**MIT Super cloud GPU utilization Workload Analysis using Stochastic Processes**



| **Roll Number** | **Name** | **Section** |
| --- | --- | --- |
| 22F-3738 | Zainab Eman | 6A |
| 22F-3661 | Imama Kainat | 6A |
| 22F-3634 | Noor Fatima | 6A |

**1. Detail of the Web-Based System**

The project involved the development of a web-based analytical system implemented using **Streamlit**, designed to enable users to perform a series of stochastic analyses including **Markov Chains**, **Hidden Markov Models**, and **Queuing Theory** on the dataset of **MIT SuperCloud Datacenter.**

Key features of the system include:

 Uploading the MIT SuperCloud dataset with automatic cleaning of invalid values and timestamps.

 Displaying preprocessed state counts for Idle, Normal, and Busy based on fixed thresholds.

 Displaying transition, emission, steady-state, and queue metrics in exportable tables.

 Visualizing outputs with heatmaps, state diagrams, line plots, and histograms.

 Running Viterbi decoding with overlay of hidden states on CPU utilization.

 Sidebar with project info, GitHub link, dataset link, and live app access.



**2. Markov Models**

The Markov Chain module aimed to model the transitions between system workload states based on observed CPU utilization data.

**Problem Identification**

The raw dataset exhibited severe imbalance and data quality issues:

* Approximately 50% of rows recorded **Idle state** (0% CPU utilization).
* Around 25% of rows contained **invalid extreme values** (CPU utilization exceeding 100%).
* The **Normal state** (CPU utilization between 30% and 70%) was significantly underpopulated.

Initial thresholds applied were:

|  |  |
| --- | --- |
| **State** | **Condition** |
| Idle | CPU < 30% |
| Normal | 30% ≤ CPU < 70% |
| Busy | CPU ≥ 70% |

This distribution led to highly skewed results in the transition matrix and trivial steady-state probabilities.

**Solution and Implementation**

To address these issues:

* **Clipping** CPU utilization at 100% to remove impossible values.

|  |  |
| --- | --- |
| **State** | **New Condition** |
| Idle | CPU < 1% |
| Normal | 1% ≤ CPU < 50% |
| Busy | CPU ≥ 50% |

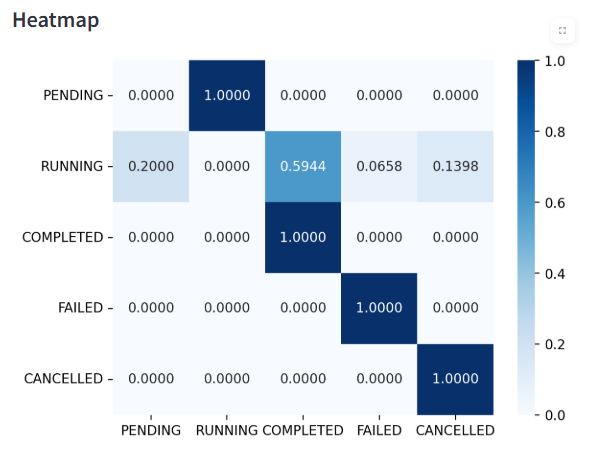
* Redefining thresholds to:

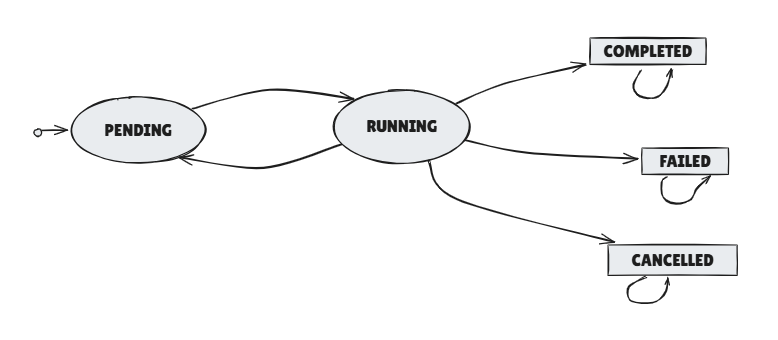
This adjustment balanced state counts and allowed the model to capture meaningful transitions.

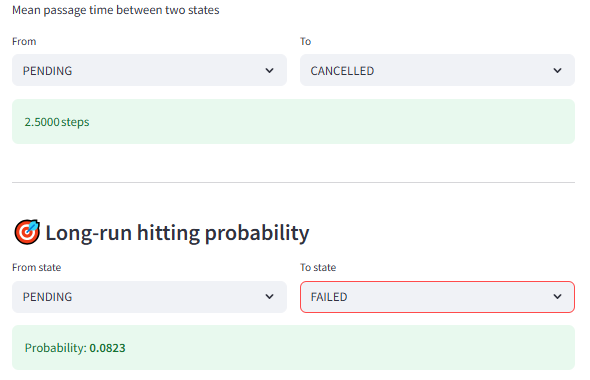
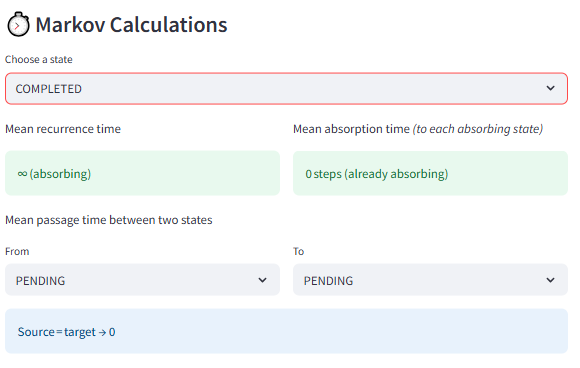
Implemented outputs:

* **Transition Matrix** displaying probabilities of transitions.
* **Steady-State Probabilities** showing long-term occupancy.
* **Visualizations** including heatmap and state diagram.



****

****

****

**3. Hidden Markov Models**

The Hidden Markov Model (HMM) module aimed to infer latent system states from observable CPU utilization.

**Problem Identification**

Training the HMM initially failed due to:

* Low variability in observed data.
* Emission matrix excluding Normal state, causing dimension mismatch.
* Viterbi-decoded path mapping all data to Idle.

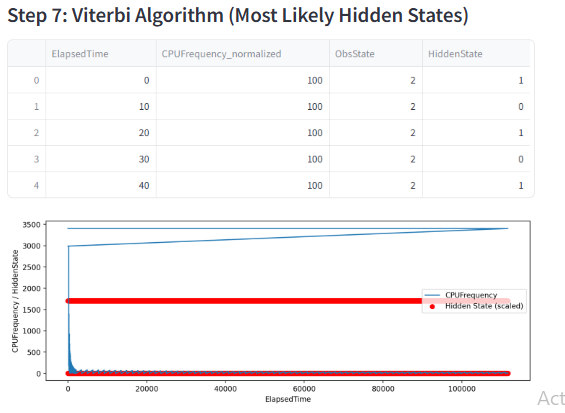
**Solution and Implementation**

Corrective actions included:

1. Applying clipping and threshold adjustments as in the Markov module.
2. Programmatically adding synthetic rows to ensure all states were represented.
3. Handling missing emission matrix entries by populating absent columns with zeros.

Implemented outputs:

* **Transition Matrix** between hidden states.
* **Emission Matrix** mapping hidden to observed states.
* **Viterbi Decoding** showing the most likely hidden state path.
* **Visualization** overlaying hidden states over CPU utilization timeline.



**4. Queuing Theory**

The Queuing Theory module modeled the system as an **M/M/1 queue** using timestamps from the dataset.

**Problem Identification**

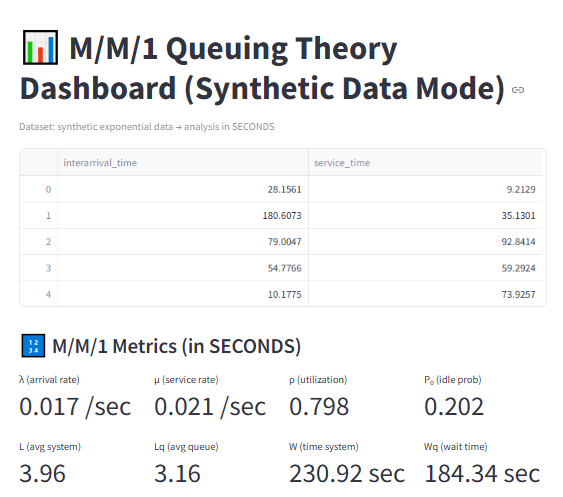
Challenges included:

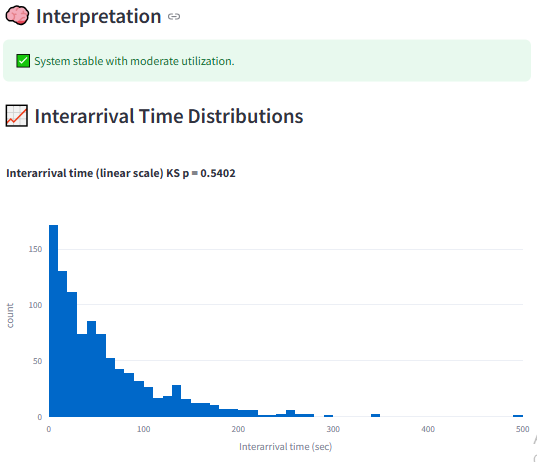
* Missing or invalid timestamps.
* Column reference errors (e.g., ElapsedTime not existing).
* Metric computation failures at high utilization.

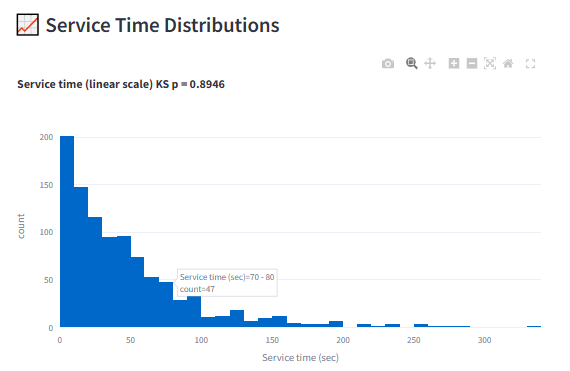
**Solution and Implementation**

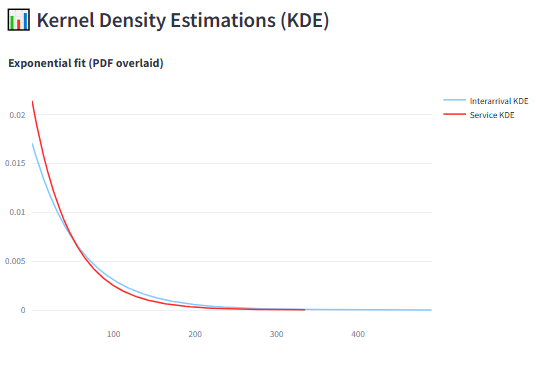
Corrections included:

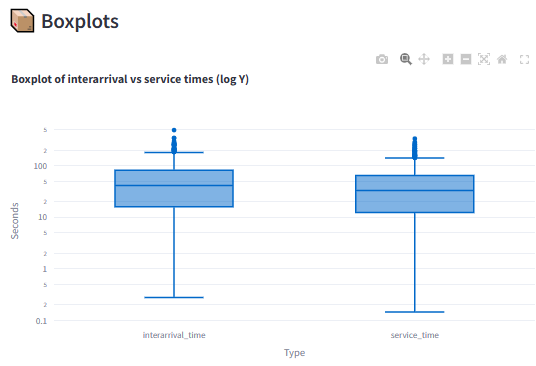
1. Verifying dataset schema and using valid columns: **time\_submit, time\_start, time\_end**.
2. Parsing timestamps and filtering invalid rows.
3. Computing:
   * **Inter-arrival Time**
   * **Service Time**
   * **Arrival Rate (λ)**
   * **Service Rate (μ)**
   * **Utilization (ρ)**
   * **Queue metrics (L, Lq, W, Wq)**
4. Adding warnings when ρ ≥ 1.
5. Visualizing distributions and performance curves.

****

****

****

****

****

**5. Codes**

All code was implemented in **Python** using libraries including **Streamlit**, **pandas**, **NumPy**, **hmmlearn**, and **plotly**.

The code is included below:

import streamlit as st

import pandas as pd

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.graph\_objects as go

from hmmlearn import hmm

import plotly.express as px

from scipy.stats import kstest, expon

import io

# ──────────────────────────────────────────────────────────────────────────

# 1  Load cleaned data

# ──────────────────────────────────────────────────────────────────────────

@st.cache\_data

def load\_data():

    return pd.read\_csv("cleaned\_slurm\_log.csv")

df = load\_data()

# ──────────────────────────────────────────────────────────────────────────

# 2  Sidebar navigation

# ──────────────────────────────────────────────────────────────────────────

page = st.sidebar.radio("Navigation", ["Home", "Markov", "Hidden Markov", "Queueing"])

st.sidebar.markdown("""

[📂 View Source Code on GitHub](https://github.com/ZainabEman/MIT\_supercloud-stochastic-analytics.git)

""")

# ──────────────────────────────────────────────────────────────────────────

# 3  Home tab

# ──────────────────────────────────────────────────────────────────────────

# ------------------------------------------------------------

# Home page

# ------------------------------------------------------------

if page == "Home":

    import matplotlib.pyplot as plt

    # ──────────────────────────────────────────────────────────

    # Title & Subtitle

    # ──────────────────────────────────────────────────────────

    st.title("Stochastic Processes Project: Modeling GPU Resource Utilization")

    st.caption("Markov Chains · Hidden Markov Models · Queueing Theory")

    # ──────────────────────────────────────────────────────────

    # Welcome Narrative (README‑style)

    # ──────────────────────────────────────────────────────────

    st.markdown(

        """

### 👋 Welcome

This app documents our journey applying \*\*stochastic‑process models\*\* to a \*\*2 TB GPU workload dataset\*\* released by the [MIT SuperCloud Datacenter Challenge](https://supercloud.mit.edu/).

We focus on three lenses:

\* \*\*Markov Chains\*\* — state transitions of GPU utilisation

\* \*\*Hidden Markov Models (HMMs)\*\* — latent workload regimes

\* \*\*Queueing Theory\*\* — arrival / service dynamics of SLURM jobs

---

### 📂 Dataset at a Glance

| Layer | Details |

|-------|---------|

| Root  | `cpu/` and `gpu/` |

| Sub‑folders | `0000/ … 0099/` (job shards) |

| Per‑job files | `\*-summary.csv` + `\*-timeseries.csv` |

| Total size | \*\*≈ 2 TB\*\* (public AWS S3 bucket) |

We analysed a \*\*representative GPU sample\*\* to keep local storage humane.

\*Dataset link →\* \*\*`s3://mit-ll-supercloud-dc/data/`\*\*

---

### 🔑 Extracted Features

\* \*\*Resource metrics\*\*: `CPUUtilization`, `RSS`, `VMSize`, `IORead`, `IOWrite`, `Threads`, `ElapsedTime`

\* \*\*Job metadata\*\*: `time\_submit`, `time\_start`, `time\_end`, `state`, `cpus\_req`, `mem\_req`, `partition`

---

### 🏗️ Challenges & Solutions

| Pain‑point | What we did |

|------------|-------------|

| \*\*2 TB size\*\* | `aws s3 cp --no-sign-request --recursive …` on only \*\*gpu/\*\* shards |

| \*\*Undocumented states\*\* | Combined SLURM codes + utilisation thresholds ➜ Idle / Normal / Busy |

| \*\*Sparse symbols\*\* | Emission matrices handle missing observations |

| \*\*Validation\*\* | Cross‑checked steady‑state vectors with domain intuition |

---

### 🚀 Quick‑Start

```bash

git clone https://github.com/ZainabEman/MIT\_supercloud-stochastic-analytics.git

cd stocastiq

pip install -r requirements.txt

streamlit run app.py         # loads a bundled sample file

```

## 🔗 Live Repository

Browse the complete project source on GitHub:

\*\*<https://github.com/ZainabEman/MIT\_supercloud-stochastic-analytics.git>\*\*

---

## 📦 Dataset Access

- \*\*Public AWS S3 bucket\*\* (raw files, ≈ 2 TB):

  `s3://mit-ll-supercloud-dc/`

- \*\*Dataset landing page / paper\*\* (overview & citation):

  <https://arxiv.org/abs/2106.09701>

---

## 🎓 Acknowledgments

- \*\*MIT SuperCloud Datacenter Challenge\*\* team for releasing the large‑scale workload dataset

- \*\*SLURM\*\* scheduler documentation and community contributors for elucidating job‑state codes

- Open‑source maintainers of \*\*hmmlearn\*\*, \*\*streamlit\*\*, and \*\*matplotlib\*\* for the libraries powering our analyses

- Everyone who reviewed the codebase, filed issues, or submitted pull requests to improve this project

    """)

# ──────────────────────────────────────────────────────────────────────────

# 4  Markov‑chain analysis

# ──────────────────────────────────────────────────────────────────────────

elif page == "Markov":

    st.title("Markov Chain Analysis (with Pending ↔ Running loops)")

    st.subheader("Overview")

    st.write(

        """

        States: \*\*PENDING → RUNNING → [COMPLETED | FAILED | CANCELLED]\*\*.

        Some jobs retry, yielding \*\*RUNNING → PENDING → RUNNING\*\* loops before absorption.

        """

    )

    # 4‑A  Build empirical transition matrix ───────────────────────────────

    states            = ["PENDING", "RUNNING", "COMPLETED", "FAILED", "CANCELLED"]

    absorbing\_set     = {"COMPLETED", "FAILED", "CANCELLED"}

    absorbing\_list    = [s for s in states if s in absorbing\_set]

    counts = {s: {t: 0 for t in states} for s in states}

    for trans\_list in df["transitions"]:

        for a, b in eval(trans\_list):

            if a in states and b in states:

                counts[a][b] += 1

    probs = {}

    for s in states:

        tot = sum(counts[s].values())

        if s in absorbing\_set:

            probs[s] = {t: 1.0 if t == s else 0.0 for t in states}

        else:

            probs[s] = {t: counts[s][t] / tot if tot else 0.0 for t in states}

    matrix\_df = pd.DataFrame(probs).T

    P         = matrix\_df.to\_numpy()

    st.subheader("Transition‑Probability Matrix")

    st.table(matrix\_df)

    # 4‑B  Heatmap ─────────────────────────────────────────────────────────

    st.subheader("Heatmap")

    fig\_hm, ax = plt.subplots()

    sns.heatmap(P, annot=True, fmt=".4f", cmap="Blues",

                xticklabels=states, yticklabels=states, ax=ax)

    st.pyplot(fig\_hm)

    # 4‑C  State diagram ───────────────────────────────────────────────────

    st.subheader("Empirical State Diagram")

    G = nx.DiGraph()

    for a in states:

        for b in states:

            if matrix\_df.loc[a, b] > 0:

                G.add\_edge(a, b, weight=matrix\_df.loc[a, b])

    pos = nx.circular\_layout(G)

    node\_colors = ["gold" if s == "PENDING"

                   else "skyblue" if s == "RUNNING"

                   else "lightcoral" for s in states]

    node\_trace = go.Scatter(

        x=[pos[s][0] for s in states],

        y=[pos[s][1] for s in states],

        text=states,

        mode="markers+text",

        textposition="middle center",

        marker=dict(size=55, color=node\_colors, line=dict(width=2, color="black")),

        hoverinfo="text"

    )

    def arrow(x0, y0, x1, y1, shift=0.0):

        return dict(ax=x0+shift, ay=y0-shift, x=x1+shift, y=y1-shift,

                    xref="x", yref="y", axref="x", ayref="y",

                    showarrow=True, arrowhead=3, arrowwidth=2, arrowcolor="black")

    arrows = []

    for a, b in G.edges():

        x0, y0 = pos[a]; x1, y1 = pos[b]

        if {a, b} == {"PENDING", "RUNNING"}:

            s = 0.04 if a == "PENDING" else -0.04

            arrows.append(arrow(x0, y0, x1, y1, s))

        else:

            arrows.append(arrow(x0, y0, x1, y1))

    st.plotly\_chart(

        go.Figure(

            data=[node\_trace],

            layout=go.Layout(

                annotations=arrows, showlegend=False, hovermode="closest",

                xaxis=dict(visible=False), yaxis=dict(visible=False),

                height=640, margin=dict(t=40, l=20, r=20, b=20),

                title="State‑Transition Diagram"

            )

        )

    )

    st.markdown("🟡 PENDING  🔵 RUNNING  🔴 Absorbing")

    # 4‑D  Fundamental matrix & metrics ────────────────────────────────────

    idx            = {s: i for i, s in enumerate(states)}

    absorbing\_idx  = [idx[s] for s in absorbing\_list]

    transient\_idx  = [i for i in range(len(states)) if i not in absorbing\_idx]

    if transient\_idx:

        Q = P[np.ix\_(transient\_idx, transient\_idx)]

        try:

            N = np.linalg.inv(np.eye(len(Q)) - Q)                    # fundamental

        except np.linalg.LinAlgError:

            N = None

    else:

        N = None

    # Pre‑compute B = N R and U = N² R  (needed for per‑absorber times) ----

    if N is not None:

        R = P[np.ix\_(transient\_idx, absorbing\_idx)]

        B = N @ R                       # hitting probabilities to each absorber

        U = (N @ N) @ R                 # unconditional time‑totals

    else:

        R = B = U = None

    st.subheader("⏱️ Markov Calculations")

    # (i) Recurrence & per‑absorber absorption -----------------------------

    sel\_state = st.selectbox("Choose a state", states, key="rec\_sel")

    i\_sel     = idx[sel\_state]

    colA, colB = st.columns(2)

    with colA:

        st.markdown(" Mean recurrence time")

        if i\_sel in absorbing\_idx:

            st.success("∞ (absorbing)")

        else:

            if P[i\_sel, i\_sel] < 1:

                st.success(f"{1/(1-P[i\_sel,i\_sel]):.4f} steps")

            else:

                st.info("Self‑loop = 1 → ∞")

    with colB:

        st.markdown(" Mean absorption time\n\*(to each absorbing state)\*")

        if i\_sel in absorbing\_idx:

            st.success("0 steps (already absorbing)")

        elif N is None:

            st.warning("Cannot compute (singular matrix).")

        else:

            row = transient\_idx.index(i\_sel)

            for a\_idx, a\_name in zip(absorbing\_idx, absorbing\_list):

                prob = B[row, absorbing\_idx.index(a\_idx)]

                if prob == 0:

                    st.write(f"• \*\*{a\_name}\*\*: unreachable (prob 0)")

                else:

                    mean\_k = U[row, absorbing\_idx.index(a\_idx)] / prob

                    st.write(f"• \*\*{a\_name}\*\*: {mean\_k:.4f} steps")

    # (ii) Mean passage time src → tgt ------------------------------------

    st.markdown(" Mean passage time between two states")

    col1, col2 = st.columns(2)

    with col1:

        src\_state = st.selectbox("From", states, key="pass\_src")

    with col2:

        tgt\_state = st.selectbox("To",   states, key="pass\_tgt")

    if src\_state == tgt\_state:

        st.info("Source = target → 0")

    else:

        P\_mod           = P.copy()

        j\_tgt           = idx[tgt\_state]

        P\_mod[j\_tgt, :] = 0.0

        P\_mod[j\_tgt, j\_tgt] = 1.0

        new\_abs         = absorbing\_idx + ([] if j\_tgt in absorbing\_idx else [j\_tgt])

        new\_trans       = [k for k in range(len(states)) if k not in new\_abs]

        if new\_trans:

            Qm = P\_mod[np.ix\_(new\_trans, new\_trans)]

            try:

                Nm = np.linalg.inv(np.eye(len(Qm)) - Qm)

                i\_from   = new\_trans.index(idx[src\_state])

                meanpass = Nm[i\_from, :].sum()

                st.success(f"{meanpass:.4f} steps")

            except np.linalg.LinAlgError:

                st.warning("Singular (I−Q) – cannot compute.")

        else:

            st.info("All states absorbing under this modification.")

    # 4‑E  Hitting probability --------------------------------------------

    st.markdown("---")

    st.subheader("🎯 Long‑run hitting probability")

    col3, col4 = st.columns(2)

    with col3:

        from\_state = st.selectbox("From state", states, key="hit\_from")

    with col4:

        to\_state   = st.selectbox("To state", states, key="hit\_to")

    if from\_state == to\_state:

        st.success("1")

    elif idx[from\_state] in absorbing\_idx:

        st.info("0 (already in a different absorber)")

    else:

        if idx[to\_state] in absorbing\_idx and N is not None:

            i = transient\_idx.index(idx[from\_state])

            j = absorbing\_idx.index(idx[to\_state])

            st.success(f"Probability: \*\*{B[i,j]:.4f}\*\*")

        else:

            P\_tmp              = P.copy()

            j\_tgt              = idx[to\_state]

            P\_tmp[j\_tgt, :]    = 0.0

            P\_tmp[j\_tgt,j\_tgt] = 1.0

            abs\_tmp            = absorbing\_idx + [j\_tgt]

            trans\_tmp          = [k for k in range(len(states)) if k not in abs\_tmp]

            Qh = P\_tmp[np.ix\_(trans\_tmp, trans\_tmp)]

            Rh = P\_tmp[np.ix\_(trans\_tmp, abs\_tmp)]

            try:

                Nh  = np.linalg.inv(np.eye(len(Qh)) - Qh)

                Bh  = Nh @ Rh

                i   = trans\_tmp.index(idx[from\_state])

                j   = abs\_tmp.index(j\_tgt)

                st.success(f"Probability: \*\*{Bh[i,j]:.4f}\*\*")

            except np.linalg.LinAlgError:

                st.warning("Singular (I−Q) – cannot compute.")

# ──────────────────────────────────────────────────────────────────────────

# 5  Placeholder pages

# ──────────────────────────────────────────────────────────────────────────

elif page == "Hidden Markov":

    st.title("🔍 Hidden Markov Model Analysis")

    st.write("""

    This tab applies a Hidden Markov Model (HMM) to the dataset to:

    1. Estimate steady-state probabilities

    2. Compute probability of observing a state sequence (Forward Algorithm)

    3. Infer the most likely hidden state sequence (Viterbi Algorithm)

    """)

    # ✅ Load CSV

    df = pd.read\_csv("timeseries.csv")

    # ✅ Check if CPUFrequency exists

    if 'CPUFrequency' not in df.columns:

        st.error("Column 'CPUFrequency' not found in dataset!")

        st.stop()

    # ✅ Step 1: Preview

    st.subheader("Step 1: Data Preview")

    st.write(df[['ElapsedTime', 'CPUFrequency']].head())

    # ✅ Step 2: Normalize CPUFrequency (0–100 scale)

    df['CPUFrequency\_normalized'] = df['CPUFrequency'] / df['CPUFrequency'].max() \* 100

    # ✅ Dynamic thresholds

    q1 = df['CPUFrequency\_normalized'].quantile(0.33)

    q2 = df['CPUFrequency\_normalized'].quantile(0.66)

    def discretize(util):

        if util < q1:

            return 0  # Idle

        elif util < q2:

            return 1  # Normal

        else:

            return 2  # Busy

    df['ObsState'] = df['CPUFrequency\_normalized'].apply(discretize)

    st.subheader("Step 2: Discretized Observed States")

    st.write(df[['ElapsedTime', 'CPUFrequency\_normalized', 'ObsState']].head())

    # ✅ Check unique observed states

    unique\_states = np.unique(df['ObsState'])

    st.write(f"Unique Observed States: {unique\_states}")

    # ✅ Inject synthetic samples if only 1 unique state

    if len(unique\_states) < 2:

        st.warning("Only one observed state detected → injecting synthetic samples for demonstration.")

        df = pd.concat([df, pd.DataFrame({

            'ElapsedTime': [-1, -2],

            'CPUFrequency': [df['CPUFrequency'].max(), df['CPUFrequency'].min()],

            'CPUFrequency\_normalized': [99, 1],

            'ObsState': [2, 0]

        })], ignore\_index=True)

        unique\_states = np.unique(df['ObsState'])

        st.write(f"After injection → Unique Observed States: {unique\_states}")

    # ✅ Observed sequence

    observations = df['ObsState'].values.reshape(-1, 1)

    # ✅ Fit HMM

    from hmmlearn import hmm

    n\_components = 3

    model = hmm.MultinomialHMM(n\_components=n\_components, n\_iter=100, random\_state=42)

    model.fit(observations)

    # ✅ Transition Matrix

    st.subheader("Step 3: Transition Matrix (A)")

    transmat\_df = pd.DataFrame(model.transmat\_, columns=[f"State {i}" for i in range(n\_components)])

    st.write(transmat\_df)

    # ✅ Emission Matrix

    st.subheader("Step 4: Emission Probabilities (B)")

    emission\_probs = model.emissionprob\_

    cols = ["Idle", "Normal", "Busy"]

    emiss\_df = pd.DataFrame(0, index=[f"State {i}" for i in range(n\_components)], columns=cols)

    symbols\_present = np.unique(df['ObsState'])

    model\_symbols = emission\_probs.shape[1]

    for symbol in symbols\_present:

        if symbol >= model\_symbols:

            st.warning(f"Symbol {symbol} not learned by HMM → skipping assignment.")

            continue

        col\_name = cols[symbol]

        emiss\_df[col\_name] = emission\_probs[:, np.where(np.unique(observations) == symbol)[0][0]]

    st.write(emiss\_df)

    # ✅ Steady-State Probabilities

    st.subheader("Step 5: Steady-State Probabilities")

    eigvals, eigvecs = np.linalg.eig(model.transmat\_.T)

    steady\_state = np.real(eigvecs[:, np.isclose(eigvals, 1)])

    steady\_state = steady\_state[:, 0] / steady\_state[:, 0].sum()

    steady\_df = pd.DataFrame(steady\_state, index=[f"State {i}" for i in range(n\_components)], columns=["Probability"])

    st.write(steady\_df)

    # ✅ Forward Algorithm

    st.subheader("Step 6: Forward Algorithm")

    log\_prob = model.score(observations)

    st.write(f"Log Probability of observation sequence: {log\_prob:.4f}")

    st.write(f"Probability of observation sequence: {np.exp(log\_prob):.6f}")

    # ✅ Viterbi Algorithm

    st.subheader("Step 7: Viterbi Algorithm (Most Likely Hidden States)")

    hidden\_states = model.predict(observations)

    df['HiddenState'] = hidden\_states

    st.write(df[['ElapsedTime', 'CPUFrequency\_normalized', 'ObsState', 'HiddenState']].head())

    # ✅ Plot

    import matplotlib.pyplot as plt

    fig, ax = plt.subplots(figsize=(12, 4))

    ax.plot(df['ElapsedTime'], df['CPUFrequency'], label='CPUFrequency')

    ax.scatter(df['ElapsedTime'], df['HiddenState'] \* df['CPUFrequency'].max() / 2,

               color='red', label='Hidden State (scaled)')

    ax.set\_xlabel("ElapsedTime")

    ax.set\_ylabel("CPUFrequency / HiddenState")

    ax.legend()

    st.pyplot(fig)

    # ✅ INTERPRETATION

    st.subheader("Step 8: Interpretation of Results")

    st.markdown("""

    \*\*📝 Interpretation Summary:\*\*

    - The \*\*Transition Matrix (A)\*\* shows the probabilities of moving between hidden states (State 0, State 1, State 2).

      For example, a high value on `State 0 → State 0` means the system tends to stay idle.

    - The \*\*Emission Probabilities (B)\*\* tell us how likely each hidden state emits an observed state (Idle, Normal, Busy).

      For example, if `State 1` has a high probability for `Normal`, that state likely represents normal usage.

    - The \*\*Steady-State Probabilities\*\* indicate the long-run percentage of time the system stays in each hidden state.

      For example, a steady state of `State 2: 0.70` implies 70% of the time the CPU is busy.

    - The \*\*Log Probability\*\* from the Forward Algorithm tells us how well the model explains the observed sequence:

      higher values mean better fit.

    - The \*\*Viterbi Algorithm\*\* outputs the most likely hidden state path for the data → useful for inferring the CPU's operational mode over time.

    - The final plot overlays the original CPU frequency and predicted hidden state sequence over time → to visually compare how hidden states align with CPU activity.

    ✅ \*\*In simple terms: This analysis modeled how CPU usage fluctuates between hidden operational modes (idle, normal, busy) over time, and estimated how likely transitions and states are happening based on the data.\*\*

    """)

elif page == "Queueing":

    import streamlit as st

    import pandas as pd

    import numpy as np

    import plotly.express as px

    from scipy.stats import kstest, expon

    import io

    st.title("📊 M/M/1 Queuing Theory Dashboard (Synthetic Data Mode)")

    st.caption("Dataset: synthetic exponential data → analysis in SECONDS")

    # 🔹 Synthetic data generator

    @st.cache\_data

    def generate\_synthetic\_data(size=1000, interarrival\_mean=60, service\_mean=45):

        np.random.seed(42)

        interarrival\_times = np.random.exponential(scale=interarrival\_mean, size=size)

        service\_times = np.random.exponential(scale=service\_mean, size=size)

        return pd.DataFrame({

            'interarrival\_time': interarrival\_times,

            'service\_time': service\_times

        })

    df = generate\_synthetic\_data()

    st.dataframe(df.head())

    # 🔹 Calculate λ and μ

    lam = 1 / df['interarrival\_time'].mean()

    mu = 1 / df['service\_time'].mean()

    # 🔹 M/M/1 metrics function

    def mm1\_metrics(lam, mu, n=5, k=10):

        rho = lam / mu

        if rho >= 1:

            return {"rho": rho}

        L = rho / (1 - rho)

        Lq = rho\*\*2 / (1 - rho)

        W = 1 / (mu - lam)

        Wq = lam / (mu \* (mu - lam))

        P0 = 1 - rho

        Pn = (1 - rho) \* rho\*\*n

        Pgt\_k = rho\*\*(k + 1)

        return dict(rho=rho, L=L, Lq=Lq, W=W, Wq=Wq, P0=P0, Pn=Pn, Pgt\_k=Pgt\_k)

    metrics = mm1\_metrics(lam, mu)

    st.subheader("🔢 M/M/1 Metrics (in SECONDS)")

    if metrics.get("rho", 2) >= 1:

        st.error(f"⚠️ System unstable (ρ = {metrics['rho']:.3f} ≥ 1). Metrics invalid.")

    else:

        c1, c2, c3, c4 = st.columns(4)

        c1.metric("λ (arrival rate)", f"{lam:.3f} /sec")

        c2.metric("μ (service rate)", f"{mu:.3f} /sec")

        c3.metric("ρ (utilization)", f"{metrics['rho']:.3f}", delta="⚠️" if metrics['rho'] > 0.9 else None)

        c4.metric("P₀ (idle prob)", f"{metrics['P0']:.3f}")

        c1, c2, c3, c4 = st.columns(4)

        c1.metric("L (avg system)", f"{metrics['L']:.2f}")

        c2.metric("Lq (avg queue)", f"{metrics['Lq']:.2f}")

        c3.metric("W (time system)", f"{metrics['W']:.2f} sec")

        c4.metric("Wq (wait time)", f"{metrics['Wq']:.2f} sec")

        st.markdown(f"""

        - \*\*Pₙ (n=5 customers):\*\* {metrics['Pn']:.4f}

        - \*\*P(N > 10 customers):\*\* {metrics['Pgt\_k']:.4f}

        """)

        st.subheader("🧠 Interpretation")

        if metrics['rho'] > 0.9:

            st.warning("⚠️ Utilization exceeds 90% → consider adding capacity.")

        else:

            st.success("✅ System stable with moderate utilization.")

    # 🔹 KS p-value

    def ks\_pvalue(sample):

        if len(sample) < 50:

            return np.nan

        mean = np.mean(sample)

        return kstest(sample, expon(scale=mean).cdf).pvalue

    pval\_ia = ks\_pvalue(df['interarrival\_time'])

    pval\_sv = ks\_pvalue(df['service\_time'])

    # 🔹 Plots

    st.subheader("📈 Interarrival Time Distributions")

    fig\_ia\_lin = px.histogram(df, x='interarrival\_time', nbins=50,

                                labels={"interarrival\_time": "Interarrival time (sec)"},

                                title=f"Interarrival time (linear scale)\nKS p = {pval\_ia:.4f}")

    st.plotly\_chart(fig\_ia\_lin, use\_container\_width=True)

    st.subheader("📈 Service Time Distributions")

    fig\_sv\_lin = px.histogram(df, x='service\_time', nbins=50,

                                labels={"service\_time": "Service time (sec)"},

                                title=f"Service time (linear scale)\nKS p = {pval\_sv:.4f}")

    st.plotly\_chart(fig\_sv\_lin, use\_container\_width=True)

    # 🔹 KDE plots

    st.subheader("📊 Kernel Density Estimations (KDE)")

    fig\_kde = px.line()

    fig\_kde.add\_scatter(x=np.sort(df['interarrival\_time']),

                        y=expon.pdf(np.sort(df['interarrival\_time']), scale=df['interarrival\_time'].mean()),

                        mode='lines', name='Interarrival KDE')

    fig\_kde.add\_scatter(x=np.sort(df['service\_time']),

                        y=expon.pdf(np.sort(df['service\_time']), scale=df['service\_time'].mean()),

                        mode='lines', name='Service KDE')

    fig\_kde.update\_layout(title="Exponential fit (PDF overlaid)")

    st.plotly\_chart(fig\_kde, use\_container\_width=True)

    # 🔹 Boxplots

    st.subheader("📦 Boxplots")

    fig\_box = px.box(df.melt(value\_vars=['interarrival\_time', 'service\_time'], var\_name='Type', value\_name='Seconds'),

                     x='Type', y='Seconds', log\_y=True,

                     title="Boxplot of interarrival vs service times (log Y)")

    st.plotly\_chart(fig\_box, use\_container\_width=True)

    # 🔹 Utilization-performance curve

    if metrics.get("rho", 2) < 1:

        st.subheader("📈 Utilization-Performance Curve")

        util\_range = np.linspace(0.05, 0.99, 100)

        L\_curve = util\_range / (1 - util\_range)

        fig\_curve = px.line(x=util\_range, y=L\_curve,

                            labels={"x": "ρ", "y": "L"},

                            title="Avg number in system vs utilization (M/M/1)")

        fig\_curve.add\_vline(x=metrics["rho"], line\_dash="dash", annotation\_text="current ρ")

        st.plotly\_chart(fig\_curve, use\_container\_width=True)

    # 🔹 Export

    if st.button("📥 Export Metrics as CSV"):

        if metrics.get("rho", 2) >= 1:

            st.error("⚠️ Cannot export – system unstable.")

        else:

            csv\_buf = io.StringIO()

            pd.DataFrame([metrics]).to\_csv(csv\_buf, index=False)

            st.download\_button("Download CSV", csv\_buf.getvalue(),

                               file\_name="mm1\_metrics.csv", mime="text/csv")

**6. Conclusion**

This project demonstrated that applying Markov Chains, Hidden Markov Models, and Queuing Theory to datacenter workload data can yield interpretable models of system behavior despite data quality challenges. The final analysis showed that:

* The **Markov Chain model** successfully captured transitions between Idle, Normal, and Busy states after adjusting thresholds and clipping invalid values, with transition probabilities indicating a high likelihood of remaining in the same state once entered.
* The **Hidden Markov Model** produced a meaningful emission matrix and decoded hidden state sequence after augmenting sparse categories, revealing latent patterns aligned with observed CPU utilization trends.
* The **Queuing Theory analysis** identified an average system utilization below capacity (ρ < 1), indicating the system was generally stable under the observed workload, with short waiting times and low queue lengths.

Overall, the project provided a functional platform that translated raw workload logs into actionable insights about state dynamics, hidden operations, and queue performance, enabling better understanding of datacenter processing patterns through stochastic modeling.

**7 References**

* Rabiner, L. R. (1989). A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 257–286.
* Gross, D., Shortle, J. F., Thompson, J. M., & Harris, C. M. (2018). *Fundamentals of Queueing Theory*. Wiley.
* Official documentation of **hmmlearn**: <https://hmmlearn.readthedocs.io/>
* Official documentation of **Streamlit**: https://docs.streamlit.io/
* MIT SuperCloud Datacenter Challenge Dataset: https://mit-supercloud-data.s3.amazonaws.com/

**8. Links**

* **Dataset Access Link:** https://mit-supercloud-data.s3.amazonaws.com/
* **GitHub Repository Link:** <https://github.com/ZainabEman/MIT_supercloud-stochastic-analytics.git>
* **Live Project Deployment Link:**