

Querying the Catalysis Hub DFT Database

1. Converts the bulks encoded inside of "GPT_CO2_Modified.csv" into queryable permutations
2. Retrieves all available adsorption energies of selected adsorbates (ADSORBATES_LIST) on bulk (catalyst composition) permutations and stores them in 'Cat_hub_all_energies.csv'
3. Parses 'Cat_hub_all_energies.csv' and retrieves the lowest (most negative) value for each bulk-adsorbate pair on low Miller index surfaces

User-Defined Variables

```
In [ ]: CSV_PATH = "GPT_CO2_Modified.csv"
        ADSORBATES_LIST = ["H2", "O2", "H", "O", "CH", "CO2", "H2O", "C", "CO", "C2H2", "CH3OH", "
        N_RESULTS = 1000 # GraphQL page size
        MAX_NUM_ELEMENTS = 4 # rows with >4 elements are skipped
        SIG_FIGS = 4 # number of eV sig figs to keep
        INDIVIDUAL_BULK = None # specify an individual bulk permutation to test instead of
```

Imports

```
In [ ]: import itertools, requests, pandas as pd, numpy as np
        from ase.data import chemical_symbols
        from tqdm.auto import tqdm
        from typing import Dict, List, Tuple
```

```
In [ ]: API_ROOT = "https://api.catalysis-hub.org/graphql"
```

Helper Function for Making a Query to Catalysis Hub

```
In [ ]: def run_query(query: str):
        r = requests.post(API_ROOT, json={"query": query})
        r.raise_for_status()
        payload = r.json()
        if "errors" in payload:
            raise RuntimeError(payload["errors"])
        return payload["data"]
```

Helper Functions for Formatting the Query String

```
In [ ]: def _gas(s):
        return f"{s.strip()}gas"

        def _star(s):
            return f"{s.strip()}star"
```

```
In [ ]: def build_adsorption_query(bulk: str, ads: str, n: int = 1000) -> str:
        fields = (
            "id Equation reactionEnergy facet reactants products "
            "dftCode dftFunctional pubId"
        ).split()
        return f"""
        {{
            reactions(first:{n},
                reactants:"{_gas(ads)}",
                products:"{_star(ads)}",
                surfaceComposition:"{bulk}") {{
```

```
edges {{ node {{ { ' '.join(fields)} }} }}
}}
}}"""
```

Function for Converting Encoded Bulks into Queryable Element Permutations

```
In [ ]: def permutation_map_from_csv(csv_path: str, max_num_elements: int) -> Tuple[Dict[str, List[str]], List[str]]:
    df = pd.read_csv(csv_path)
    valid = set(chemical_symbols[1:])
    elem_cols = [c for c in df.columns if c in valid]

    df_unique = df[elem_cols].drop_duplicates().reset_index(drop=True)
    perm_map, skipped = {}, []

    for _, row in df_unique.iterrows():
        present = [e for e in elem_cols if row[e] == 1]
        if not present:
            continue

        alpha = "".join(sorted(present))
        if len(present) > max_num_elements:
            skipped.append(alpha)
            continue # do not generate permutations if more than max num of elements present

        perm_map[alpha] = ["".join(p) for p in itertools.permutations(present)]

    return perm_map, skipped
```

Function for Making Queries

```
In [ ]: def fetch_adsorption_with_progress(
    perm_map: Dict[str, List[str]],
    INDIVIDUAL_BULK,
    skipped_alpha: List[str],
    adsorbates: List[str],
    n_results: int = 1000,
) -> Tuple[pd.DataFrame, pd.DataFrame]:

    total_queries = sum(len(perms)*len(adsorbates) for perms in perm_map.values())
    overall_bar = tqdm(total=total_queries,
                        desc="Overall GraphQL calls",
                        dynamic_ncols=True)

    raw_frames, summary_rows = [], []

    if INDIVIDUAL_BULK != None:
        perm_map = Dict[INDIVIDUAL_BULK, List[INDIVIDUAL_BULK]]

    for alpha_key, perms in perm_map.items():
        for ads in adsorbates:

            inner = tqdm(total=len(perms),
                        desc=f"{alpha_key} -> {ads}",
                        leave=False,
                        dynamic_ncols=True)

            best_E = best_bulk = None

            for perm in perms:
                inner.update(1)
                overall_bar.update(1)

            edges = run_query(build_adsorption_query(perm, ads, n_results))
```

```

        )["reactions"]["edges"]
    if not edges:
        continue

    df_tmp = pd.DataFrame(edge["node"] for edge in edges)
    df_tmp["bulk_db"] = perm
    df_tmp["adsorbate"] = ads
    raw_frames.append(df_tmp)

    energies = pd.to_numeric(df_tmp["reactionEnergy"], errors="coerce")
    best_E = float(energies.min(skipna=True))
    best_bulk = perm
    break # stop at first successful permutation

    inner.close()
    summary_rows.append(dict(
        bulk_query = alpha_key,
        bulk_db = best_bulk,
        adsorbate = ads,
        E_ads_eV = best_E
    ))

overall_bar.close()

for alpha_key in skipped_alpha:
    for ads in adsorbates:
        summary_rows.append(dict(
            bulk_query = alpha_key,
            bulk_db = None,
            adsorbate = ads,
            E_ads_eV = None
        ))

df_raw = pd.concat(raw_frames, ignore_index=True) if raw_frames else pd.DataFrame()
df_search_results = pd.DataFrame(summary_rows)
return df_raw, df_search_results

```

Run Statements

```

In [ ]: perm_map, skipped_keys = permutation_map_from_csv(
        CSV_PATH, max_num_elements=MAX_NUM_ELEMENTS)

df, df_search_results = fetch_adsorption_with_progress(
    perm_map, INDIVIDUAL_BULK, skipped_keys, ADSORBATES_LIST, n_results=N_RESULTS)

print("Done. Preview:")
df_search_results.head()

```

Storing Retrieved Energies

```

In [ ]: df.to_csv('Cat_hub_all_energies.csv', index=False)

In [ ]: df_search_results["E_ads_eV"] = df_search_results["E_ads_eV"].apply(lambda x: None if (x is
df_search_results

In [ ]: df_search_results["E_ads_eV"].isna().sum()

In [ ]: df_search_results.to_csv('Cat_hub_search_results.csv', index=False)

In [ ]:

```