import pandas as pd import json import os import matplotlib.pyplot as plt from pathlib import Path from datetime import datetime

# Configuration

HISTORY_FILE = "benchmark_history.json" HISTORY_DIR = Path("benchmark_results/history") CHARTS_DIR = Path("benchmark_results/history/charts")

def load_history(): """Load benchmark history from file""" # Create directories if they don't exist HISTORY_DIR.mkdir(parents=True, exist_ok=True) CHARTS_DIR.mkdir(parents=True, exist_ok=True)

```
history_path = HISTORY_DIR / HISTORY_FILE

if not history_path.exists():
    return {
        "metadata": {
            "created": datetime.now().isoformat(),
            "last_updated": datetime.now().isoformat()
        },
        "dimensions": [],
        "resonance": {},
        "one_draw": {},
        "full_processing": {}
    }

with open(history_path, 'r') as f:
    return json.load(f)
```

def find_latest_benchmark(): """Find the latest benchmark result file""" benchmark_dir = Path("benchmark_results") if not benchmark_dir.exists(): return None

```
    # Find all benchmark subdirectories
    benchmark_dirs = [d for d in benchmark_dir.iterdir() if d.is_dir() and
    d.name.startswith("benchmark_")]

    if not benchmark_dirs:
        return None

    # Sort by directory name (which includes timestamp)
    latest_dir = sorted(benchmark_dirs)[-1]

    # Find JSON files in the latest directory
    json_files = list(latest_dir.glob("*.json"))

    if not json_files:
        return None

    # Return the first JSON file (typically only one)
    return json_files[0]
```

def update_history(history, new_results): """Update benchmark history with new results""" # Update metadata history["metadata"]["last_updated"] = datetime.now().isoformat()

```
# Get device info
device = new_results["metadata"]["device"]
timestamp = datetime.strptime(
    new_results["metadata"]["timestamp"],
    "%Y-%m-%dT%H:%M:%S.%f"
).strftime("%Y-%m-%d")

# Update dimensions if needed
for dim in new_results["metadata"]["dimensions"]:
    if dim not in history["dimensions"]:
        history["dimensions"].append(dim)

# Sort dimensions
history["dimensions"] = sorted(history["dimensions"])

# Update resonance results
if "resonance" in new_results:
    if device not in history["resonance"]:
        history["resonance"][device] = {}

    for result in new_results["resonance"]:
        dim = result["dimensions"]
        if dim not in history["resonance"][device]:
            history["resonance"][device][dim] = []

        history["resonance"][device][dim].append({
            "timestamp": timestamp,
            "standard": result["results"]["standard"],
            "classical": result["results"]["classical"]
        })

# Update one_draw results
if "one_draw" in new_results:
    if device not in history["one_draw"]:
        history["one_draw"][device] = {}

    for result in new_results["one_draw"]:
        dim = result["dimensions"]
        if dim not in history["one_draw"][device]:
            history["one_draw"][device][dim] = {}

        for pattern_count, data in result["results"].items():
            if pattern_count not in history["one_draw"][device][dim]:
                history["one_draw"][device][dim][pattern_count] = []

            history["one_draw"][device][dim][pattern_count].append({
                "timestamp": timestamp,
                "one_draw": data["one_draw"],
                "classical": data["classical"],
                "binary": data["binary"],
                "grover": data["grover"]
            })

# Update full_processing results
if "full_processing" in new_results:
    if device not in history["full_processing"]:
        history["full_processing"][device] = {}

    for result in new_results["full_processing"]:
        dim = result["dimensions"]
        if dim not in history["full_processing"][device]:
            history["full_processing"][device][dim] = []

        history["full_processing"][device][dim].append({
            "timestamp": timestamp,
            "single": result["results"]["single"]
        })

return history
```

def save_history(history): """Save benchmark history to file""" history_path = HISTORY_DIR / HISTORY_FILE

```
with open(history_path, 'w') as f:
    json.dump(history, f, indent=2)
```

def create_history_charts(history): """Create charts from benchmark history""" # Create resonance history charts for device, device_data in history["resonance"].items(): device_name = device.replace(":", "_")

```
    # Create device-specific chart
    plt.figure(figsize=(12, 8))

    for dim, dim_data in device_data.items():
        # Extract timestamps and values
        timestamps = [d["timestamp"] for d in dim_data]
        standard_values = [d["standard"] for d in dim_data]

        # Plot data
        plt.plot(timestamps, standard_values, 'o-', label=f'Dim {dim}')

    plt.title(f'Resonance Performance History ({device})')
    plt.xlabel('Date')
    plt.ylabel('Time (ms)')
    plt.xticks(rotation=45)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend()
    plt.tight_layout()

    # Save chart
    chart_path = CHARTS_DIR / f"resonance_history_{device_name}.png"
    plt.savefig(chart_path)
    plt.close()

# Create one_draw history charts for selected pattern counts
pattern_counts_to_plot = ["128", "1024"]  # Select a few representative pattern counts

for device, device_data in history["one_draw"].items():
    device_name = device.replace(":", "_")

    for pattern_count in pattern_counts_to_plot:
        # Check if this pattern count exists
        has_data = False
        for dim_data in device_data.values():
            if pattern_count in dim_data:
                has_data = True
                break

        if not has_data:
            continue

        # Create pattern count specific chart
        plt.figure(figsize=(12, 8))

        for dim, dim_data in device_data.items():
            if pattern_count in dim_data:
                # Extract timestamps and values
                timestamps = [d["timestamp"] for d in dim_data[pattern_count]]
                one_draw_values = [d["one_draw"] for d in dim_data[pattern_count]]

                # Plot data
                plt.plot(timestamps, one_draw_values, 'o-', label=f'Dim {dim}')

        plt.title(f'One Draw Search Performance History (Pattern Count: {pattern_count},
{device})')
        plt.xlabel('Date')
        plt.ylabel('Time (ms)')
        plt.xticks(rotation=45)
        plt.grid(True, linestyle='--', alpha=0.7)
        plt.legend()
        plt.tight_layout()

        # Save chart
        chart_path = CHARTS_DIR / f"one_draw_history_pc{pattern_count}_{device_name}.png"
        plt.savefig(chart_path)
        plt.close()

# Create full processing history charts
for device, device_data in history["full_processing"].items():
    device_name = device.replace(":", "_")

    # Create device-specific chart
    plt.figure(figsize=(12, 8))

    for dim, dim_data in device_data.items():
        # Extract timestamps and values
        timestamps = [d["timestamp"] for d in dim_data]
        single_values = [d["single"] for d in dim_data]

        # Plot data
        plt.plot(timestamps, single_values, 'o-', label=f'Dim {dim}')

    plt.title(f'Processing Performance History ({device})')
    plt.xlabel('Date')
    plt.ylabel('Time (ms)')
    plt.xticks(rotation=45)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend()
    plt.tight_layout()

    # Save chart
    chart_path = CHARTS_DIR / f"processing_history_{device_name}.png"
```

```
        plt.savefig(chart_path)
        plt.close()
```

def main(): """Main function""" print("Updating benchmark history...")

```
    # Load existing history
    history = load_history()

    # Find latest benchmark results
    latest_benchmark = find_latest_benchmark()

    if latest_benchmark is None:
        print("No benchmark results found.")
        return

    print(f"Found latest benchmark results: {latest_benchmark}")

    # Load new results
    with open(latest_benchmark, 'r') as f:
        new_results = json.load(f)

    # Update history
    history = update_history(history, new_results)

    # Save updated history
    save_history(history)

    # Create history charts
    create_history_charts(history)

    print("Benchmark history updated successfully.")
    print(f"History saved to: {HISTORY_DIR / HISTORY_FILE}")
    print(f"Charts saved to: {CHARTS_DIR}")
```

if **name** == "**main**": main()