A. HAADF images with overlayed STEM-EDS maps of Ru/Fe sample. The first map is an overlay of Ru and Fe. The second map is the Ru signal, and the third map is the Fe signal. B. Linescans for particle regions for the Ru/Fe sample at 0.4/0.4 mM metal concentration.

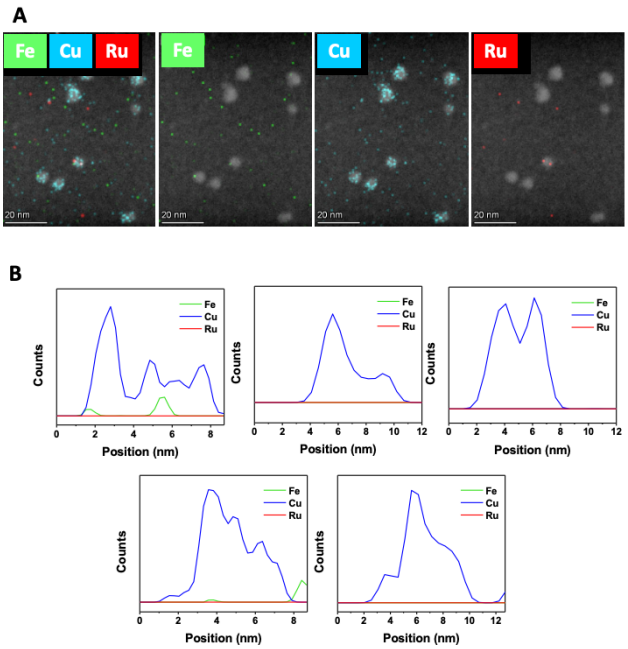


Figure 9: STEM-EDS mapping and linescans of a Cu/Ru/Fe sample at 0.4/0.1/0.3 mM. A. HAADF images with overlayed STEM-EDS maps of Cu/Ru/Fe sample. The first map is an overlay of Cu, Fe, and Ru. The second map is the Fe signal, the third map is the Cu signal, and the fourth map is the Ru signal. B. Linescans for particle regions for the Cu/Ru/Fe sample at 0.4/0.1/0.3 mM metal concentration.

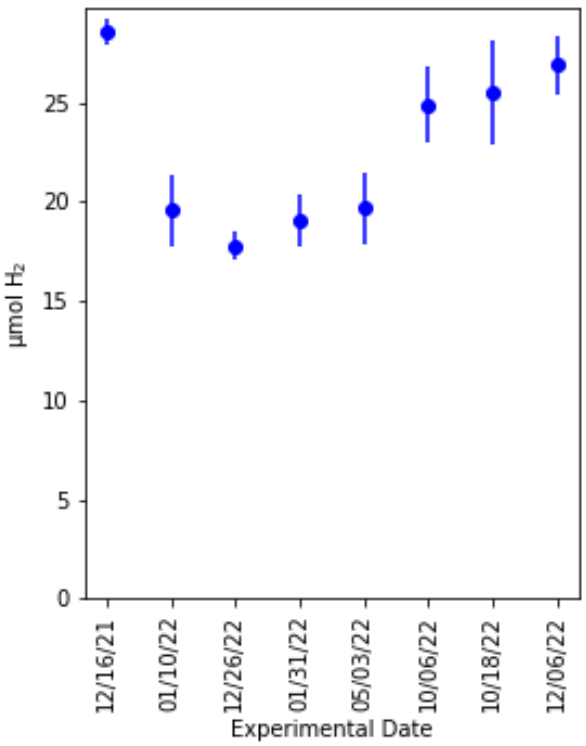


Figure 10: The average internal standard wells performance in µmol H_2 with standard error bars. Each experiment containing internal standard wells were plotted in here and the experiment dates are on the x axis.

3.2 Cu Normalization

Since the internal standard is a molecular catalyst and the system tested here is for nanoparticle catalysts, the internal standard does not account for variations associated with nanoparticle growth. Since Cu readily reduces to form nanoparticles and we observed that its HER activity is independent of concentration, every plate was also normalized by the average Cu activity in each well as shown in 11. Cu is an adequate proxy for activity sensitivity to nanoparticle growth in these experiments because we saw clear correlations where plates with high Cu activity also exhibited high multi metallic activities and vice versa.

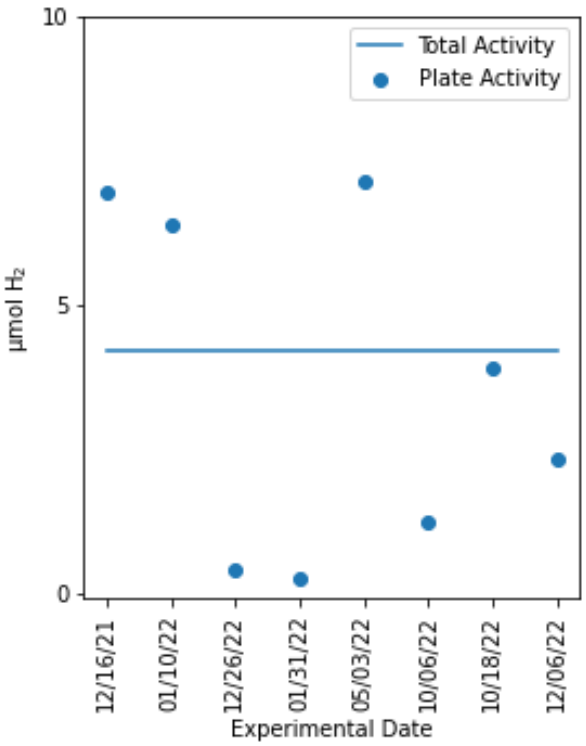


Figure 11: The average Cu well performance for each experiment is plotted here in µmol H_2, and the experiment dates are on the x axis.

Given the same reaction conditions, we would expect the nanoparticle growth and subsequently the activity here to be consistent. In each plate, the pure Cu well concentrations spanned 0.1 to 0.8 mM with no clear activity dependence on concentration. The scatter points in 11 are the average Cu activity in a plate, and the horizontal line is the average Cu activity from all of the pure Cu wells.

4 Figure generation

4.1 Generating Figures for the Manuscript

The figures from the manuscript were generated in the following sections. Data was extracted from the raw data files into consolidated json data files that are included and used in the subsequent sections. The main analysis package used to process the data is a python module called doeanalysis.

This module is a sub-module for `gespyranto`.

This section documents how the figures for the manuscript have been made with the exception of the TEM images, XPS spectra, and HAADF mapping. Each figure is represented in a section.

4.2 Data extraction

This code can only be run by people with access to the Shared GoogleDrive used in the project. This code is not broadly reproducible, but results in two json files that are included in this document which make the subsequent codes reproducible by others. We provide the code here to document how we extracted the data, and store it in a json file.

```
import sys
import os
import json

# Load gespyranto
from gespyranto.plate import Plate
from gespyranto.umolh import umolH
from gespyranto.doeanalysis import analysis

# Load plotting
from plotly.subplots import make_subplots
import plotly.graph_objects as go
from ternary_diagram import TernaryDiagram
import ternary

# model building
from sklearn.preprocessing import PolynomialFeatures
from scipy.optimize import minimize
import statsmodels.api as sm
from statsmodels.formula.api import ols


#This cell is used to generate the data dataframe that is used for the rest of this analysis
plates = ['/content/drive/Shareddrives/h2-data-science/data/ImmiscibleMetals/121621RuCuPEGSH-NH',
          '/content/drive/Shareddrives/h2-data-science/data/ImmiscibleMetals/0110CuFe',
          '/content/drive/Shareddrives/h2-data-science/data/ImmiscibleMetals/012622CuFe',
          '/content/drive/Shareddrives/h2-data-science/data/ImmiscibleMetals/013122RuCu',
          '/content/drive/Shareddrives/h2-data-science/data/ImmiscibleMetals/031522CuFeRu',
          '/content/drive/Shareddrives/h2-data-science/data/ImmiscibleMetals/050322RuCu',
          '/content/drive/Shareddrives/h2-data-science/data/ImmiscibleMetals/100622RuCuFe',
          '/content/drive/Shareddrives/h2-data-science/data/ImmiscibleMetals/101822RuCuFe',
          '/content/drive/Shareddrives/h2-data-science/data/ImmiscibleMetals/120622RuCuFe']
df = pd.DataFrame({})
for i in plates:
  print(i)
  p1 = Plate(i)
  umolH1 = p1.data['umolH']
  df1 = umolH1.dataframe()
  df = pd.concat([df, df1])

# remove condensed internal standard
df = df[~((df.directory.str.contains('1206')&(df.index.isin([99, 100]))))]
#remove condensed rufe with no replicates on side
df = df[~((df.directory.str.contains('1206')&(df.index.isin([62, 63]))))]

#remove mispipetted wells
dead_wells = np.array([9, 11, 34, 46, 56, 58, 70,82, 83, 92, 94, 95])
df = df[~((df.directory.str.contains('1006')&(df.index.isin(dead_wells))))]
df = df[~((df.directory.str.contains('1006')&(df.Fe>=0.5)))]

# Internal Standard DataFrame
dfint=df[df.attributes.str.contains('Internal Standard|Int Stand')]
# get scales for normalization
```

```
int_stand_avg = []
int_stand_std = []
for i in dfint.directory.unique():
    df_umol = dfint[dfint.directory == i].umolH_max
    int_stand_avg.append(np.mean(df_umol))
    int_stand_std.append(np.std(df_umol))
int_stand = int_stand_avg/int_stand_avg[0]
norm_scale = dict(zip(dfint.directory.unique(), int_stand))

#save internal standard dataframe
dfint.reset_index(inplace = True)
os.chdir('/content/drive/Shareddrives/h2-data-science/publications/4-curu-cufe/')
intstand = dfint.to_json('manuscript/data/intstand.json')

#clean up dataframe
df = df[~df.attributes.str.contains('Internal Standard|empty|None|none')]
df = df[df.attributes.str.contains('PEGSH')].copy()
df.drop(columns = ['None', 'PEGamine'], inplace = True)
df.fillna(0, inplace = True)

# Scale by Internal Standard
def scale_int(row):
  dir = row.directory
  return norm_scale[dir]
df['scale'] = df.apply(scale_int, axis = 1)
df['norm'] = df.scale * df.umolH_max

# Scale by Cu
cuMon = df[(df.Cu != 0)&(df.Ru == 0)&(df.Fe == 0)].copy()
cu_stand_avg = []
cu_stand_std = []
for i in range(len(cuMon.directory.unique())):
    dfdir = cuMon[cuMon.directory == cuMon.directory.unique()[i]]
    cu_stand_avg.append(dfdir.norm.mean())
    cu_stand_std.append(np.std(dfdir.norm))
cu_stand = cu_stand_avg

# since the limit of detection of activity is 2, any average under 2 is left alone and
# any average above 2 is adjusted so that the dividing factor gives us ~ 2
for i in range(len(cu_stand)):
    if cu_stand[i]<2:
      cu_stand[i] =1
    if cu_stand[i]>2:
      cu_stand[i] =cu_stand[i]/2
cu_scale = dict(zip(cuMon.directory.unique(), cu_stand))

def scale_cu(row):
  dir = row.directory
  return cu_scale[dir]
df['cu_scale'] = df.apply(scale_cu, axis = 1)
df['cu_norm'] = df.norm/(df.cu_scale)

#save dataframe
os.chdir('/content/drive/Shareddrives/h2-data-science/publications/4-curu-cufe/')
df.reset_index(inplace = True)
data = df.to_json('manuscript/data/data.json')
```

The data is attached to this pdf here:

supporting-information.org

data.json

intstand.json

These files can be extracted and used in the subsequent sections.

4.3 Loading the data

```
import pandas as pd

df = pd.read_json('data/data.json')
df.set_index('Well-number', inplace = True)
dfint = pd.read_json('data/intstand.json')
```

4.4 Figure 1

```
import matplotlib.pyplot as plt

df3 = df[['Cu', 'Fe', 'Ru', 'cu_norm', 'umolH_max', 'directory']]
cuMon = df3[(df3.Cu != 0)&(df3.Ru == 0)&(df3.Fe == 0)].copy()
ruMon = df3[(df3.Cu == 0)&(df3.Ru != 0)&(df3.Fe == 0)].copy()
feMon = df3[(df3.Cu == 0)&(df3.Ru == 0)&(df3.Fe != 0)].copy()
feMon = feMon[~((feMon.Fe>0.69)&(feMon.Fe<0.71))]
directories = df.directory.unique()

fig, (ax, ax1, ax2) = plt.subplots(3,1, figsize = (5, 14),sharey = True)
x = cuMon.groupby('Cu').mean(numeric_only=True).index.to_numpy()
y = cuMon.groupby('Cu').mean(numeric_only=True).cu_norm.to_numpy()
std = cuMon.groupby('Cu').std().cu_norm.to_numpy()
ax.errorbar(x, y, yerr = std, fmt = 'o')
ax.set_xlabel(f'Cu Concentration (mM)', fontsize = 16)
ax.tick_params(axis='both', which='major', labelsize=14)
ax.set_ylim((0,14))
ax.set_ylabel(u' Scaled \u03bcmol H$_2$',fontsize = 16)
ax.text(-0.1, 15,'A', fontsize = 22)
ax.plot(np.linspace(0, 1), np.repeat(1.1,len(np.linspace(0, 1))), 'k--', alpha =0.5)

x = ruMon.groupby('Ru').mean(numeric_only=True).index.to_numpy()
y = ruMon.groupby('Ru').mean(numeric_only=True).cu_norm.to_numpy()
std = ruMon.groupby('Ru').std().cu_norm.to_numpy()
ax1.errorbar(x, y, yerr = std, fmt = 'o', color = 'red')
ax1.set_xlabel(f'Ru Concentration (mM)', fontsize = 16)
ax1.tick_params(axis='both', which='major', labelsize=14)
ax1.set_ylabel(u'Scaled \u03bcmol H$_2$',fontsize = 16)
ax1.text(-0.1, 15,'C', fontsize = 22)
ax1.plot(np.linspace(0, 1), np.repeat(1.1, len(np.linspace(0, 1))), 'k--', alpha =0.5)

x = feMon.groupby('Fe').mean(numeric_only=True).index.to_numpy()
y = feMon.groupby('Fe').mean(numeric_only=True).cu_norm.to_numpy()
std = feMon.groupby('Fe').std().cu_norm.to_numpy()
ax2.errorbar(x, y, yerr = std, fmt = 'o', color = 'green')
ax2.set_xlabel(f'Fe Concentration (mM)', fontsize = 16)
ax2.tick_params(axis='both', which='major', labelsize=14)
ax2.set_ylabel(u'Scaled \u03bcmol H$_2$',fontsize = 16)
ax2.text(-0.1, 15,'E', fontsize = 22)
ax2.plot(np.linspace(0, 1), np.repeat(1.1,len(np.linspace(0, 1))), 'k--', alpha =0.5)

plt.tight_layout()
for ext in ['png', 'pdf']:
  fig.savefig(f'figures/monometallic-activities-curufe.{ext}', dpi=600)
plt.close()

from pycse.orgmode import Figure
Figure('./figures/monometallic-activities-curufe.png',
       caption='Figure 1 in the manuscript.',
       attributes=(('org', ':width 300'),
                   ('latex', ':placement [H] :width 3.25in')))
```

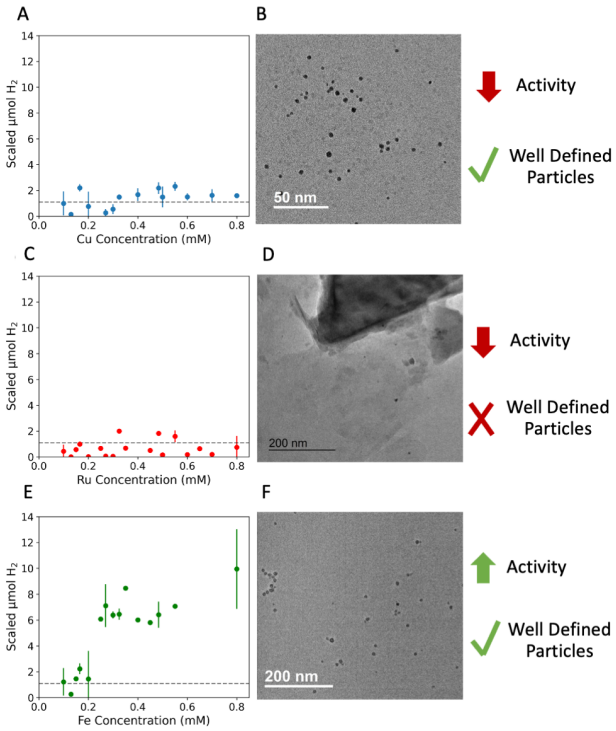


Figure 12: Figure 1 in the manuscript.

4.5 Figure 2

```
# create bimetallic dataframes
dfcuru = df[(df.Cu != 0)&(df.Ru != 0)&(df.Fe == 0)]
dfcufe = df[(df.Cu != 0)&(df.Ru == 0)&(df.Fe != 0)]
dfrufe = df[(df.Cu == 0)&(df.Ru != 0)&(df.Fe != 0)]

#plot Cu Ru
metals = ['Cu', 'Ru']

fig, (ax, ax1, ax2) = plt.subplots(nrows = 1, ncols = 3, figsize = (12,7), sharey = True)
#activity
m = analysis(dfcuru, metals, metals, 'cu_norm').avg_df().drop(columns = ['cu_norm_med'])
m.fillna(0, inplace = True)
m = m[~(m.cu_norm_avg == 0)]
m['molFrac'] = m[metals[0]]/(m[metals[0]] + m[metals[1]])
m['molTot'] = m[metals[0]] + m[metals[1]]
x = m.molFrac.to_numpy()
y = m.cu_norm_avg.to_numpy()
ax.scatter(x, y, s = (m.molTot)*150, alpha = 0.7, color = 'purple')
ax.tick_params(axis='both', which='major', labelsize=14)
ax.set_xlabel('Mole Fraction Cu', fontsize = 16)
ax.set_ylabel('Scaled \u03bcmol H$_2$', fontsize = 16)
ax.set_title('CuRu', fontsize = 16)
ax.scatter(np.ones(len(cuMon.Cu.unique())),
           cuMon.groupby('Cu').mean(numeric_only=True).cu_norm,
           s = cuMon.groupby('Cu').count().index.to_numpy()*150)
ax.scatter(np.zeros(len(ruMon.Ru.unique())),
           ruMon.groupby('Ru').mean(numeric_only=True).cu_norm,
           s = ruMon.groupby('Ru').count().index.to_numpy()*150, c = 'red')
ax.text(0, 17,'A', fontsize = 20)
ax.legend(['CuRu', 'Cu', 'Ru'],  fontsize = 14)
ax.plot(np.linspace(0, 1), np.repeat(1.1,len(np.linspace(0, 1))), 'k--', alpha =0.5)

```

```
# plot Cu Fe
metals = ['Fe', 'Cu']
m = analysis(dfcufe, metals, metals, 'cu_norm').avg_df().drop(columns = ['cu_norm_med'])
m.fillna(0, inplace = True)
m = m[~(m.cu_norm_avg == 0)]
m['molFrac'] = m[metals[0]]/(m[metals[0]] + m[metals[1]])
m['molTot'] = m[metals[0]] + m[metals[1]]
x = m.molFrac.to_numpy()
y = m.cu_norm_avg.to_numpy()
ax1.scatter(x, y, s = (m.molTot)*150, alpha = 0.7, color = 'purple')
ax1.tick_params(axis='both', which='major', labelsize=14)
ax1.set_xlabel('Mole Fraction Fe', fontsize = 16)
ax1.set_title('CuFe', fontsize = 16)
ax1.scatter(np.zeros(len(cuMon.Cu.unique())),
           cuMon.groupby('Cu').mean(numeric_only=True).cu_norm,
           s = cuMon.groupby('Cu').count().index.to_numpy()*150)
ax1.scatter(np.ones(len(feMon.Fe.unique())),
           feMon.groupby('Fe').mean(numeric_only=True).cu_norm,
           s = feMon.groupby('Fe').count().index.to_numpy()*150, c = 'green')
ax1.text(0, 17,'B', fontsize = 20)
ax1.legend(['CuFe', 'Cu', 'Fe'],  fontsize = 14, loc='upper right')
ax1.plot(np.linspace(0, 1), np.repeat(1.1,len(np.linspace(0, 1))), 'k--', alpha =0.5)

# plot activity for RuFe
metals = ['Ru', 'Fe']
m = analysis(dfrufe, metals, metals, 'cu_norm').avg_df().drop(columns = ['cu_norm_med'])
m.fillna(0, inplace = True)
m = m[~(m.cu_norm_avg == 0)]
m['molFrac'] = m[metals[0]]/(m[metals[0]] + m[metals[1]])
m['molTot'] = m[metals[0]] + m[metals[1]]
x = m.molFrac.to_numpy()
y = m.cu_norm_avg.to_numpy()

ax2.scatter(x, y, s = (m.molTot)*150, alpha = 0.7, color = 'purple')
ax2.tick_params(axis='both', which='major', labelsize=14)
ax2.set_xlabel('Mole Fraction Ru', fontsize = 16)
ax2.set_title('RuFe', fontsize = 16)
ax2.scatter(np.ones(len(ruMon.Ru.unique())),
           ruMon.groupby('Ru').mean(numeric_only=True).cu_norm,
           s = ruMon.groupby('Ru').count().index.to_numpy()*150, c = 'red')
ax2.scatter(np.zeros(len(feMon.Fe.unique())),
           feMon.groupby('Fe').mean(numeric_only=True).cu_norm,
           s = feMon.groupby('Fe').count().index.to_numpy()*150, c = 'green')
ax2.text(0, 17,'C', fontsize = 20)
ax2.legend(['RuFe', 'Ru', 'Fe'], fontsize = 14)
ax2.plot(np.linspace(0, 1), np.repeat(1.1,len(np.linspace(0, 1))), 'k--', alpha =0.5)

  plt.legend(loc='upper center', ncol=2)
  plt.tight_layout()
  for ext in ['png', 'pdf']:
    plt.savefig(f'figures/curu-cufe-rufe-activity.{ext}', dpi=600)
  plt.close()
  return Figure('./figures/curu-cufe-rufe-activity.png',
                caption='Figure 2 in the manuscript',
                attributes=(('org', ':width 500'),
                            ('latex', ':placement [H] :width 3.25in')))
```

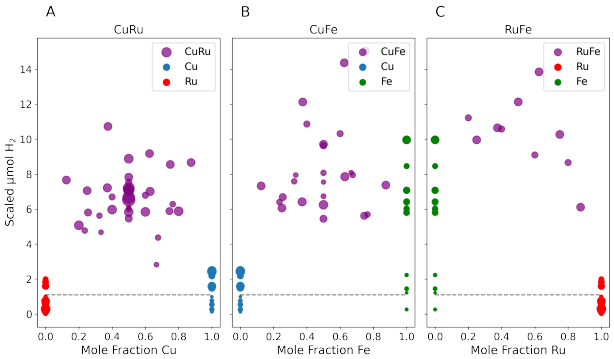


Figure 13: Figure 3 in the manuscript

4.6 Figure 3

```
# monometallic models
#cu
cudata = cuMon[['Cu', 'cu_norm']].groupby('Cu').mean(numeric_only=True)
std = cuMon.groupby('Cu').std(numeric_only=True).cu_norm
cumodel = sm.OLS(cudata.cu_norm, cudata.index).fit()

#ru
rudata = ruMon[['Ru', 'cu_norm']].groupby('Ru').mean(numeric_only=True)
std = ruMon.groupby('Ru').std(numeric_only=True).cu_norm
rumodel = sm.OLS(rudata.cu_norm, rudata.index).fit()

#fe
fedata = feMon[feMon.Fe != 0.6].groupby('Fe').mean(numeric_only=True)
std = feMon[feMon.Fe != 0.6].groupby('Fe').std(numeric_only=True).cu_norm
x = np.array([fedata.index, 1/fedata.index, fedata.index**2, np.log(fedata.index)])
femodel = sm.OLS(fedata.cu_norm, x.T).fit()

#synergy calculations
def synergy(row):
    cu = row.Cu
    ru = row.Ru
    fe = row.Fe
    cu_pure = cumodel.predict(cu)
    ru_pure = rumodel.predict(ru)
    if fe != 0:
        fe_x = np.array([fe, 1/fe, fe**2, np.log(fe)])
    elif fe == 0:
        fe_x = np.array([0, 0, 0, 0])
    fe_pure = femodel.predict(fe_x.T)
    if fe_pure<0:
      fe_pure = 0
    syn = row.cu_norm - cu_pure - ru_pure - fe_pure
    return syn[0]

dfsyn = pd.DataFrame()
for i in df.directory.unique():
  dfdir = df[df.directory == i]
  cuMon1 = dfdir[(dfdir.Cu!=0)&(dfdir.Ru == 0)&(dfdir.Fe == 0)]
  ruMon1 = dfdir[(dfdir.Cu==0)&(dfdir.Ru != 0)&(dfdir.Fe == 0)]
  feMon1 = dfdir[(dfdir.Cu==0)&(dfdir.Ru == 0)&(dfdir.Fe != 0)]

  dfdir.loc[:,'synergy']= dfdir.apply(synergy, axis = 1)
  dfsyn = pd.concat([dfsyn,
                     dfdir[['Cu', 'Fe', 'Ru', 'synergy', 'cu_norm', 'umolH_max', 'directory', 'attributes']]])
syncuru = dfsyn[(dfsyn.Cu != 0)&(dfsyn.Ru != 0)&(dfsyn.Fe == 0)]
syncufe = dfsyn[(dfsyn.Cu != 0)&(dfsyn.Ru == 0)&(dfsyn.Fe != 0)]
```

```
synrufe = dfsyn[(dfsyn.Cu == 0)&(dfsyn.Ru != 0)&(dfsyn.Fe != 0)]

fig, (ax, ax1, ax2) = plt.subplots(nrows = 1, ncols = 3, figsize = (12,7), sharey = True)

metals = ['Cu', 'Ru']
m = analysis(syncuru, metals, metals, 'synergy').avg_df().drop(columns = ['synergy_med'])
m.fillna(0, inplace = True)
m = m[~(m.synergy_avg == 0)]
m['molFrac'] = m[metals[0]]/(m[metals[0]] + m[metals[1]])
m['molTot'] = m[metals[0]] + m[metals[1]]
x = m.molFrac.to_numpy()
y = m.synergy_avg.to_numpy()
ax.scatter(x, y, s = (m.molTot)*150, alpha = 0.7, color = 'purple')
ax.set_yticks(np.arange(-3, 14))
ax.tick_params(axis='both', which='major', labelsize=14)
ax.set_xlabel('Mole Fraction Cu (mM)', fontsize = 16)
ax.set_ylabel('Scaled Synergy \u03bcmol H$_2$', fontsize = 16)
ax.set_title('CuRu', fontsize = 16)
ax.scatter(1, 0 , s = 200)
ax.scatter(0, 0, c = 'red', s = 200)
ax.text(0, 11,'A', fontsize = 20)
ax.legend(['CuRu', 'Cu', 'Ru'], fontsize = 14)
ax.plot([0,1], [0,0], 'k-')

metals = ['Fe', 'Cu']
m = analysis(syncufe, metals, metals, 'synergy').avg_df().drop(columns = ['synergy_med'])
m.fillna(0, inplace = True)
m = m[~(m.synergy_avg == 0)]
m['molFrac'] = m[metals[0]]/(m[metals[0]] + m[metals[1]])
m['molTot'] = m[metals[0]] + m[metals[1]]
x = m.molFrac.to_numpy()
y = m.synergy_avg.to_numpy()
ax1.scatter(x, y, s = (m.molTot)*150, alpha = 0.7, color = 'purple')
ax1.tick_params(axis='both', which='major', labelsize=14)
ax1.set_xlabel('Mole Fraction Fe (mM)', fontsize = 16)
ax1.set_title('CuFe', fontsize = 16)
ax1.scatter(1, 0 ,c = 'green', s = 200)
ax1.scatter(0, 0, s = 200)
ax1.legend(['CuFe', 'Fe', 'Cu'], fontsize = 14)
ax1.text(0, 11,'B', fontsize = 20)
ax1.plot([0,1], [0,0], 'k-')

metals = ['Ru', 'Fe']
m = analysis(synrufe, metals, metals, 'synergy').avg_df().drop(columns = ['synergy_med'])
m.fillna(0, inplace = True)
m = m[~(m.synergy_avg == 0)]
m['molFrac'] = m[metals[0]]/(m[metals[0]] + m[metals[1]])
m['molTot'] = m[metals[0]] + m[metals[1]]
x = m.molFrac.to_numpy()
y = m.synergy_avg.to_numpy()
ax2.scatter(x, y, s = (m.molTot)*150, alpha = 0.7, color = 'purple')
ax2.tick_params(axis='both', which='major', labelsize=14)
ax2.set_xlabel('Mole Fraction Ru (mM)', fontsize = 16)
ax2.set_title('RuFe', fontsize = 16)
ax2.scatter(1, 0 ,c = 'red', s = 200)
ax2.scatter(0, 0, c = 'green', s = 200)
ax2.text(0, 11,'C', fontsize = 20)
ax2.legend(['RuFe', 'Ru', 'Fe'], fontsize = 14, loc = 'upper right')
ax2.plot([0,1], [0,0], 'k-')

plt.tight_layout()
for ext in ['png', 'pdf']:
  plt.savefig(f'figures/curu-cufe-rufe-synergy.{ext}', dpi = 600)
plt.close()

Figure('./figures/curu-cufe-rufe-synergy.png',
       caption='Figure 3 in the manuscript',
       attributes=(('org', ':width 500'),
```

```
115                                    ('latex', ':placement [H] :width 3.25in')))
```

/var/folders/3q/ht_2mtk52hl7ydxrcr87z2gr0000gn/T/ipykernel_6969/278245199.py:18: SettingWithCopyWarning: A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy df1['int_stand_base'][df1.path == i] = v /var/folders/3q/ht_2mtk52hl7ydxrcr87z2gr0000gn/T/ipykernel_6969/278245199.py:100: UserWarning: This figure includes Axes that are not compatible with tight_layout, so results might be incorrect. plt.tight_layout()

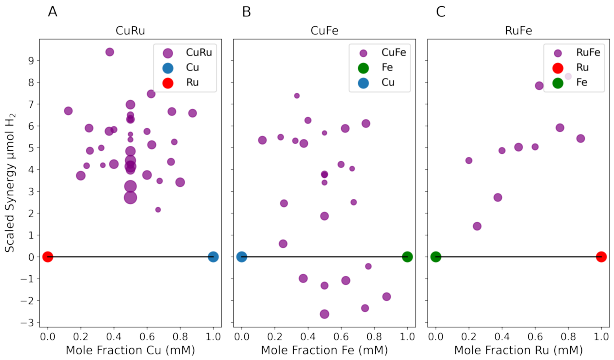


Figure 14: Figure 3 in the manuscript

4.7 Figure 5

4.7.1 Figure 5a

```
1  # get Trimetallic dataframe
2  dftri = df[df.directory.str.contains('1006|1018|1206')].reset_index(drop = True)
3  dftri.Ru[dftri[(dftri.Ru< 0.064)&(dftri.Ru>0.062)].index] = 0.05
4  dftri.Fe[dftri[(dftri.Fe< 0.064)&(dftri.Fe>0.062)].index] = 0.05
5
6  #remove poorly pipetted wells
7  dftri = dftri[~(dftri.directory.str.contains('1006')&(dftri.Fe>=0.5))]
8  dftri = dftri[~(dftri.directory.str.contains('1018')&((dftri.Fe!=0)&(dftri.Cu!=0)&((dftri.Ru!=0))))]
9
10 #create monometallic df
11 dfcuMon = dftri[(dftri.Cu != 0)&(dftri.Ru == 0)&(dftri.Fe == 0)].copy()
12 dfruMon = dftri[(dftri.Cu == 0)&(dftri.Ru != 0)&(dftri.Fe == 0)].copy()
13 dffeMon = dftri[(dftri.Cu == 0)&(dftri.Ru == 0)&(dftri.Fe != 0)].copy()
14 dfMon = pd.concat([dfcuMon, dfruMon, dffeMon])
15 dftri = dftri.drop(dfMon.index, axis = 0)
16
17 #create multi metallic df with averages
18 dfMul = dftri.copy()
19
20 unique_vals = []
21 comp = ['Cu', 'Ru', 'Fe']
22 for i in comp:
23     unique_vals.append(dfMul[i].unique())
24
25 df1 = pd.DataFrame(columns = comp + [f'umolH_max_avg', f'umolH_max_max', f'umolH_max_stddev'])
26 for i in unique_vals[0]:
27   for j in unique_vals[1]:
28     for k in unique_vals[2]:
29       df_temp = dfMul[(dfMul[comp[0]] == i)&(dfMul[comp[1]] == j)&(dfMul[comp[2]] == k)]
30       if len(df_temp)>0:
```

```
        avg = np.array(df_temp['cu_norm'].values.tolist()).mean(axis = 0)
        med = np.max(np.array(df_temp['cu_norm'].values.tolist()))
        error = np.array(df_temp['cu_norm'].values.tolist()).std(axis = 0)
        ind = df_temp.index.values
        df1 = pd.concat([df1,pd.DataFrame({comp[0]: i,
                        comp[1]: j,
                        comp[2]: k,
                        f'umolH_max_avg': avg,
                        f'umolH_max_max': med,
                        f'umolH_max_stddev': error,
                        'wells': [ind]})])
df1.reset_index(drop='index', inplace = True)
df1.drop(index = 23, inplace = True)

dfmon08 = pd.DataFrame(np.array([[0.8, 0, 0, dfcuMon[dfcuMon.Cu == 0.7].cu_norm.mean()],
                                 [0, 0.8, 0, dfruMon[dfruMon.Ru == 0.8].cu_norm.mean()],
                                 [0, 0, 0.8, dffeMon[dffeMon.Fe == 0.8].cu_norm.mean()]]),
                        columns = ['Cu', 'Ru', 'Fe', 'umolH_max_avg'])

# corners of the trimetallic
df1 = pd.concat([df1, dfmon08])
df1.fillna(0, inplace = True)
```

4.7.2 Figure 5a and b

```
# make dataframe with trimetallics
dftri = dfsyn[dfsyn.directory.str.contains('1006|1018|1206')].reset_index(drop = True)
dftri.Ru[dftri[(dftri.Ru< 0.064)&(dftri.Ru>0.062)].index] = 0.05
dftri.Fe[dftri[(dftri.Fe< 0.064)&(dftri.Fe>0.062)].index] = 0.05
#remove poorly pipetted wells
dftri = dftri[~(dftri.directory.str.contains('1006')&(dftri.Fe>=0.5))]
dftri = dftri[~(dftri.directory.str.contains('1018')&((dftri.Fe!=0)&(dftri.Cu!=0)&((dftri.Ru!=0))))]

#create monometallic df
dfcuMon = dftri[(dftri.Cu != 0)&(dftri.Ru == 0)&(dftri.Fe == 0)].copy()
dfruMon = dftri[(dftri.Cu == 0)&(dftri.Ru != 0)&(dftri.Fe == 0)].copy()
dffeMon = dftri[(dftri.Cu == 0)&(dftri.Ru == 0)&(dftri.Fe != 0)].copy()
dfMon = pd.concat([dfcuMon, dfruMon, dffeMon])
dftri = dftri.drop(dfMon.index, axis = 0)

#Get multimetallic avg df
dfMul = dftri.copy()

unique_vals = []
comp = ['Cu', 'Ru', 'Fe']
for i in comp:
    unique_vals.append(dfMul[i].unique())

df2 = pd.DataFrame(columns = comp + [f'syn_avg', f'syn_max', f'syn_stddev', 'umolH_max_avg'])
for i in unique_vals[0]:
  for j in unique_vals[1]:
    for k in unique_vals[2]:
      df_temp = dfMul[(dfMul[comp[0]] == i)&(dfMul[comp[1]] == j)&(dfMul[comp[2]] == k)]
      if len(df_temp)>0:
        avg = np.array(df_temp['synergy'].values.tolist()).mean(axis = 0)
        med = np.max(np.array(df_temp['synergy'].values.tolist()))
        error = np.array(df_temp['synergy'].values.tolist()).std(axis = 0)
        umolavg = np.array(df_temp['umolH_max'].values.tolist()).mean(axis = 0)
        ind = df_temp.index.values
        df2 = pd.concat([df2,pd.DataFrame({comp[0]: i,
                        comp[1]: j,
                        comp[2]: k,
                        f'syn_avg': avg,
                        f'syn_max': med,
                        f'syn_stddev': error,
                        f'umolH_max_avg': umolavg,
```

```
                          'wells': [ind]})])
df2.reset_index(drop='index', inplace = True)
df2.drop(index = [23, 59], inplace = True)

#add corners
dfmon08['syn_avg'] = 0
dfmon08['syn_max'] = 0
dfmon08['syn_stddev'] = 0
dfmon08['totmet'] = 0.8

df2 = pd.concat([df2, dfmon08])
df2.fillna(0, inplace = True)

#plot figure
fig, axes  = plt.subplots(2, 1, figsize = (5, 9), facecolor='white')
materials = ['Ru', 'Fe', 'Cu']

td1 = TernaryDiagram(materials=materials, ax=axes[0])
td1.contour(df1[materials],
            z = df1['umolH_max_avg'],
            z_min = 0,
            z_max = df1['umolH_max_avg'].max(), verbose = True, algorithm = 'serial')

td2 = TernaryDiagram(materials=materials, ax=axes[1])
td2.contour(df2[materials],
            z = df2['syn_avg'],
            z_min = df2['syn_avg'].min(),
            #z_max = 17)
            z_max = df2['syn_avg'].max())
fig.text(0.97, 0.7,u' Scaled \u03bcmol H$_2$', fontsize = 13, rotation = 270)
fig.text(0.97, 0.15,u'Scaled Synergy \u03bcmol H$_2$', fontsize = 13, rotation = 270)
fig.text(0.05, 0.93,'A', fontsize = 16)
fig.text(0.05, 0.46,'B', fontsize = 16)

fig.tight_layout()
for ext in ['png', 'pdf']:
    plt.savefig(f'figures/activity-synergy-ternary.{ext}', dpi=600)
  plt.close()
Figure(f'./figures/activity-synergy-ternary.png',
            caption='Figure 5 in the manuscript.',
            attributes=(('org', ':width 500'),
                        ('latex', ':placement [H] :width 3.25in')))


```

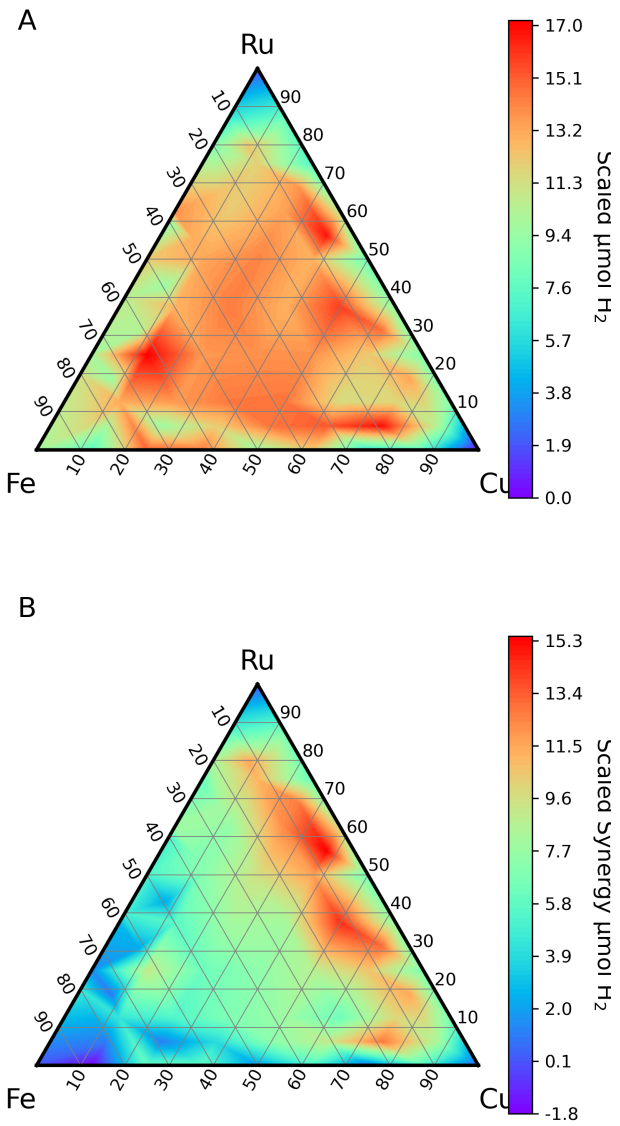


Figure 15: Figure 5 in the manuscript.

4.8 SI Figure for ternary data points

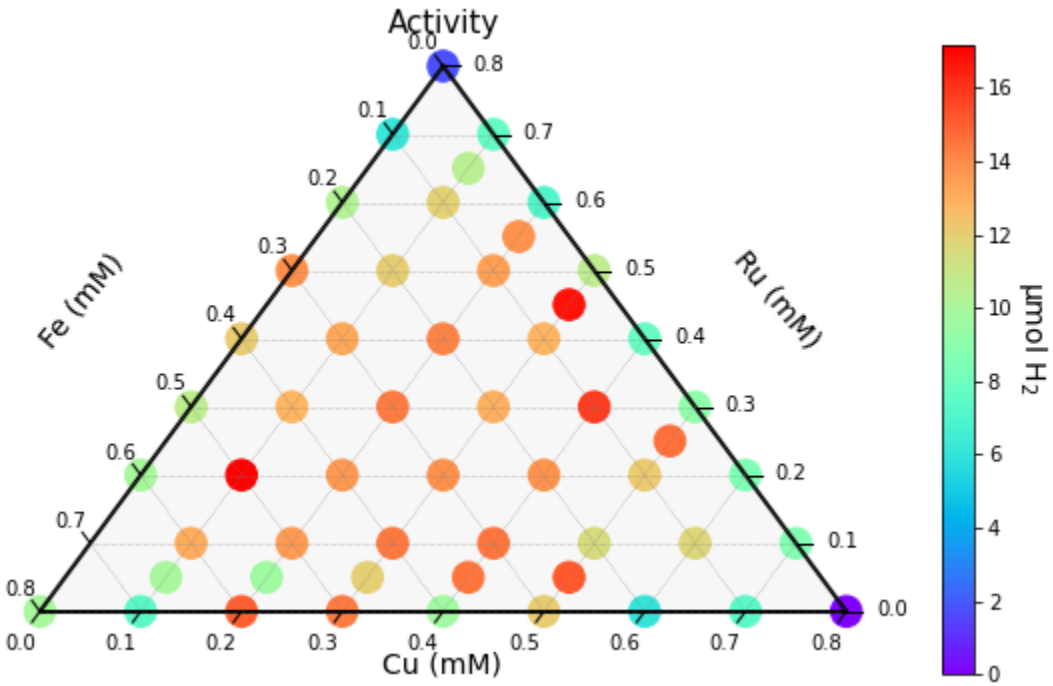
This plot is the measured activity data after Cu and internal standard normalization for the ternary system at 0.8 mM total metal concentration. The plots shown in the main text are interpolated heat maps from this data.

```
# drop the values outside of 0.8 mM
df_scatter = df1.drop(index = [41, 44, 45, 46, 37, 54, 55, 58, 59, 60, 61, 62, 63, 64, 65])

# Create x and y
x = df_scatter[['Cu', 'Ru', 'Fe']].to_numpy()
y = df_scatter['umolH_max_avg'].to_numpy()
```

```

scaled = (y - np.min(y)) / (np.max(y)-np.min(y))

scale = 0.8
figure, tax = ternary.figure(scale=scale)
figure.set_size_inches(8,5)
stddev = ((std+1)*200)

# Plot data
norm = plt.Normalize(vmin=np.min(y), vmax=np.max(y))
cmap = 'rainbow'
ax = tax.scatter(x, c = scaled, s = 240, cmap = cmap)
figure.colorbar(plt.cm.ScalarMappable(norm=norm, cmap=cmap), ax = ax)

ax.set_title("Activity", fontsize=15)
tax.boundary(linewidth=2.0)
tax.gridlines(multiple=0.1, color="grey")
tax.ticks(axis='lbr', linewidth=1, multiple=0.1, tick_formats="%.1f", offset = 0.02)
tax.clear_matplotlib_ticks()
tax.get_axes().axis('off')
tax.left_axis_label("Fe (mM)", offset = 0.2, fontsize = 14)
tax.right_axis_label("Ru (mM)", offset = 0.2, fontsize = 14)
tax.bottom_axis_label("Cu (mM)", offset = 0.2, fontsize = 14)
figure.text(0.9, 0.45,u'\u03bcmol H$_2$', rotation = 270, fontsize = 14)
figure.tight_layout()

figure.savefig(f'figures/si-activity-ternary-avg.png', dpi = 300)
plt.close()
Figure('./figures/si-activity-ternary-avg.png',
       attributes=(('org', ':width 500'),
                   ('latex', ':placement [H] :width 6in')))
```

/var/folders/3q/ht_2mtk52hl7ydxrcr87z2gr0000gn/T/ipykernel_6969/2523530347.py:18: SettingWithCopyWarning: A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy df1['int_stand_base'][df1.path == i] = v

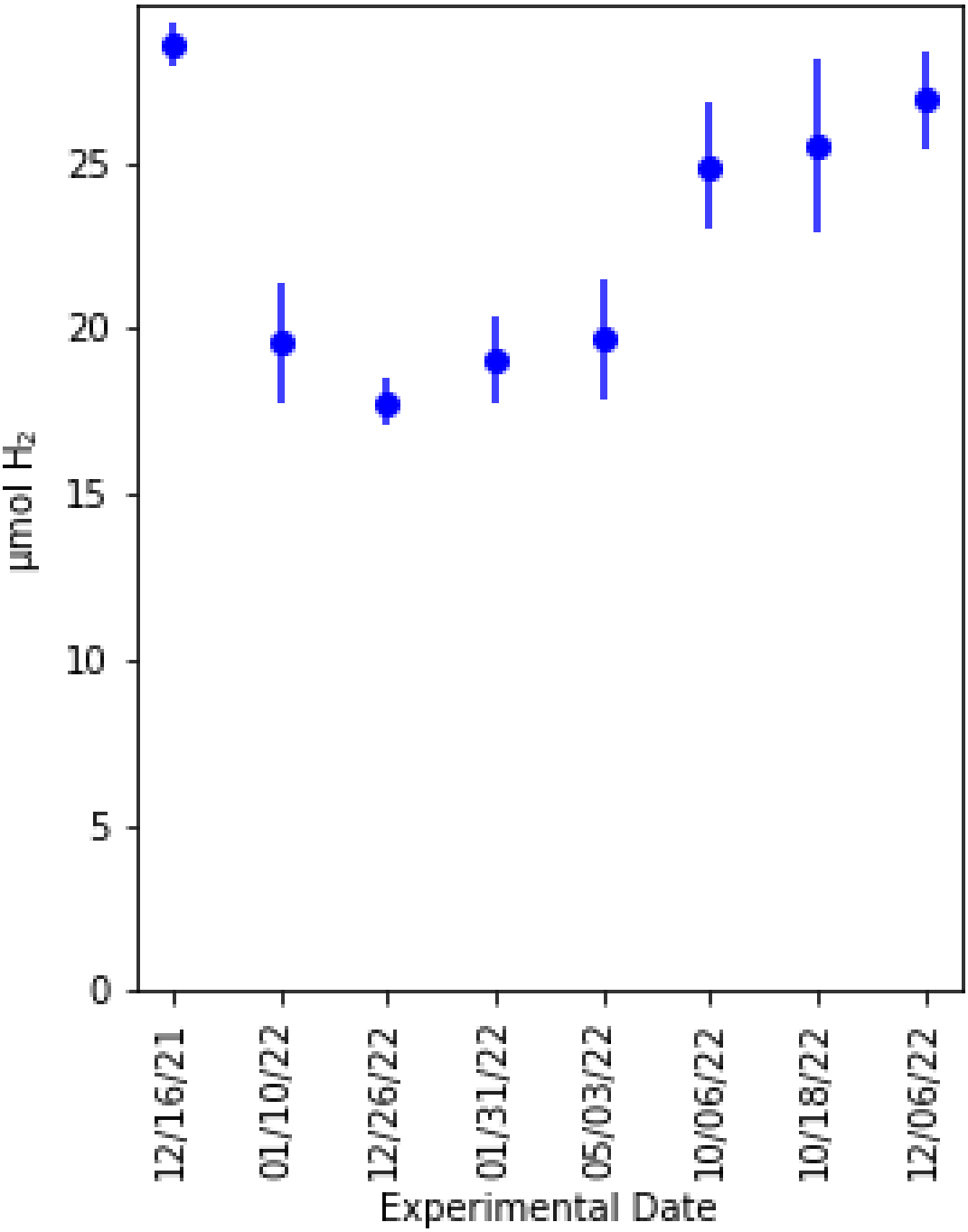


4.9 SI Figure on internal standards

The data for this study was taken from 8 different plates. External sources of variance are accounted for by the use of 6 wells with a molecular internal standard on every experimental plate. The internal standard variation from plate to plate accounts for factors such as detection tape sensitivity, light intensity, humidity, or temperature. The average internal standard activity was used to normalize the data on each plate respectively and is plotted here for each experimental trial.

```
dfint = pd.read_json('data/intstand.json')
dates = ['12/16/21','01/10/22', '12/26/22','01/31/22', '05/03/22','10/06/22', '10/18/22', '12/06/22']

mean_act =[]
std_act = []
for i in dfint.directory.unique():
  df_temp = dfint[dfint.directory == i]
  mean_act.append(df_temp.umolH_max.mean(numeric_only = True))
  std_act.append(df_temp.umolH_max.std(numeric_only = True))

fig, ax = plt.subplots(figsize = (4, 5))
ax.errorbar(dates, mean_act, yerr = std_act, fmt = 'bo')
ax.set_xticklabels(dates, rotation = 'vertical')
ax.set_yticks(np.arange(0, 30, 5))
ax.set_ylabel(u'\u03bcmol H$_2$')
ax.set_xlabel('Experimental Date')
fig.tight_layout()
fig.savefig(f'figures/si-intstand.png', dpi = 300)

Figure('./figures/si-intstand.png',
       attributes=(('org', ':width 500'),
                   ('latex', ':placement [H] :width 6in')))
```

/var/folders/3q/ht_2mtk52hl7ydxrcr87z2gr0000gn/T/ipykernel_6969/2727707073.py:8: UserWarning: FixedFormatter should only be used together with FixedLocator ax.set_xticklabels(dfint.paths, rotation=90)

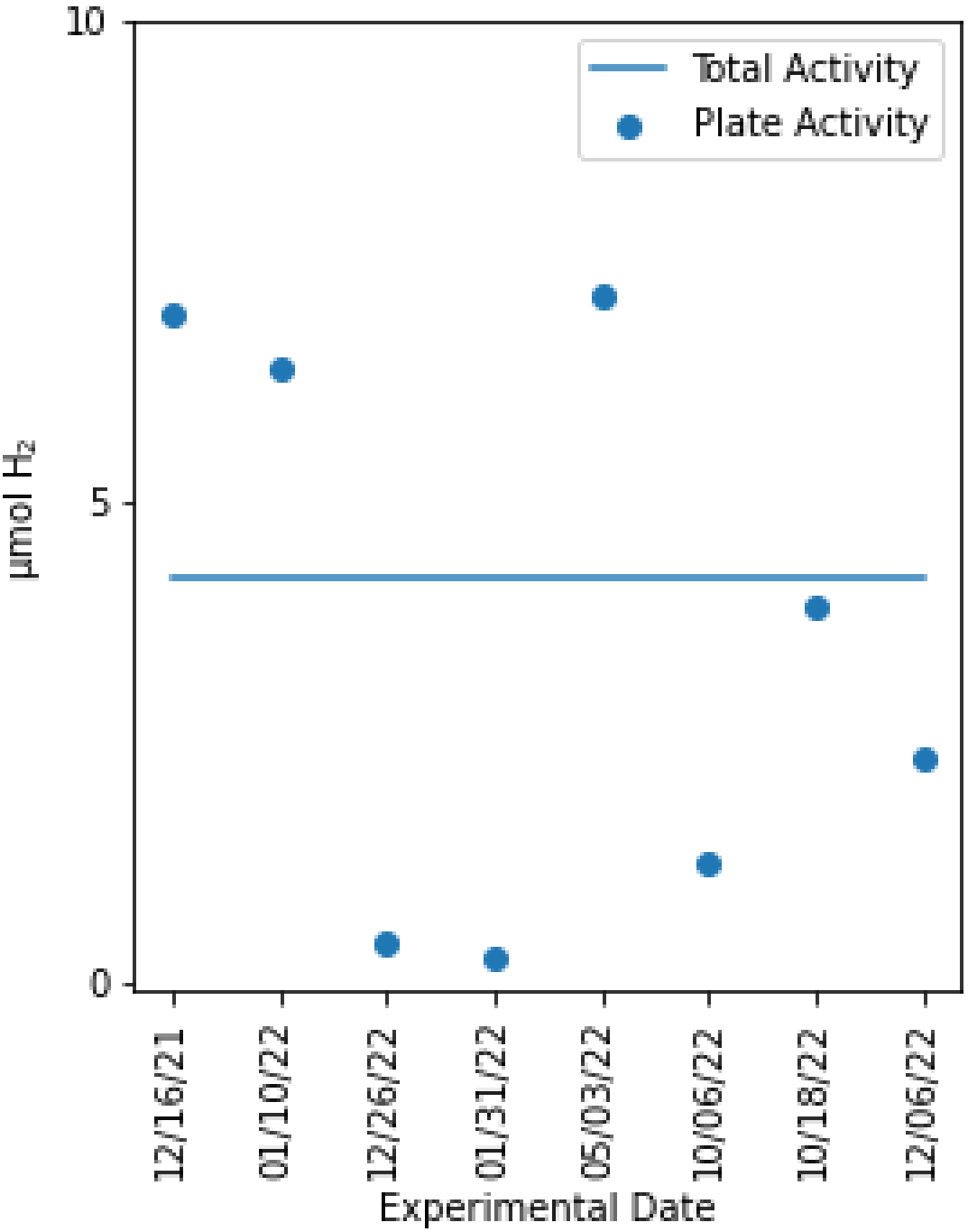


4.10 SI Figure on Cu normalization

The data for this study was taken from 8 different plates. Due to the *in-situ* nature of the nanoparticle synthesis, variance due to nanoparticle growth must be accounted for. Since Cu consistently form stable nanoparticles and its H2 activity is independent of concentration, the average pure Cu activity was used as a normalization factor. The average Cu activity for each plate is plotted here for each experimental plate run.

```
mean_act =[]
std_act = []
for i in cuMon.directory.unique():
  df_temp = cuMon[cuMon.directory == i]
  mean_act.append(df_temp.umolH_max.mean(numeric_only = True))
  #std_act.append(df_temp.umolH_max.std(numeric_only = True))

fig, ax = plt.subplots(figsize = (4, 5))
#ax.errorbar(dates, mean_act, yerr = std_act, fmt = 'bo')
ax.scatter(dates, mean_act)
ax.plot(dates, np.repeat(cuMon.umolH_max.mean(), len(dates)))
ax.set_xticklabels(dates, rotation = 'vertical')
ax.set_yticks(np.arange(0, 11, 5))
ax.set_ylabel(u'\u03bcmol H$_2$')
ax.set_xlabel('Experimental Date')
ax.legend(['Total Activity', 'Plate Activity'])

plt.tight_layout()
plt.savefig(f'figures/si-cunorm.png', dpi = 300)

Figure('./figures/si-cunorm.png',
       attributes=(('org', ':width 500'),
                   ('latex', ':placement [H] :width 6in')))
```

/var/folders/3q/ht_2mtk52hl7ydxrcr87z2gr0000gn/T/ipykernel_6969/2727707073.py:8: UserWarning: FixedFormatter should only be used together with FixedLocator ax.set_xticklabels(dfint.paths, rotation=90)


10
5
0
µmol H₂
Total Activity
Plate Activity
12/16/21
01/10/22
12/26/22
01/31/22
05/03/22
10/06/22
10/18/22
12/06/22
Experimental Date