CascadEye: A Machine Learning System for Cascading Failure Risk Analysis in Power Grids

Course: CSC 792 – Machine Learning Systems Design
Instructor: Chulwoo Pack
Team: Syed Abdullah Al Nahid, Mohammad Johurul Islam, Yingman Bian
Date: December 02, 2025

GitHub repository: *[Please Click]*

1. Problem Definition

Modern power grids experience thousands of small transmission line outages every year. Most of these events do not grow, but a small fraction can propagate across several steps and become large cascading failures. These rare but high-impact events are difficult to study because utility outage records are noisy, inconsistent, and spread across long time periods. As a result, engineers often lack simple tools to inspect patterns inside these cascades or understand how failures spread from one line to another.

Several research papers model cascading failures using statistical estimation, interaction matrices, or stochastic simulations. Some use Expectation–Maximization (EM) to infer how one line may trigger the next. Others explore machine learning models such as sequence networks or graph methods. However, most of these methods remain research-only. They require custom code, heavy dependencies, and complex workflows that are difficult for practitioners to adopt. Utilities typically rely on spreadsheets or manual inspection, which limits large-scale analysis.

We aim to close this gap by building CascadEye, a lightweight desktop application that helps users explore cascading failure data, estimate interaction patterns, and visualize system-level risk. Our system loads raw outage records, rebuilds cascade sequences, and constructs a learned interaction matrix using either a simple frequency baseline or a pretrained LSTM-Attention model. It then performs fast Monte-Carlo simulations to test how well the model reproduces real cascade sizes.

Our goal is to make advanced analysis accessible. CascadEye provides a clean interface, a structured data pipeline, and automatic evaluation methods (CCDF plots, KS distance, and directional tests). This design allows engineers and researchers to explore cascading behavior without writing code, while still grounding the analysis in well-established modeling approaches.

2. System Design

CascadEye is designed as a compact but complete ML system that takes raw outage records and turns them into interpretable insights about cascading failures. The system follows a clear pipeline: data ingestion → sequence construction → model inference → simulation → validation → monitoring. Figure 1 illustrates this flow and shows how the components interact.

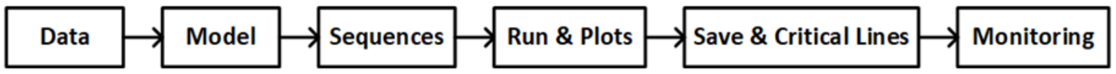


Figure 1. CascadEye workflow: Data → Model → Sequences → Run & Plots → Save & Critical Lines → Monitoring.

2.1 User Interface and Workflow Structure

The application is implemented as a Tkinter desktop tool. We chose Tkinter because it runs fully offline, requires no web server, has zero deployment overhead, and works on standard laptops without GPU support. The interface is organized into six tabs that mirror the ML pipeline and guide the user step-by-step.

- **Data** loads the original outage dataset, displays structured summaries, and provides a *Dataset Insights* view with cascade counts, max generations, and unique line IDs.
- **Model** loads a pretrained LSTM + Multi-Head Attention model and reveals its architecture, optimizer, activation functions, training KPIs, and SHA-1 fingerprint.
- **Sequences** converts outage events into padded binary tensors used for model prediction.
- **Run & Plots** performs Monte-Carlo simulations, builds the interaction matrix, and visualizes outputs such as PDF comparisons, CCDF overlays, and the interaction graph.
- **Save & Critical Lines** displays Top-k high-influence lines and their strongest downstream targets.
- **Monitoring** shows KS drift, directional-test trends, and recent evaluation logs using embedded Matplotlib figures.

The result is a simple, predictable workflow where each tab exposes only the tools needed for that stage.

2.2 Data Pipeline and Preprocessing

The system begins with a BPA-style cascade dataset, where each row represents one line outage within a generation. When a file is loaded, CascadEye checks for required fields, computes summary statistics, and shows a preview of the first rows. A **SHA-1 hash** of the dataset is stored to tie every run to the exact data version. Users may also generate a synthetic feeder dataset, which follows the same structure but is created from topological rules that mimic line adjacency.

2.3 Modeling and Interaction Matrix Estimation

CascadEye uses a pretrained **LSTM with two attention heads** to model propagation patterns. This model is well-suited for cascading failures because the process is sequential and long-range dependencies matter. For each generation, the model outputs a failure-probability vector. These predictions are aggregated into an **interaction matrix** (B), where (B_{i,j}) represents how strongly the failure of line (i) influences line (j) at the next step. All model metadata—hyperparameters, optimizer, training loss, validation accuracy, and file hash—is recorded for reproducibility.

2.4 Simulation and Validation Engine

To test whether the learned (B) reflects real behavior, CascadEye runs thousands of Monte-Carlo simulations. Simulated cascades grow generation by generation until no new failures occur or a generation limit is reached. The tool then compares simulated and real cascade sizes using **log-log PDFs, CCDF overlays, the Kolmogorov–Smirnov (KS) statistic, and a directional expectation test** that checks whether larger cascades correspond to higher predicted risk.

2.5 Logging, Critical Lines, and Monitoring

All operations—data load, model selection, simulation runs, KS statistics, and directional-test metrics—are written to an `events.jsonl` log. Each run also produces a folder containing the B-matrix, Top-k critical lines, validation metrics, and fingerprints. The Monitoring tab reads these logs to show recent KS drift, directional-test trends, and a table of past evaluations. This lightweight setup provides transparency and reproducibility without requiring any cloud infrastructure.

3. Machine Learning Component

CascadEye uses a pretrained LSTM + Multi-Head Attention model to learn how failures propagate across generations. Instead of forecasting real-time outages, the model focuses on extracting an interpretable interaction matrix B, where each entry captures how strongly one line influences another during a cascade. This section summarizes the training data, the model architecture, and the iterative steps that led to the final design.

3.1 Training Data and Sequence Construction

The model is trained on historical cascading-outage data from the Bonneville Power Administration (BPA). Each row represents a line failure within a propagation step. For sequence modeling, each cascade is converted into a series of binary vectors, where each vector has one entry per transmission line, with 1 indicating failure in that generation.

Cascades vary in length, so we pad all sequences to a fixed time horizon. From this padded structure, we construct supervised learning pairs:

- $\mathbf{X}_g$: failure pattern at generation g
- $\mathbf{X}_{g+1}$: failure pattern at generation $g + 1$

This frame-by-frame structure teaches the model how outages evolve, and it allows the LSTM to learn temporal dependencies that simple frequency statistics cannot capture.

3.2 LSTM + Multi-Head Attention Architecture

The model uses a 64-unit LSTM to process the sequence of failure vectors, capturing how outages evolve over time. A two-head attention module highlights the most influential patterns, enhancing interpretability. A final fully connected layer with a sigmoid activation produces a probability vector for the likelihood of each line failing next.

To construct the interaction matrix **B**, we aggregate these predicted probabilities across all cascades. Entry **B(i, j)** reflects how strongly the failure of line *i* contributes to the failure of line *j* in the next generation.

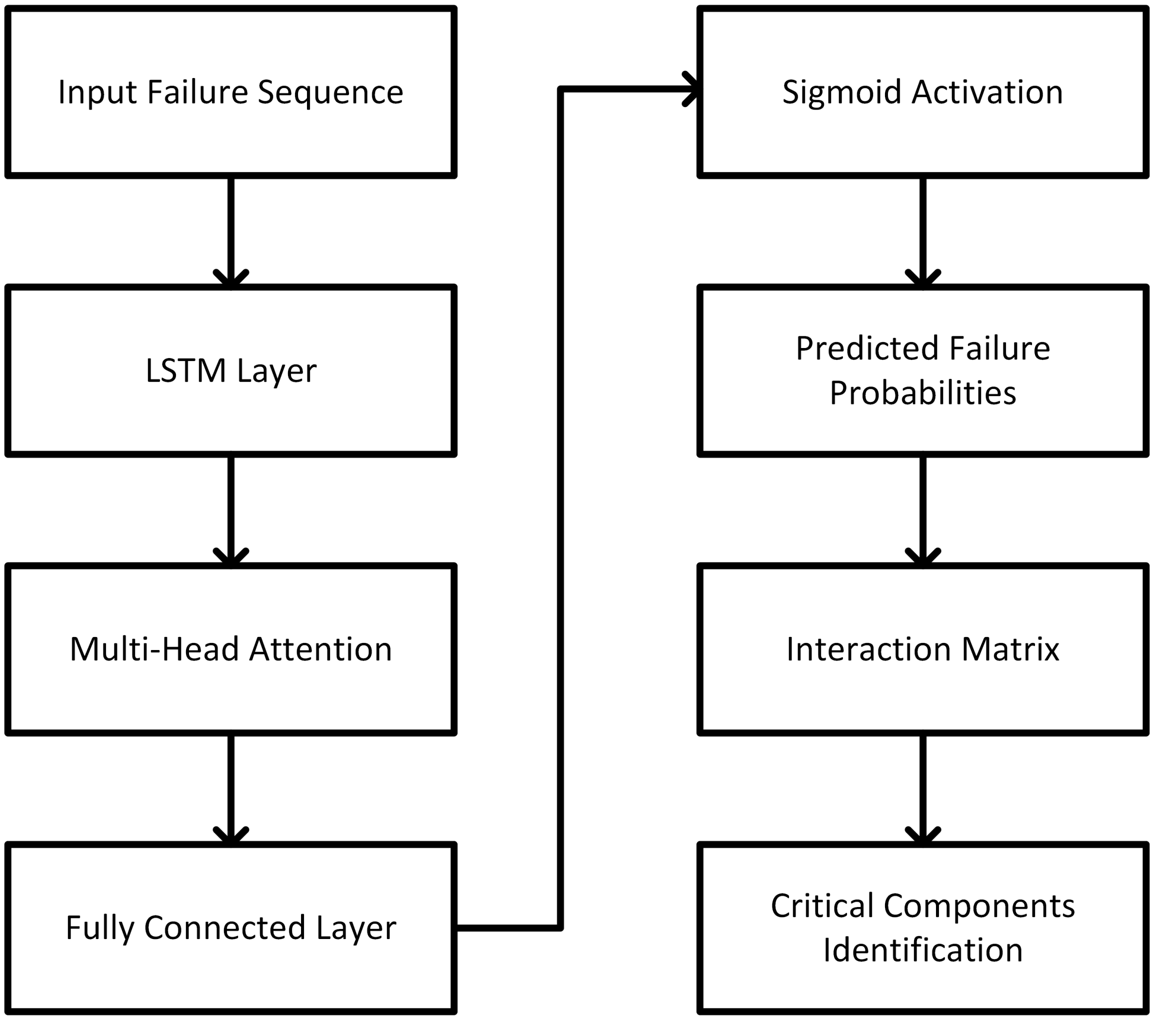


Figure 2. Model architecture used in CascadEye (LSTM + Multi-Head Attention).

3.3 Iterative Development and Refinement

We refined the model through several iterations. Our goal was to find an architecture that captures nonlinear propagation patterns while staying stable during training. We compared single-layer LSTM, stacked LSTM, GRU, and attention-based models. All were trained and tested on the same padded BPA sequences to ensure a fair comparison.

The LSTM with multi-head attention gave the best results. It reached 80% recall when detecting ground-truth critical lines. Models without attention, or with only one head, had much lower recall. Simple LSTM and GRU models stayed around 5% recall, which is close to random. These results show that multi-head attention is necessary to capture the diverse and nonlinear dependencies in real cascades.

After selecting the best architecture, we tuned hidden size, attention heads, and learning rate. The final configuration used 64 hidden units, two attention heads, Adam optimizer, and BCE loss. Every checkpoint was fingerprinted with SHA-1, and key performance indicators were logged for full reproducibility inside CascadEye.

4. System Evaluation

We evaluated CascadEye at three levels:

1. The quality of the learned interaction matrix from the LSTM–Attention model.
2. How well simulated cascades match empirical cascade sizes.
3. How stable and interpretable the critical-line rankings are across runs.

We also use the logs to understand limitations and typical failure modes.

4.1 Model-level Performance

During training, the LSTM + 2-head Attention model reached a train loss of about **0.033** and a validation loss of about **0.031**. It also achieved **0.998** validation accuracy and a **precision@15 of 0.8**, meaning that about 80% of the top-15 predicted lines fail in the next generation of the test data. These results show that the model learns the next-step prediction task well. However, we use these scores only as a screening step. Models with low loss and reasonable precision are then used to generate the interaction matrices for our simulations, rather than assuming that the learned B-matrix is perfect.

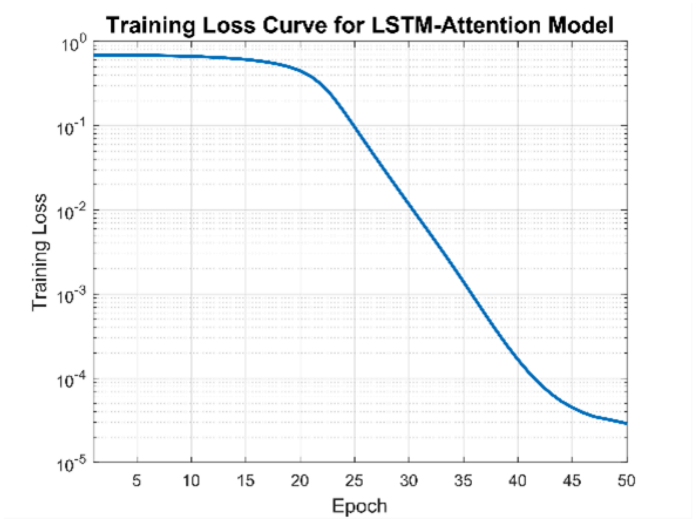


Figure 4. Training loss over 50 epochs for the selected LSTM–Attention model. The model converges smoothly and reaches a stable low loss.

4.2 Simulation and Distributional Validation (Ultra-concise)

We validate the model by generating **1,000–5,000 Monte-Carlo cascades** and comparing their size distribution with empirical BPA cascades. We cap simulations at **20–50 generations** and rescale the spectral radius of **B** to keep cascades stable. Across runs, the **KS statistic** typically lies around **0.47–0.48**, rising to about **0.76** in larger synthetic tests. The

log–log CCDF overlays show that the model captures the heavy-tailed structure but slightly under-represents the largest cascades due to stability constraints.

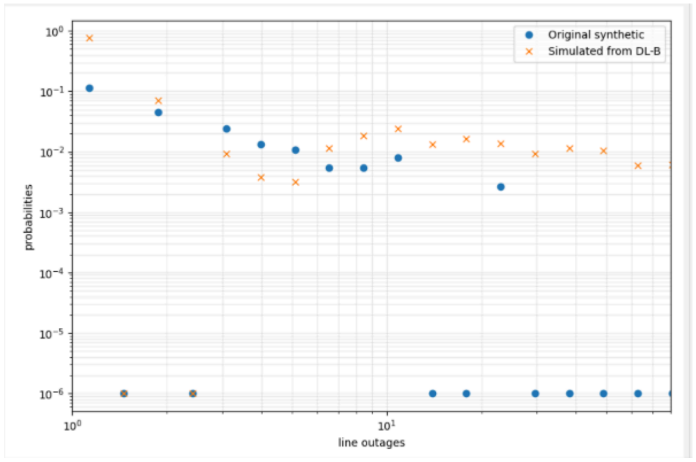


Figure 5. Empirical vs simulated cascade size CCDF.

4.3 Critical-line rankings and stability

For each run, **CascadEye** produces a ranked list of critical transmission lines based on their influence scores in the interaction matrix. In a representative run, Top-15 scores fall roughly between **130–141**, with lines such as **504, 92, 198, 391, 524, 580, 357, 550, 105, 365** appearing consistently at the top. These lines recur across multiple runs and different synthetic-feeder configurations, suggesting that the ranking is stable and not driven by noise or random seeds.

Practically, this means **CascadEye** can highlight a small set of "always risky" components, which users can further inspect through the influence popup in the UI.

Line	Influence score
504	140.6331
92	140.2647
198	139.4976
391	139.0931
524	138.7598
580	138.0401
357	137.5693
550	137.2867
105	136.8229
365	135.6737
261	135.2111

Line	Influence score
541	135.2046
498	131.9585
546	130.2199
352	130.2148

Table 1. Top-15 critical lines ranked by influence score.

4.4 Limitations and failure modes

Our evaluation highlights several limitations. KS values around **0.47–0.76** show that the model does not perfectly match the empirical cascade-size distribution, so it is best viewed as a **diagnostic tool** rather than an exact simulator. Occasional **negative directional-test slopes** also indicate that risk scores can mis-order smaller cascades, reflecting the approximate nature of the learned interaction matrix.

The system currently validates only against **historical outage logs**, not full grid physics or operator behaviors. This means some real-world mechanisms—such as protection actions or dispatch adjustments—are not captured.

Even with these limitations, **CascadEye** works well as an interpretable analysis system. It produces stable critical-line rankings and reasonable heavy-tailed behavior, and the monitoring dashboard makes model drift and structural weaknesses visible to users.

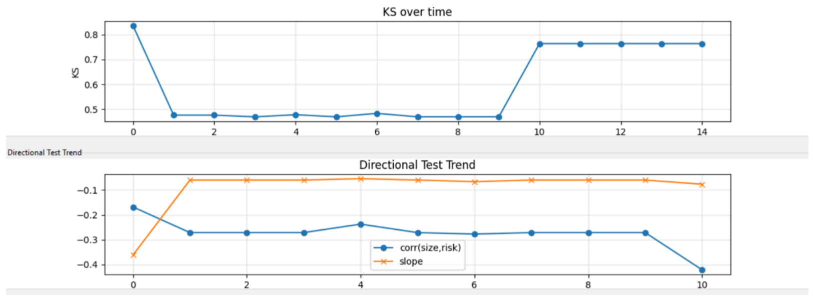


Figure 6. Monitoring tab screenshot showing recent KS statistics and directional-test trends.

5. Application Demonstration

CascadEye is a lightweight desktop application that guides users through the full workflow of cascading-failure analysis. **Figure 7** shows the main interface, organized into six tabs that mirror the ML pipeline: Data, Model, Sequences, Run & Plots, Save & Critical Lines, and Monitoring. This structure keeps the tool simple to navigate and makes each step visually clear.

The workflow begins in the **Data** tab, where users load the outage dataset and view summary statistics. **Figure 8** displays the Dataset Insights window, including cascade counts, maximum generations, unique lines, and a short data guide. Users may also generate a synthetic feeder, with its summary shown in **Figure 9**.

Next, the **Model** tab (**Figure 10**) loads pretrained LSTM–Multi-Head Attention checkpoints and displays architecture metadata, optimizer settings, KPIs, and SHA-1 fingerprints for reproducibility. The **Sequences** tab (**Figure 11**) builds binary tensors and lets users inspect any cascade, generation by generation.

Simulation and validation occur in the **Run & Plots** tab (**Figure 12**), where users run Monte-Carlo cascades, generate CCDF overlays, and perform directional-test validation. Critical-line rankings are handled in the **Save & Critical Lines** tab (**Figure 13**), including pop-up inspection windows. Finally, the **Monitoring** tab (**Figure 14**) visualizes KS drift, directional-test trends, and recent evaluation logs, helping users track model stability.

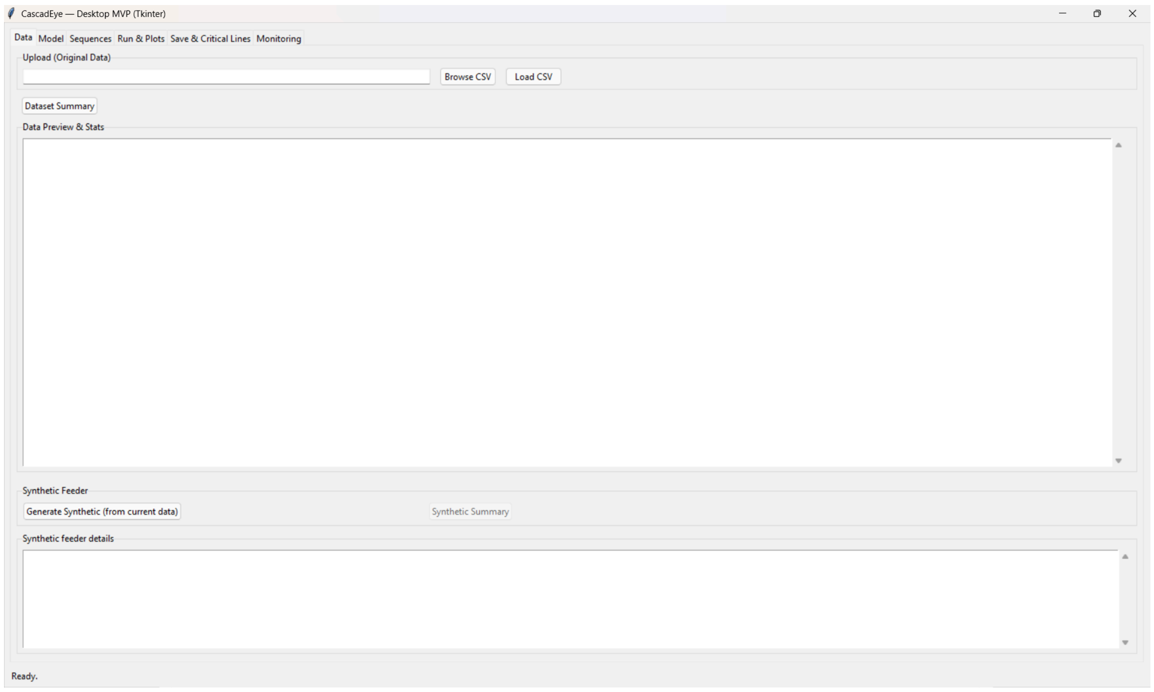
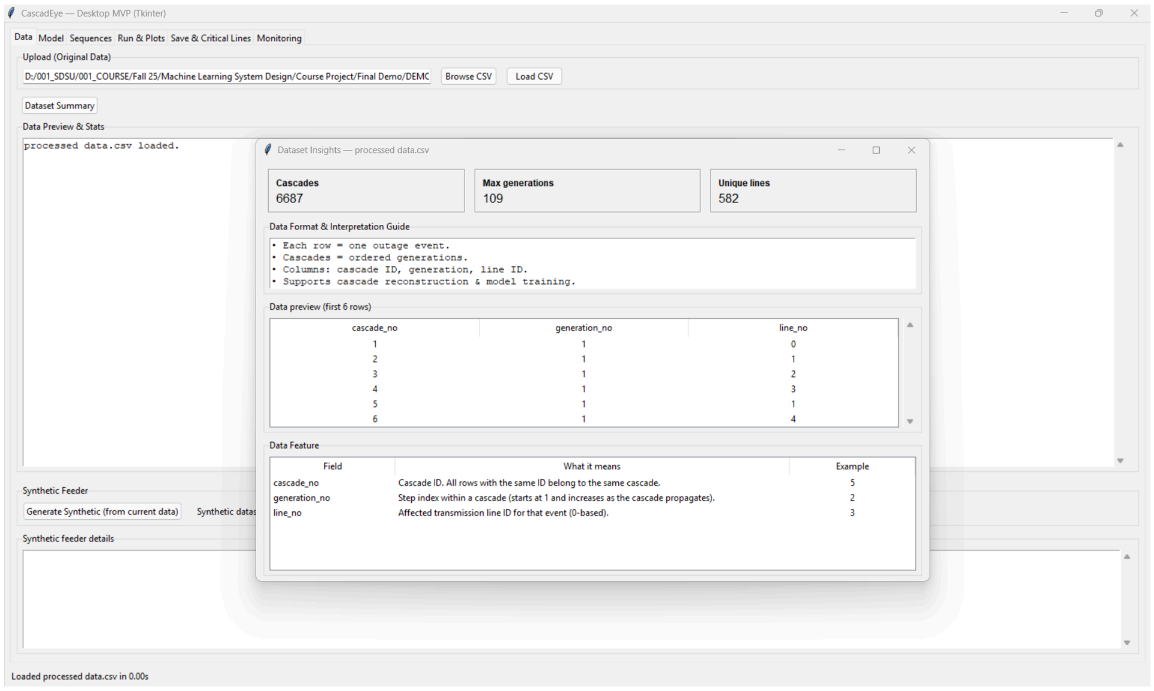


Figure 7. Main CascadEye UI showing the full six-tab layout.



D:/001_SDSU/001_COURSE/Fall 25/Machine Learning System Design/Course Project/Final Demo/DEMC

```
processed data.csv loaded.
```

Cascades	Max generations	Unique lines
6687	109	582

Data Format & Interpretation Guide

```
• Each row = one outage event.
• Cascades = ordered generations.
• Columns: cascade ID, generation, line ID.
• Supports cascade reconstruction & model training.
```

Data preview (first 6 rows)

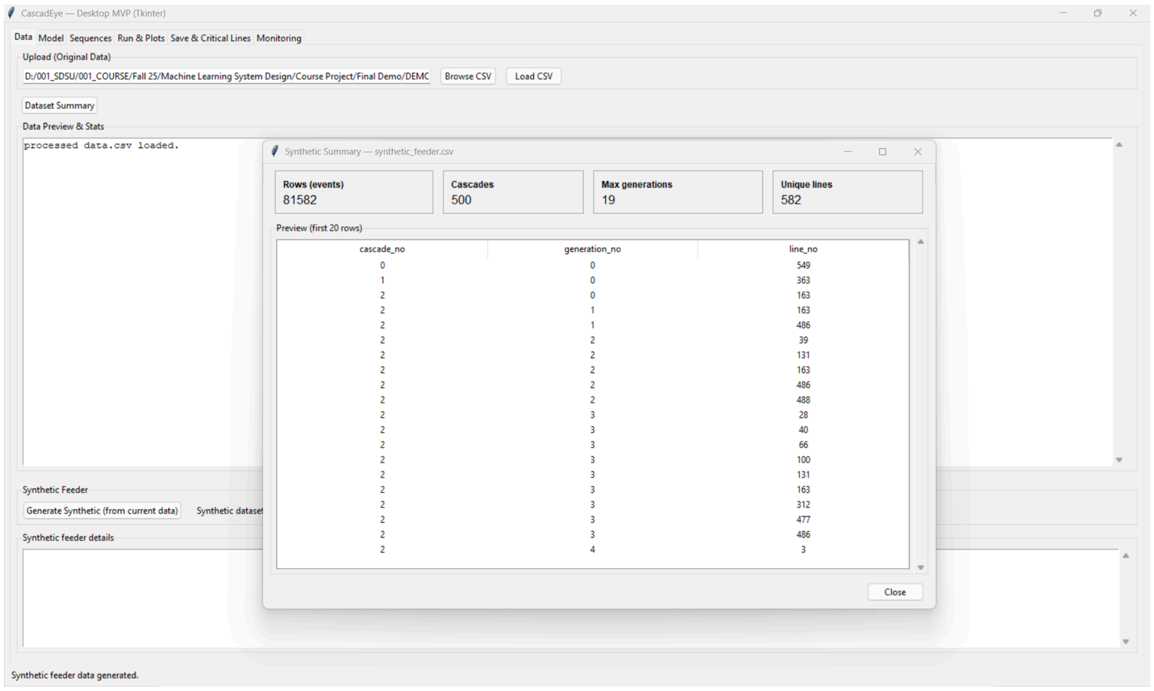
cascade_no	generation_no	line_no
1	1	0
2	1	1
3	1	2
4	1	3
5	1	1
6	1	4

Data Feature

Field	What it means	Example
cascade_no	Cascade ID. All rows with the same ID belong to the same cascade.	5
generation_no	Step index within a cascade (starts at 1 and increases as the cascade propagates).	2
line_no	Affected transmission line ID for that event (0-based).	3

Loaded processed data.csv in 0.00s

Figure 8. Data tab window with cascade statistics and a preview of the outage dataset.



D:/001_SDSU/001_COURSE/Fall 25/Machine Learning System Design/Course Project/Final Demo/DEMC

```
processed data.csv loaded.
```

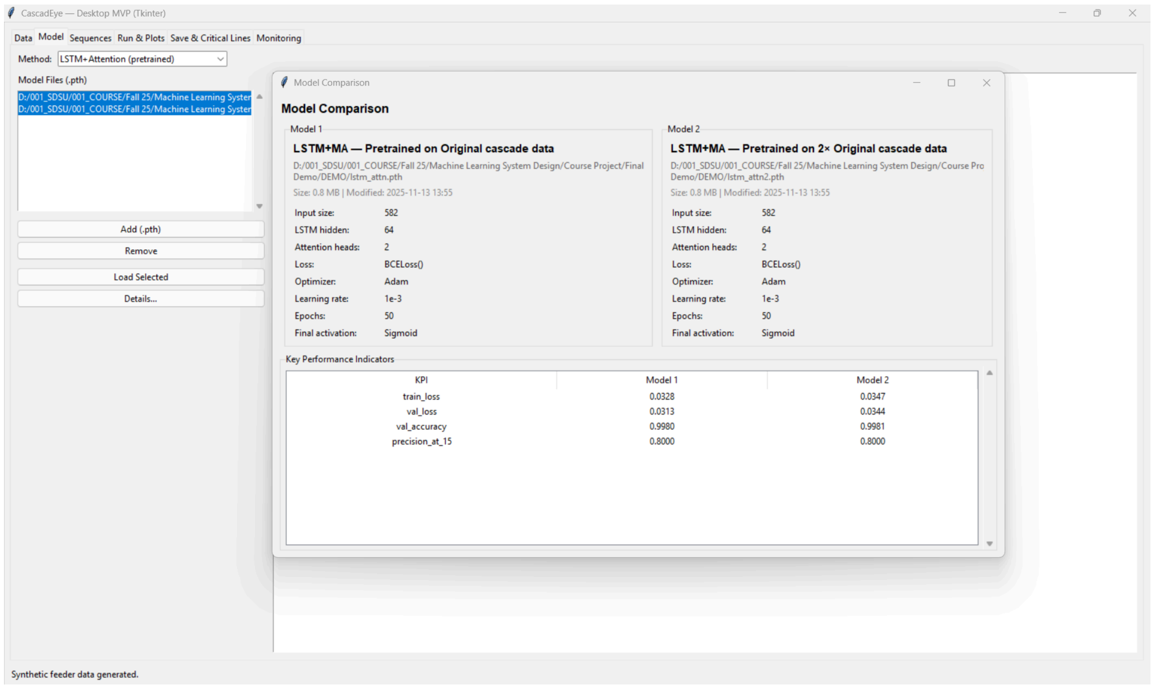
Rows (events)	Cascades	Max generations	Unique lines
81582	500	19	582

Preview (first 20 rows)

cascade_no	generation_no	line_no
0	0	549
1	0	363
2	0	163
2	1	163
2	1	486
2	2	39
2	2	131
2	2	163
2	2	486
2	2	488
2	3	28
2	3	40
2	3	66
2	3	100
2	3	131
2	3	163
2	3	312
2	3	477
2	3	486
2	4	3

Synthetic feeder data generated.

Figure 9. Synthetic feeder summary window generated from the current dataset.



Method: LSTM+Attention (pretrained)

Model Files (.pth)

D:/001_SDSU/001_COURSE/Fall 25/Machine Learning Syster
D:/001_SDSU/001_COURSE/Fall 25/Machine Learning Syster

Model Comparison

Model Comparison

Model 1

LSTM+MA — Pretrained on Original cascade data

D:/001_SDSU/001_COURSE/Fall 25/Machine Learning System Design/Course Project/Final Demo/DEMO/lstm_attn.pth

Size: 0.8 MB | Modified: 2025-11-13 13:55

Input size:	582
LSTM hidden:	64
Attention heads:	2
Loss:	BCELoss()
Optimizer:	Adam
Learning rate:	1e-3
Epochs:	50
Final activation:	Sigmoid

Model 2

LSTM+MA — Pretrained on 2× Original cascade data

D:/001_SDSU/001_COURSE/Fall 25/Machine Learning System Design/Course Pro Demo/DEMO/lstm_attn2.pth

Size: 0.8 MB | Modified: 2025-11-13 13:55

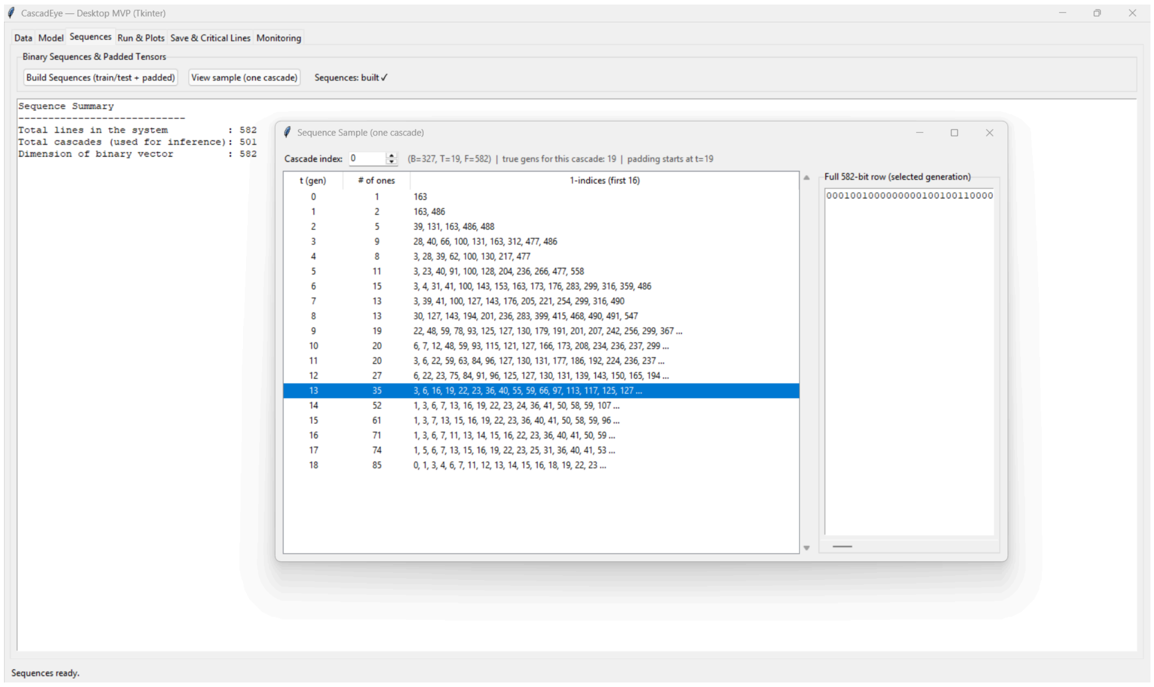
Input size:	582
LSTM hidden:	64
Attention heads:	2
Loss:	BCELoss()
Optimizer:	Adam
Learning rate:	1e-3
Epochs:	50
Final activation:	Sigmoid

Key Performance Indicators

KPI	Model 1	Model 2
train_loss	0.0328	0.0347
val_loss	0.0313	0.0344
val_accuracy	0.9980	0.9981
precision_at_15	0.8000	0.8000

Synthetic feeder data generated.

Figure 10. Model tab showing architecture metadata, KPIs, and checkpoint fingerprints.


CascadEye — Desktop MVP (Tkinter)
Data
Model
Sequences
Run & Plots
Save & Critical Lines
Monitoring
Binary Sequences & Padded Tensors
Build Sequences (train/test + padded)
View sample (one cascade)

Sequences: built ✓

```
Sequence Summary
-----------------------------
Total lines in the system          : 582
Total cascades (used for inference): 501
Dimension of binary vector         : 582
```

Sequence Sample (one cascade)

Cascade index: 0 (B=327, T=19, F=582) | true gens for this cascade: 19 | padding starts at t=19

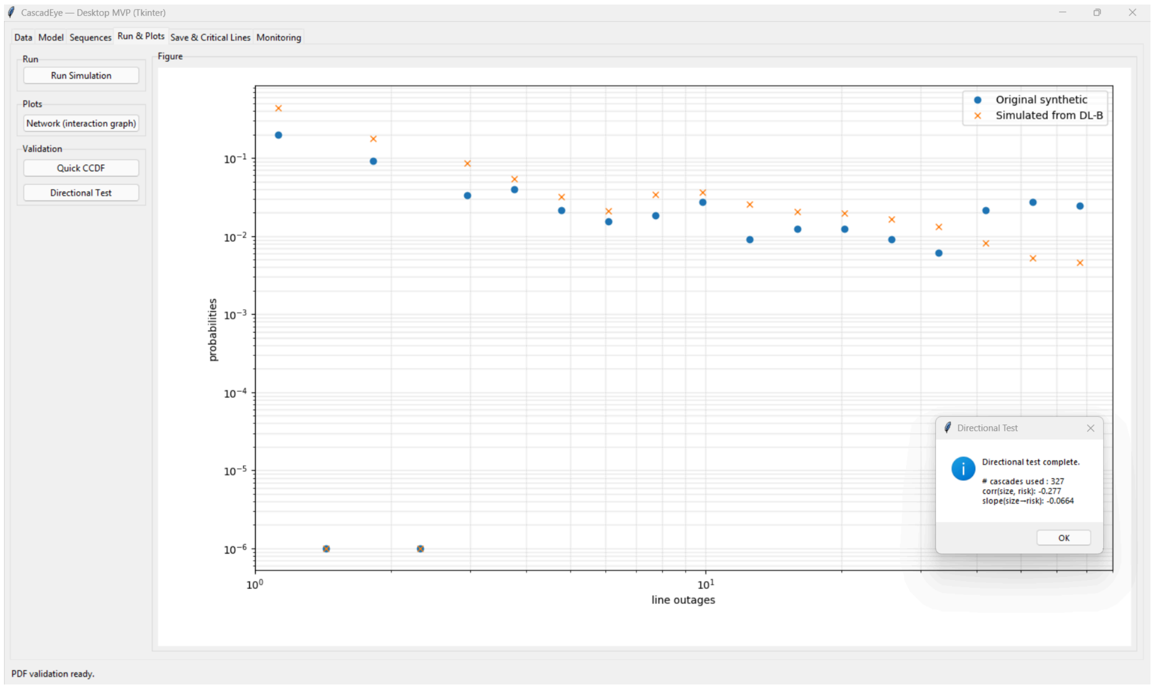
t (gen)	# of ones	1-indices (first 16)
0	1	163
1	2	163, 486
2	5	39, 131, 163, 486, 488
3	9	28, 40, 66, 100, 131, 163, 312, 477, 486
4	8	3, 28, 39, 62, 100, 130, 217, 477
5	11	3, 23, 40, 91, 100, 128, 204, 236, 266, 477, 558
6	15	3, 4, 31, 41, 100, 143, 153, 163, 173, 176, 283, 299, 316, 359, 486
7	13	3, 39, 41, 100, 127, 143, 176, 205, 221, 254, 299, 316, 490
8	13	30, 127, 143, 194, 201, 236, 283, 399, 415, 468, 490, 491, 547
9	19	22, 48, 59, 78, 93, 125, 127, 130, 179, 191, 201, 207, 242, 256, 299, 367 ...
10	20	6, 7, 12, 48, 59, 93, 115, 121, 127, 166, 173, 208, 234, 236, 237, 299 ...
11	20	3, 6, 22, 59, 63, 84, 96, 127, 130, 131, 177, 186, 192, 224, 236, 237 ...
12	27	6, 22, 23, 75, 84, 91, 96, 125, 127, 130, 131, 139, 143, 150, 165, 194 ...
13	35	3, 6, 16, 19, 22, 23, 36, 40, 55, 59, 66, 97, 113, 117, 125, 127 ...
14	52	1, 3, 6, 7, 13, 16, 19, 22, 23, 24, 36, 41, 50, 58, 59, 107 ...
15	61	1, 3, 7, 13, 15, 16, 19, 22, 23, 36, 40, 41, 50, 58, 59, 96 ...
16	71	1, 3, 6, 7, 11, 13, 14, 15, 16, 22, 23, 36, 40, 41, 50, 59 ...
17	74	1, 5, 6, 7, 13, 15, 16, 19, 22, 23, 25, 31, 36, 40, 41, 53 ...
18	85	0, 1, 3, 4, 6, 7, 11, 12, 13, 14, 15, 16, 18, 19, 22, 23 ...

Full 582-bit row (selected generation)

```
0001001000000000100100110000
```

Sequences ready.

Figure 11. Sequences tab showing binary sequence construction and sample previews.

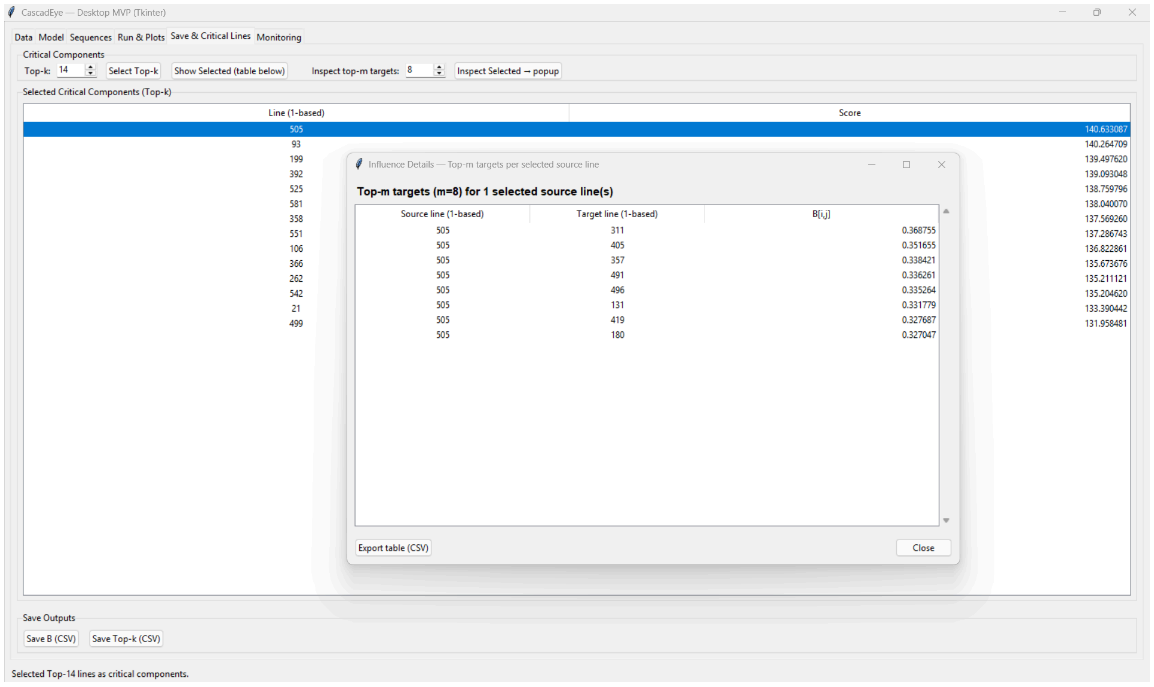


Directional test complete.

cascades used : 327
corr(size, risk): -0.277
slope(size→risk): -0.0664

PDF validation ready.

Figure 12. Run & Plots tab showing CCDF validation results and the directional-test popup.



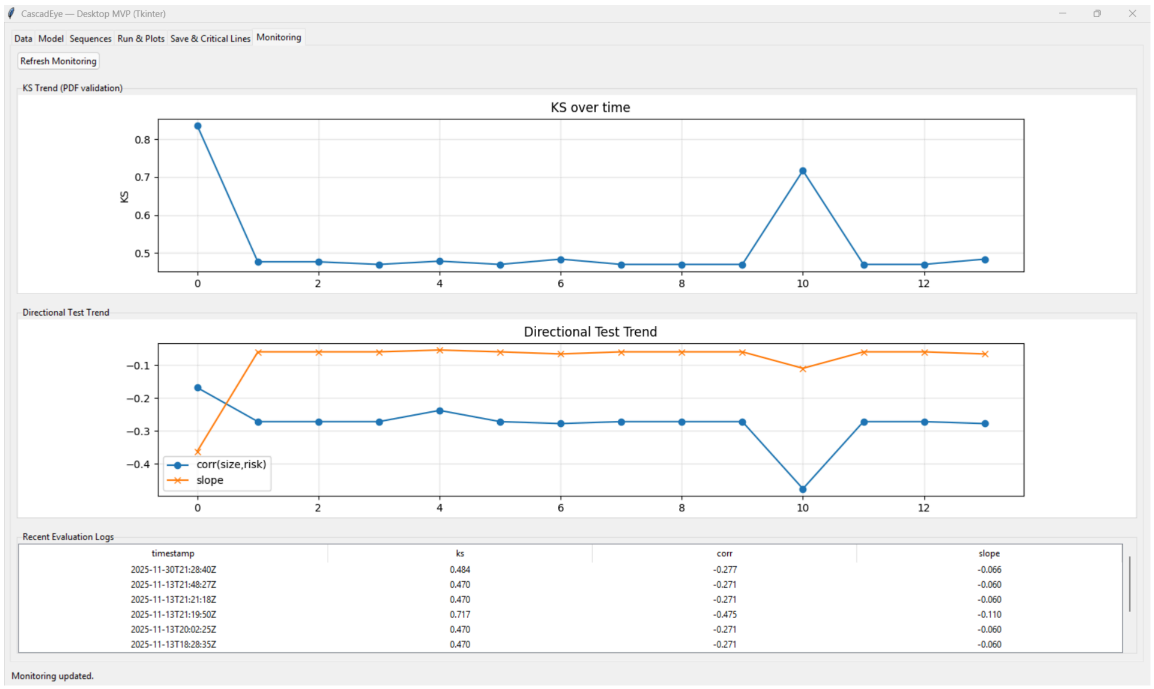
Line (1-based)	Score
505	140.633087
93	140.264709
199	139.497620
392	139.093048
525	138.759796
581	138.040070
358	137.569260
551	137.286743
106	136.822861
366	135.673676
262	135.211121
542	135.204620
21	133.390442
499	131.958481

Top-m targets (m=8) for 1 selected source line(s)

Source line (1-based)	Target line (1-based)	B[i,j]
505	311	0.368755
505	405	0.351655
505	357	0.338421
505	491	0.336261
505	496	0.335264
505	131	0.331779
505	419	0.327687
505	180	0.327047

Selected Top-14 lines as critical components.

Figure 13. Critical-line ranking interface with the influence-popup window for selected lines.



timestamp	ks	corr	slope
2025-11-30T21:28:40Z	0.484	-0.277	-0.066
2025-11-13T21:48:27Z	0.470	-0.271	-0.060
2025-11-13T21:21:18Z	0.470	-0.271	-0.060
2025-11-13T21:19:50Z	0.717	-0.475	-0.110
2025-11-13T20:02:25Z	0.470	-0.271	-0.060
2025-11-13T18:28:35Z	0.470	-0.271	-0.060

Monitoring updated.

Figure 14. Monitoring dashboard showing KS statistics, directional-test trends, and recent evaluation logs.

6. Reflection

This project gave us a chance to design a complete machine-learning system from end to end, and it allowed us to explore how ML, simulation, and MLOps ideas can support cascading-failure analysis. Several things worked well, both technically and in the design of the system.

6.1 What Worked Well

The decision to build a **desktop Tkinter application** proved effective. It kept the tool lightweight, offline, and easy to run on any machine without installing a web server. The tab-based design also worked well because it naturally matched the analysis pipeline and made the workflow easy to follow. On the modeling side, the **LSTM + Multi-Head Attention** architecture was a strong choice. It produced stable interaction matrices, meaningful influence scores, and critical-line rankings that were consistent across runs. The attention mechanism improved interpretability, which is important for a power-systems use case. Our choice to add **structured logging**, SHA-1 fingerprints, and a **Monitoring** tab also worked well. These additions made the system feel more like a real **MLOps** pipeline and helped us catch model drift during testing.

6.2 What Did Not Work and What We Would Improve

The largest challenge came from the simulation component. Although the Monte-Carlo engine performs well, it does not perfectly match the extreme tail of the empirical distribution. This is partly due to the spectral-radius constraint we use for stability, but it also highlights the limits of the learned interaction matrix. Another issue was the directional test. In some runs, the slope becomes negative, which means the risk scores misorder small cascades. This showed us that the model sometimes captures global patterns but struggles with small, noisy sequences. We also discovered that the UI becomes crowded when showing large plots. A more flexible layout or collapsible panels would help. Finally, managing multiple pretrained models became confusing for some users, so we will improve model labeling and configuration management in the future.

6.3 If Given Unlimited Time and Resources

We would expand the system into a more complete cascading-failure analysis platform. First, we would replace a simple Monte-Carlo simulation with a **physics-aware simulator** that incorporates load flow, protection actions, and operator responses. This would allow us to validate B-matrices against richer grid behavior. We would also integrate online (along with offline) learning so that the model can update as new outage sequences arrive. Another addition would be a web-based dashboard for remote use and automated daily monitoring. Finally, we would explore graph neural networks (GNNs), which may better exploit the topology of the transmission network and provide more stable interaction estimates.

6.4 Plans for Moving Forward

We intend to continue developing CascadEye beyond the course. The system already functions as a practical research tool, and we plan to use it in future work on cascading failure modeling. One goal is to package CascadEye as a standalone open-source tool so others can use it with their own datasets. We also plan to test the system on additional utility outage logs and explore community-based EM methods as a comparison. Overall, the project gave us a strong foundation and a clear path for further development.

7. Broader Impacts

CascadEye is designed to help researchers and utilities understand how cascading failures spread across transmission networks. Its intended use is analytical and educational. The system provides a structured way to explore historical outage patterns, test data-driven interaction models, and identify components that appear consistently influential. In this sense, **CascadEye** supports better planning and reliability studies, especially for teams that do not have access to large simulation platforms.

However, the tool also has potential unintended uses. Because **CascadEye** highlights critical transmission lines and shows how failures propagate, there is a risk that the information could be misinterpreted or used to exaggerate system vulnerabilities. The model also simplifies real grid behavior, which might lead inexperienced users to draw incorrect operational conclusions. Another concern is over-trust: users might assume that the B-matrix or the Monte-Carlo simulations represent the physical power system with full accuracy, even though they are approximations based only on outage logs.

To mitigate these risks, we made several design choices. First, the system runs **fully offline** and does not fetch or share data externally, reducing exposure of sensitive grid information. We also include validation plots (PDF, CCDF, KS, directional tests) to help users see where the model diverges from real data. This transparency encourages cautious interpretation. Finally, the system logs every run with SHA-1 hashes, making it possible to trace results back to specific datasets and models. This helps prevent accidental misuse and supports responsible research practices. Overall, **CascadEye** aims to contribute positively to grid-reliability research while promoting safe and informed use.

References

References

1. J. Qi, “Utility outage data-driven interaction networks for cascading failure analysis and mitigation,” *IEEE Transactions on Power Systems*, vol. 36, no. 2, pp. 1409–1418, Mar. 2021.

2. J. Qi, J. Wang, and K. Sun, “Efficient estimation of component interactions for cascading failure analysis by EM algorithm,” *IEEE Transactions on Power Systems*, vol. 33, no. 3, pp. 3153–3161, May 2018.

3. J. Qi, K. Sun, and S. Mei, “An interaction model for simulation and mitigation of cascading failures,” *IEEE Transactions on Power Systems*, vol. 30, no. 2, pp. 804–819, Mar. 2015.

4. SciPy Developers, *SciPy: Scientific Library for Python*. Available at: https://scipy.org

5. Tkinter Documentation, *Python Standard GUI Framework*. Available at: https://docs.python.org/3/library/tkinter.html

6. CSC 792 — Machine Learning System Design, Fall 2025. *Course materials and lectures on system design, offline/online evaluation, and lightweight MLOps* (used for system-design and evaluation principles).