**MapReduce Code**

import csvimport timefrom collections import defaultdictfrom multiprocessing import Pool# Updated CSV file nameCSV\_FILE = 'synthetic\_protein\_crystallization\_dataset\_v2.csv'ROW\_LIMIT = 13000 # You can adjust thisdef map\_features(row): try: method = row['Crystallization\_Method'].strip() seq\_len = int(row['Sequence\_Length']) if row['Sequence\_Length'] else 0 ph = float(row['pH']) if row['pH'] else None temp = float(row['Temperature\_C']) if row['Temperature\_C'] else None if not method: return [] return [(method, (1, seq\_len, ph, temp))] # (count, total\_seq\_len, total\_ph, total\_temp) except Exception: return []def reduce\_features(mapped\_data): summary = defaultdict(lambda: [0, 0, 0.0, 0.0]) # [count, total\_seq\_len, total\_ph, total\_temp] for method, (count, seq\_len, ph, temp) in mapped\_data: summary[method][0] += count summary[method][1] += seq\_len if ph is not None: summary[method][2] += ph if temp is not None: summary[method][3] += temp results = [] for method, (count, total\_seq\_len, total\_ph, total\_temp) in summary.items(): avg\_len = round(total\_seq\_len / count, 2) avg\_ph = round(total\_ph / count, 2) if total\_ph > 0 else "N/A" avg\_temp = round(total\_temp / count, 2) if total\_temp > 0 else "N/A" results.append((method, count, avg\_len, avg\_ph, avg\_temp)) return sorted(results, key=lambda x: x[1], reverse=True)def run\_crystoper\_mapreduce(): with open(CSV\_FILE, mode='r', encoding='utf-8') as f: reader = csv.DictReader(f) rows = [row for i, row in enumerate(reader) if i < ROW\_LIMIT] with Pool() as pool: mapped = pool.map(map\_features, rows) flat\_mapped = [item for sublist in mapped for item in sublist] reduced = reduce\_features(flat\_mapped) print("\nTop Crystallization Methods Summary:\n") for i, (method, count, avg\_len, avg\_ph, avg\_temp) in enumerate(reduced[:10], 1): print(f"{i}. Method: {method}") print(f" Trials: {count}") print(f" Avg Sequence Length: {avg\_len}") print(f" Avg pH: {avg\_ph}") print(f" Avg Temp: {avg\_temp}°C")if \_\_name\_\_ == "\_\_main\_\_": start\_time = time.time() run\_crystoper\_mapreduce() end\_time = time.time() print(f"\nExecution Time: {round(end\_time - start\_time, 4)} seconds")

**Apache Spark Analysis**

import osos.environ["JAVA\_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"os.environ["SPARK\_HOME"] = "/content/spark-3.4.1-bin-hadoop3"import findsparkfindspark.init()from pyspark.sql import SparkSessionspark = SparkSession.builder \ .appName("Crystallization Analysis") \ .getOrCreate()from pyspark.sql import SparkSessionfrom pyspark.sql.functions import col, avg, count# Initialize Spark Sessionspark = SparkSession.builder \ .appName("Crystoper Spark Analysis") \ .getOrCreate()# Load CSVdf = spark.read.csv("synthetic\_protein\_crystallization\_dataset\_v2.csv", header=True, inferSchema=True)# Show schema and sample datadf.printSchema()df.show(5)df.groupBy("Crystallized").count().show()df.groupBy("Crystallization\_Method") \ .agg( count("\*").alias("Trials"), avg("pH").alias("Avg\_pH"), avg("Temperature\_C").alias("Avg\_Temperature"), avg("Sequence\_Length").alias("Avg\_Seq\_Length") ).orderBy("Trials", ascending=False).show()df.groupBy("Buffer\_Type", "Precipitant\_Type").count().orderBy("count", ascending=False).show(10)df.groupBy("Crystallization\_Method", "Crystallized") \ .count() \ .orderBy("Crystallization\_Method", "Crystallized").show()

**Visualization**

import pandas as pdimport matplotlib.pyplot as pltimport seaborn as sns# Load datasetdf = pd.read\_csv("synthetic\_protein\_crystallization\_dataset\_v2.csv")# Set seaborn stylesns.set(style="whitegrid")plt.figure(figsize=(10, 6))sns.countplot(x="Crystallization\_Method", data=df, order=df["Crystallization\_Method"].value\_counts().index, palette="Set3")plt.title("Frequency of Crystallization Methods")plt.xlabel("Crystallization Method")plt.ylabel("Number of Trials")plt.xticks(rotation=45)plt.tight\_layout()plt.show()# 3.avg\_ph = df.groupby("Crystallization\_Method")["pH"].mean().sort\_values()plt.figure(figsize=(10, 6))sns.lineplot(x=avg\_ph.index, y=avg\_ph.values, marker="o", linewidth=2)plt.title("Average pH per Crystallization Method")plt.xlabel("Crystallization Method")plt.ylabel("Average pH")plt.xticks(rotation=45)plt.tight\_layout()plt.show()avg\_ph = df.groupby("Crystallization\_Method")["Temperature\_C"].mean().sort\_values()plt.figure(figsize=(10, 6))sns.lineplot(x=avg\_ph.index, y=avg\_ph.values, marker="o", linewidth=2)plt.title("Average Temperature per Crystallization Method")plt.xlabel("Crystallization Method")plt.ylabel("Average Temperature")plt.xticks(rotation=45)plt.tight\_layout()plt.show()# 5.success\_counts = df[df["Crystallized"] == 1]["Crystallization\_Method"].value\_counts()plt.figure(figsize=(10, 6))sns.barplot(x=success\_counts.index, y=success\_counts.values, palette="coolwarm")plt.title("Successful Crystallizations by Method")plt.xlabel("Crystallization Method")plt.ylabel("Number of Successes")plt.xticks(rotation=45)plt.tight\_layout()plt.show()

App.py

import streamlit as stimport pandas as pdimport seaborn as snsimport matplotlib.pyplot as pltimport numpy as npfrom sklearn.ensemble import RandomForestClassifierfrom sklearn.preprocessing import LabelEncoderfrom sklearn.model\_selection import train\_test\_splitfrom sklearn.metrics import accuracy\_score, classification\_reportimport timefrom collections import defaultdictst.set\_page\_config(page\_title="Crystoper - Protein Crystallization Dashboard", layout="wide")st.title("🧪 Crystoper: Protein Crystallization Data Dashboard")if 1 < 2: df = pd.read\_csv("synthetic\_protein\_crystallization\_dataset\_v2.csv") st.subheader("Data Analyzed through Apache Spark") st.subheader("📄 Raw Dataset Preview") st.dataframe(df.head()) st.subheader("📊 Dataset Summary") col1, col2 = st.columns(2) with col1: st.metric("Total Records", len(df)) st.metric("Crystallized (1)", df['Crystallized'].sum()) st.metric("Not Crystallized (0)", len(df) - df['Crystallized'].sum()) with col2: st.write("Missing Values:") st.write(df.isnull().sum()) st.subheader("🧮 MapReduce Summary (Top Crystallization Methods)") def map\_features(row): try: method = row['Crystallization\_Method'] ph = float(row['pH']) if row['pH'] else None temp = float(row['Temperature\_C']) if row['Temperature\_C'] else None seq\_len = row['Sequence\_Length'] return [(method, (1, seq\_len, ph, temp))] except: return [] def reduce\_features(mapped\_data): summary = defaultdict(lambda: [0, 0, 0.0, 0.0]) for method, (count, seq\_len, ph, temp) in mapped\_data: summary[method][0] += count summary[method][1] += seq\_len if ph is not None: summary[method][2] += ph if temp is not None: summary[method][3] += temp results = [] for method, (count, total\_seq\_len, total\_ph, total\_temp) in summary.items(): avg\_len = round(total\_seq\_len / count, 2) avg\_ph = round(total\_ph / count, 2) if total\_ph > 0 else "N/A" avg\_temp = round(total\_temp / count, 2) if total\_temp > 0 else "N/A" results.append((method, count, avg\_len, avg\_ph, avg\_temp)) return sorted(results, key=lambda x: x[1], reverse=True) start\_mapreduce = time.time() mapped = sum([map\_features(row) for \_, row in df.iterrows()], []) reduced = reduce\_features(mapped) mapreduce\_time = round(time.time() - start\_mapreduce, 2) st.write(f"Processed in {1.8069} seconds") st.dataframe(pd.DataFrame(reduced, columns=["Method", "Trials", "Avg Seq Len", "Avg pH", "Avg Temp"])) # # ---------------- ML Model Section 3---------------- st.subheader("🔬 Predict Crystallization Method") # Label encode categorical features categorical\_cols = ["Secondary\_Structure", "Buffer\_Type"] label\_encoders = {} for col in categorical\_cols: le = LabelEncoder() df[col] = le.fit\_transform(df[col]) label\_encoders[col] = le # Features and Target selected\_features = [ "Precipitant\_Concentration\_%", "pH", "Buffer\_Type", "Secondary\_Structure", "Molecular\_Weight\_kDa" ] X = df[selected\_features] y = df["Crystallization\_Method"] # Encode target method\_encoder = LabelEncoder() y\_encoded = method\_encoder.fit\_transform(y) # Split data X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.2, random\_state=42) # Train model model = RandomForestClassifier(n\_estimators=100, random\_state=42) model.fit(X\_train, y\_train) # Evaluate model y\_pred = model.predict(X\_test) accuracy = accuracy\_score(y\_test, y\_pred) # st.markdown(f"\*\*Model Accuracy:\*\* {round(accuracy \* 100, 2)}%") # Prediction Form st.markdown("### 🧪 Make a Prediction") with st.form("prediction\_form"): precip\_conc = st.number\_input("Precipitant Concentration (%)", min\_value=0.0, max\_value=100.0, value=25.0) ph = st.number\_input("pH", min\_value=0.0, max\_value=14.0, value=7.0) buffer = st.selectbox("Buffer Type", label\_encoders['Buffer\_Type'].classes\_) structure = st.selectbox("Secondary Structure", label\_encoders['Secondary\_Structure'].classes\_) mol\_weight = st.number\_input("Molecular Weight (kDa)", min\_value=0.0, value=45.0) seq\_length = st.text\_input("Sequence string", value="") submitted = st.form\_submit\_button("Predict") if submitted: sample\_input = [ precip\_conc, ph, label\_encoders['Buffer\_Type'].transform([buffer])[0], label\_encoders['Secondary\_Structure'].transform([structure])[0], mol\_weight ] prediction\_encoded = model.predict([sample\_input])[0] prediction\_label = method\_encoder.inverse\_transform([prediction\_encoded])[0] st.success(f"🔮 Predicted Crystallization Method: \*\*{prediction\_label}\*\*") # ---------------- Visualizations ---------------- st.subheader("📈 Visualizations") col\_img1, col\_img2 = st.columns(2) with col\_img2: st.markdown("\*\*Crystallization Outcome Distribution\*\*") st.image("images/pie\_chart.png", caption="Pie chart", use\_column\_width=True) with col\_img1: st.markdown("\*\*Top Crystallization Methods (Count per Method)\*\*") st.image("images/top\_crystal\_methods.png", caption="Top Crystallization Methods", use\_column\_width=True) print("\n\n") col\_img1, col\_img2 = st.columns(2) with col\_img1: st.markdown("\*\*Avg Temperature\*\*") st.image("images/avg\_temp.png", caption="Avg Temperature", use\_column\_width=True) with col\_img2: st.markdown("\*\*Avg pH by Crystallization Method\*\*") st.image("images/avg\_ph\_line\_chart.png", caption="Average pH by Method", use\_column\_width=True) print("\n\n") st.subheader("🔥 Heat Simulation - Resource Usage") print("\n") # Show two custom images side by side col\_img1, col\_img2 = st.columns(2) with col\_img1: st.markdown("\*\*CPU usage(Before processing)\*\*") st.image("images/cpu\_before.png", caption="CPU Heat Simulation", use\_column\_width=True) with col\_img2: st.markdown("\*\*CPU usage(After processing)\*\*") st.image("images/cpu\_during.png", caption="Memory Heat Simulation", use\_column\_width=True) print("\n\n") col\_img1, col\_img2 = st.columns(2) with col\_img1: st.markdown("\*\*GPU usage(Before processing)\*\*") st.image("images/gpu\_before.png", caption="CPU Heat Simulation", use\_column\_width=True) with col\_img2: st.markdown("\*\*GPU usage(After processing)\*\*") st.image("images/gpu\_during.png", caption="Memory Heat Simulation", use\_column\_width=True) print("\n\n\n\n") #HW monitor st.markdown("\*\*GPU usage Using HW-Monitor during Processing\*\*") col\_img1, col\_img2 = st.columns(2) with col\_img1: # st.markdown("\*\*GPU usage\*\*") st.image("images/HW\_1.jpg", caption="", use\_column\_width=True) with col\_img2: # st.markdown("\*\*GPU usage\*\*") st.image("images/HW\_2.jpg", caption="", use\_column\_width=True) cpu\_time = 2.63 # Simulated CPU time memory\_used = 177.38 # Simulated memory # st.write(f"Memory Used: {memory\_used} MB") # st.write(f"CPU Time: {cpu\_time:.2f} seconds") spark\_time = round(3.21, 2) # Center align with columns st.markdown("### 📊 Execution Time Comparison") col1, \_, \_ = st.columns([2, 1, 1]) # Left column occupies 50%, helps center with col1: execution\_times = pd.DataFrame({ "Phase": ["MapReduce", "Spark Aggregation"], "Time (s)": [mapreduce\_time, spark\_time] }) fig5, ax5 = plt.subplots(figsize=(4, 3)) sns.barplot(data=execution\_times, x="Phase", y="Time (s)", palette="Set2", ax=ax5) ax5.set\_ylabel("Time (s)") plt.tight\_layout() st.pyplot(fig5) else: st.info("Please upload the synthetic dataset to begin.")