Electronic Supplementary Material (ESI) for Digital Discovery.

The Materials Experiment Knowledge Graph: Supporting Information

Michael J. Statt[1,*], Brian A. Rohr[1,*], Dan Guevarra[2,3], Ja'Nya Breeden[3], Santosh K. Suram[4], and John M. Gregoire[2,3,*]

[1]Modelyst LLC, Palo Alto, CA 94306
[2]Division of Engineering and Applied Science, California Institute of Technology, Pasadena, CA 91125
[3]Liquid Sunlight Alliance, California Institute of Technology, Pasadena, CA 91125
[4]Toyota Research Institute, Los Altos, CA 94022
[*]E-mail: brian.rohr@modelyst.io, michael.statt@modelyst.io, gregoire@caltech.edu

Node and edge types

Entity Type	Count
Analysis	3,980,832
AnalysisDetails	7
Collection	3,612
Element	46
Process	23,591
ProcessData	5,425,961
ProcessDetail	13,434
Sample	12,159,999
SampleProcess	30,656,471
pH	15

Table S1. Summary of entity count in the Neo4j graph database. Each row represents the count of a unique entity type.

Relationship Type	Count
ANALYSIS_DETAILS	3,980,832
CONTAINS	3,527,845
CollectionSample	12,159,990
NEXT	18,590,083
PH	6,267
PROCESS	30,656,471
PROCESS_DETAIL	23,488
ProcessDataAnalysis	4,050,703
SAMPLE	3,065,6471
SampleProcessProcessData	7,777,908

Table S2. Summary of relationship count in the Neo4j graph database. Each row represents the count of a unique relationship type.

Computational Methods

The computational resources used to execute the use cases are as follows:

The extract, transform, load (ETL) process was carried out using a python library called DBgen (https://github.com/modelyst/dbgen), which was specifically designed to instantiate complicated, scientific data pipelines. PostgreSQL (https://www.postgresql.org/) was used to create the SQL database, and the Neo4j community edition (https://neo4j.com/) was used to create the graph database. The process of migrating the data from the SQL database to the graph database was done using a python library

Table S3. Resources used for each task

Task	Resource Used
Extracting, transforming, and loading (ETL) the filesystem data into the SQL database	DBgen
Hosting the SQL database	PostgreSQL, AWS EC2, Docker
Migrating the SQL database to the graph database	PG4J
Hosting the graph database	Neo4j, AWS EC2, Docker
Querying graph database and processing data for design of experiments use cases	Python, Jupyter notebook server (local)

called PG4J (https://github.com/modelyst/pg4j), which is capable of migrating any PostgreSQL database to Neo4j. For query timing, the SQL database and the graph database were run in docker containers on AWS EC2. Specifically, the EC2 instance was a t2.xlarge, and the docker images were postgres:14 and neo4j:5.5 for the SQL and graph databases, respectively. Cypher queries and data processing for the design of experiments use cases were executed in Jupyter notebooks running on a local JupyterHub server (Intel i9-11900K, 64 GB RAM). The computational methods are summarized in table S3.

Design of experiments use case

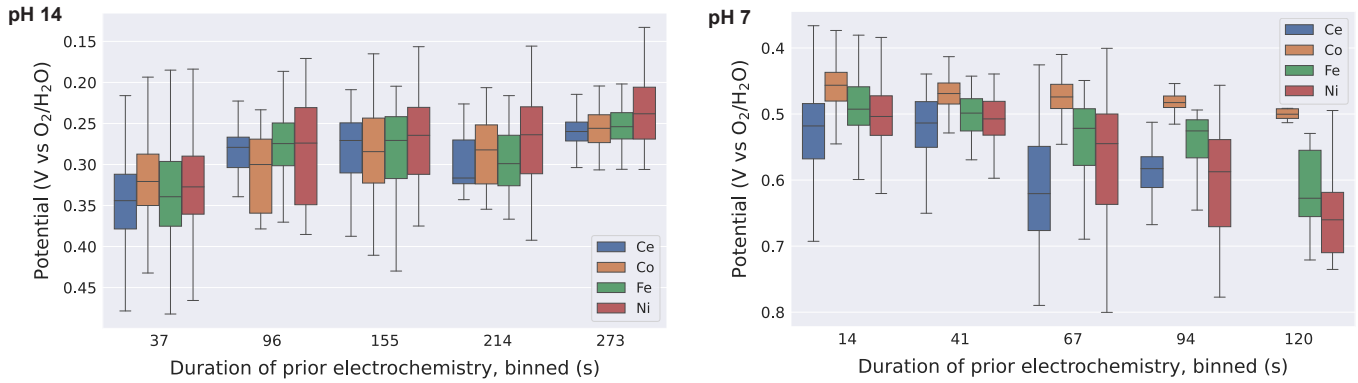

Figure S1. A summary of OER activity with 4982 measurements in pH 14 electrolyte (left) and 6524 measurements in pH 7 electrolyte (right). The catalysts with at least 70% concentration of the given element (Ce, Co, Fe, or Ni) are then grouped by 5 bins of total duration of electrochemical operation prior to the activity measurement. The catalyst overpotential at 3 mA/cm^2 is shown as an inverted vertical axis so that higher activity is shown as higher position in the figure. The consistent upward trend with increasing duration demonstrates a universal condition in pH 13 electrolyte, which is not observed in pH 7 electrolyte.

Table S4. Candidate composition spaces proposed by automated design of experiment for measurement in pH 3 electrolyte, sorted by 5th percentile overpotential at 10 mA/cm^2 from pH 7 measurement.

Composition space	Eta (V) 3 mA/cm^2	Eta (V) 10 mA/cm^2	plate ID	sample count
Co-Fe-La-Ni	0.118	0.142	1740	1365.0
Co-Fe-La-Ni	0.118	0.142	1723	1365.0
Ce-Co-Fe-Ni	0.191	0.175	1749	30.0
Ce-Co-Fe-Ni	0.191	0.175	1751	30.0
Ce-Co-Fe-Ni	0.191	0.175	1754	30.0
Ce-Co-Fe-Ni	0.191	0.175	1755	30.0
Ce-Co-Fe-Ni	0.191	0.175	1756	30.0
Ce-Co-Fe-Ni	0.191	0.175	1762	388.0
Ce-Co-Fe-Ni	0.191	0.175	1763	388.0
Ce-Co-Fe-Ni	0.191	0.175	1774	30.0
Ce-Co-Fe-Ni	0.191	0.175	2486	388.0
Ce-Co-Fe-Ni	0.191	0.175	2487	388.0
Ce-Co-Fe-Ni	0.191	0.175	2488	388.0
Ce-Co-Fe-La-Ni	0.163	0.184	1757	1683.0
Ce-Co-La-Ni	0.435	0.483	1721	1398.0
Ce-Co-La-Ni	0.435	0.483	1720	1398.0
Ce-Co-La-Ni	0.435	0.483	2369	30.0
Ce-Co-La-Ni	0.435	0.483	2367	30.0
Ce-Co-La-Ni	0.435	0.483	1722	1398.0
Ce-Co-La-Ni	0.435	0.483	1750	30.0
Ce-Co-La-Ni	0.435	0.483	1719	1398.0
Ce-Co-La-Ni	0.435	0.483	1829	1398.0
Co-Cu-Fe-Mn-Sn-Ta	0.556	0.631	3673	3.0
Co-Cu-Fe-Mn-Sn-Ta	0.556	0.631	3859	63.0
Co-Cu-Fe-Mn-Sn-Ta	0.556	0.631	3865	63.0
Co-Cu-Fe-Mn-Sn-Ta	0.556	0.631	3866	63.0
Co-Cu-Fe-Mn-Sn-Ta	0.556	0.631	3867	63.0
Co-Cu-Fe-Mn-Sn-Ta	0.556	0.631	3870	63.0
Ce-Co-Ni-Zn	0.521	0.634	2532	1597.0

Creation of Database Subsets

To investigate the scaling of query times in both the SQL and graph databases, we created three smaller versions of the original database. The first database fragment was created by removing the last half of the rows in the sample-process table, ordered by their process timestamps. We then deleted all rows in other tables that were no longer linked to a sample-process. This process was repeated two more times to create two additional database fragments, with 3/4 and 7/8 of the rows in the sample-process table deleted. Each fragment was migrated to Neo4j using the tools described above, resulting in a series of MPS-style and MEKG-style databases that share the same information and contain 1/8, 1/4, and 1/2 of the number of Sample-Processes in the full MPS and MEKG databases.

Query 4 in Cypher and SQL

Below is the code for Query 4 in Cypher:

```
MATCH
 path=(sp1:SampleProcess)-[:NEXT]->(sp2)-[:NEXT]->(sp3)-[:NEXT]->(sp4)-[:NEXT]->(sp5),
 (a1:Analysis)<--(pda1:ProcessData)<--(sp1)-->(p1:Process)-->(pd1:ProcessDetail),
 (a2:Analysis)<--(pda2:ProcessData)<--(sp2)-->(p2:Process)-->(pd2:ProcessDetail),
 (a3:Analysis)<--(pda3:ProcessData)<--(sp3)-->(p3:Process)-->(pd3:ProcessDetail),
 (a4:Analysis)<--(pda4:ProcessData)<--(sp4)-->(p4:Process)-->(pd4:ProcessDetail),
 (a5:Analysis)<--(pda5:ProcessData)<--(sp5)-->(p5:Process)-->(pd5:ProcessDetail)
WHERE
pd1.technique STARTS WITH 'CA'
 AND pd2.technique STARTS WITH 'CA'
 AND pd3.technique STARTS WITH 'CA'
 AND pd4.technique STARTS WITH 'CA'
 AND pd5.technique STARTS WITH 'CV'
 AND a5.name = 'CV_FOMS_standard'
 AND apoc.convert.fromJsonMap(a1.output)['I.A_ave'] > 1e-7
 AND apoc.convert.fromJsonMap(a2.output)['I.A_ave'] > 1e-8
 AND apoc.convert.fromJsonMap(a3.output)['I.A_ave'] > 1e-9
 AND apoc.convert.fromJsonMap(a4.output)['I.A_ave'] > 1e-10
 AND apoc.convert.fromJsonMap(a5.output)['I.A_max'] > 1e-6
 AND apoc.convert.fromJsonMap(pd1.parameters)['electrolyte'] CONTAINS 'NaOH'
 AND apoc.convert.fromJsonMap(pd2.parameters)['electrolyte'] CONTAINS 'NaOH'
 AND apoc.convert.fromJsonMap(pd3.parameters)['electrolyte'] CONTAINS 'NaOH'
 AND apoc.convert.fromJsonMap(pd4.parameters)['electrolyte'] CONTAINS 'NaOH'
 AND apoc.convert.fromJsonMap(pd5.parameters)['electrolyte'] CONTAINS 'NaOH'
RETURN count(path)
```

Below is the code for Query 4 in SQL:

```
with your_table as (
select
 sp.sample_id,
 p1."timestamp",
 p1."ordering",
 pd1.technique,
 pd1.parameters,
 a."name",
 a."output"
from
 sample_process sp
join process p1 on
 sp.process_id = p1.id
left join process_detail pd1 on
 p1.process_detail_id = pd1.id
left join sample_process_process_data sppd on
 sppd.sample_process_id = sp.id
left join process_data pd on
 sppd.process_data_id = pd.id
left join process_data_analysis pda on
 pda.process_data_id = pd.id
left join analysis a on
 pda.analysis_id = a.id
where
 sp.sample_id in (
```

```
 select
  sp3.sample_id
 from
  sample_process sp3
 join process p3 on
  sp3.process_id = p3.id
 join process_detail pd3 on
  p3.process_detail_id = pd3.id
 where
  pd3.technique like 'CV%'
 )
 and sp.sample_id in (
 select
  sp2.sample_id
 from
  sample_process sp2
 join process p2 on
  sp2.process_id = p2.id
 join process_detail pd2 on
  p2.process_detail_id = pd2.id
 where
  pd2.technique like 'CA%'
 group by
  sp2.sample_id
 having
  count(*) >= 4)
),
filtered_labels as (
select
 sample_id
from
 your_table
group by
 sample_id
having
 COUNT(*) >= 5
),
sequenced_data as (
select
 t1.sample_id,
 t1.Timestamp,
 t1.ordering,
 t1.technique,
 t1.parameters,
 t1."name",
 t1."output",
 row_number() over (partition by t1.sample_id
order by
 t1.Timestamp,
 t1.ordering) as RowNum
from
 your_table t1
inner join
 filtered_labels fl on
 t1.sample_id = fl.sample_id
```

```
),
json_agg_data as (
select
 sd1.sample_id,
 json_agg(sd1.technique) over (partition by sd1.sample_id
order by
 sd1.RowNum rows between current row and 4 following) as technique_seq,
 json_agg(sd1.parameters) over (partition by sd1.sample_id
order by
 sd1.RowNum rows between current row and 4 following) as parameters_seq,
 json_agg(sd1.name) over (partition by sd1.sample_id
order by
 sd1.RowNum rows between current row and 4 following) as name_seq,
 json_agg(sd1.output) over (partition by sd1.sample_id
order by
 sd1.RowNum rows between current row and 4 following) as output_seq,
 COUNT(*) over (partition by sd1.sample_id
order by
 sd1.RowNum rows between current row and 4 following) as technique_seq_count
from
 sequenced_data sd1
)
select
 count(*)
 -- sample_id
 -- technique_seq,
 -- parameters_seq,
 -- name_seq,
 -- output_seq
from
 json_agg_data
where
 technique_seq_count = 5
 and technique_seq->>0 like 'CA%'
 and technique_seq->>1 like 'CA%'
 and technique_seq->>2 like 'CA%'
 and technique_seq->>3 like 'CA%'
 and technique_seq->>4 like 'CV%'
 and name_seq->>4 = 'CV_FOMS_standard'
 and (output_seq->0->>'I.A_ave')::float > 1e-7
 and (output_seq->1->>'I.A_ave')::float > 1e-8
 and (output_seq->2->>'I.A_ave')::float > 1e-9
 and (output_seq->3->>'I.A_ave')::float > 1e-10
 and (output_seq->4->>'I.A_max')::float > 1e-6
 and parameters_seq->0->>'electrolyte' like '%NaOH%'
 and parameters_seq->1->>'electrolyte' like '%NaOH%'
 and parameters_seq->2->>'electrolyte' like '%NaOH%'
 and parameters_seq->3->>'electrolyte' like '%NaOH%'
 and parameters_seq->4->>'electrolyte' like '%NaOH%'
```