Unsupervised Anomaly Detection for High-Energy Physics Using VAEs

•••

AI Model For Physics
Hero Rfaat Mohammed
Academic Year: 2023/2024

Project Overview

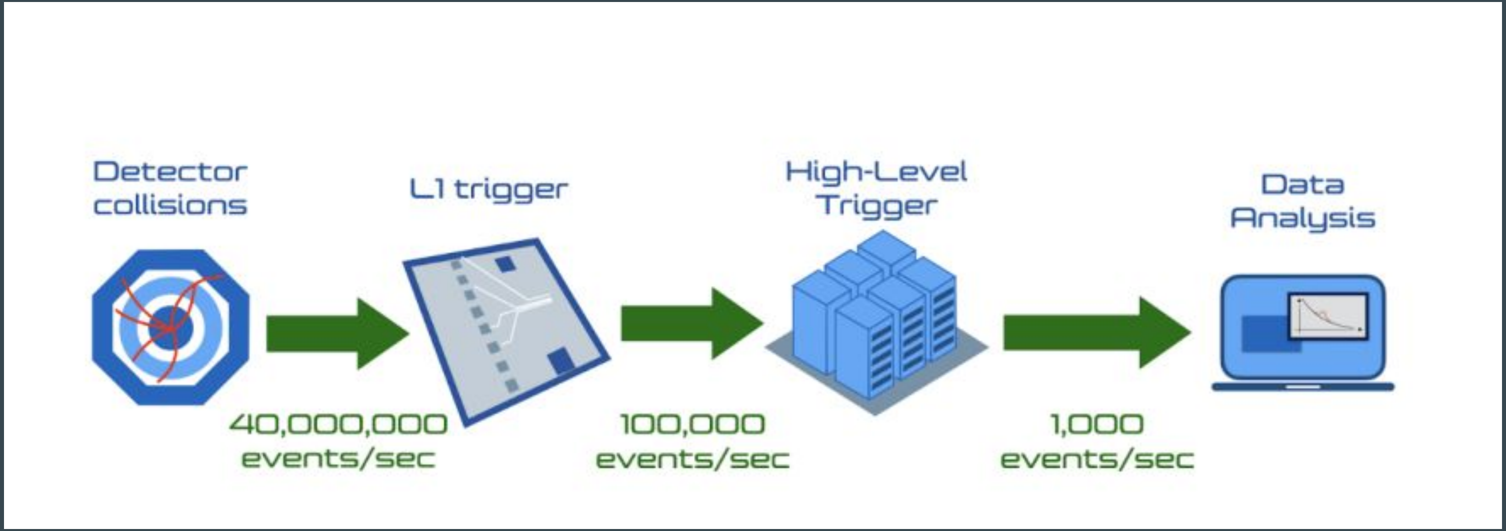
Objective: Develop an unsupervised anomaly detection framework using Variational Autoencoders (VAEs) to identify potential new physics phenomena in LHC data.

Focus: Detecting events deviating from the Standard Model (SM) in high-energy physics.

Dataset: Simulated data streams from LHC detectors, including a "blackbox" dataset with new physics events.

Dataset Description

- Format: Hierarchical Data Format version 5 (HDF5).
- Components:
  - "Particles": Contains event data (N, 19, 4) with N as the number of events.
  - "Particles_Classes" and "Particles_Names".
- Physics Objects: MET, electrons, muons, and jets, characterized by transverse momentum (pT), pseudorapidity (η), and azimuthal angle (ϕ).



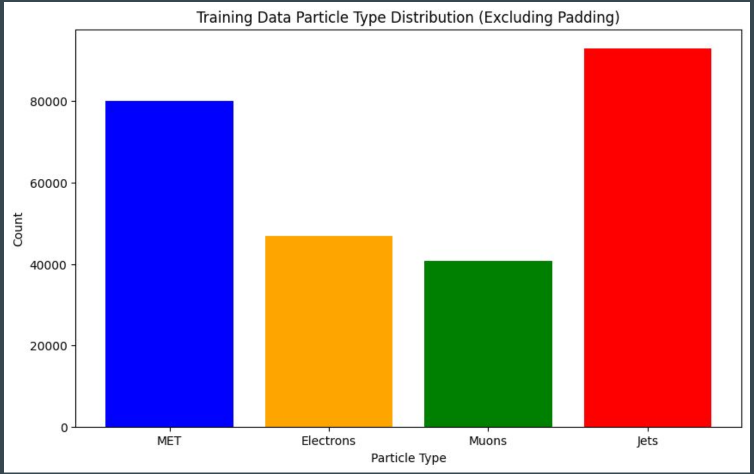
Data Preprocessing

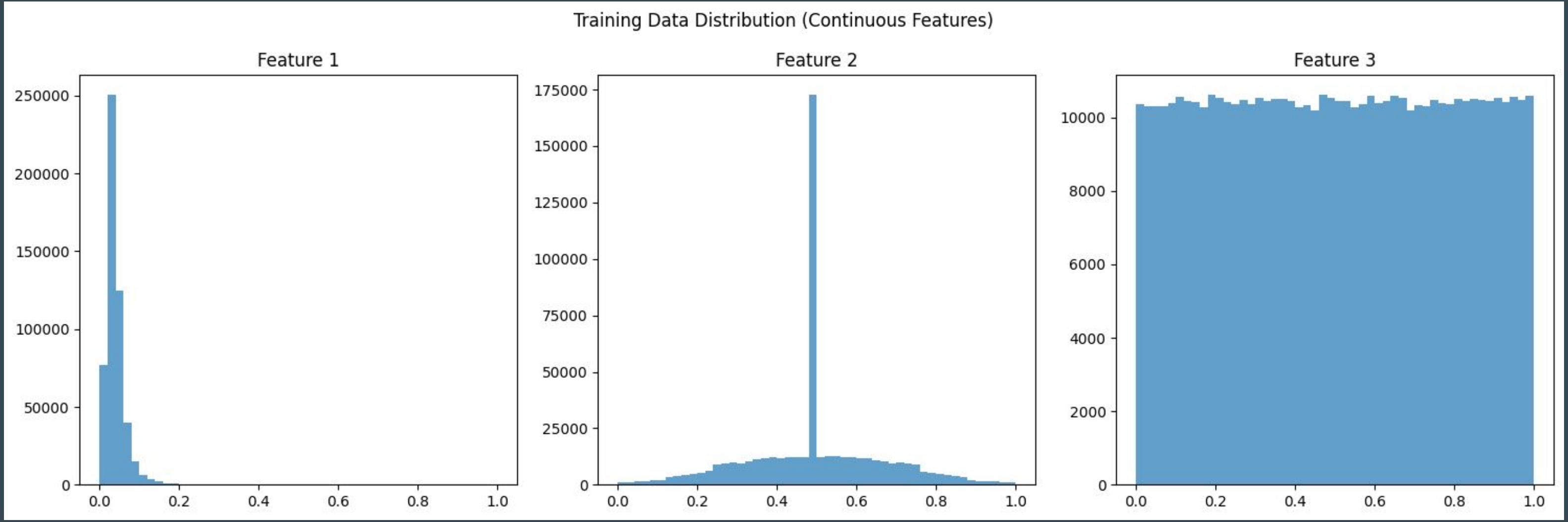
Normalization: Continuous features normalized using min-max scaling.

One-Hot Encoding: Categorical features, representing particle types.

Data Split:

- Training (SM data)
- Validation (hyperparameter tuning)
- Testing (BSM data)
- Blackbox data for final evaluation.


Training Data Particle Type Distribution (Excluding Padding)
Count
80000
60000
40000
20000
0
MET
Electrons
Muons
Jets
Particle Type


Training Data Distribution (Continuous Features)
Feature 1
Feature 2
Feature 3
250000
200000
150000
100000
50000
175000
150000
125000
100000
75000
50000
25000
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0

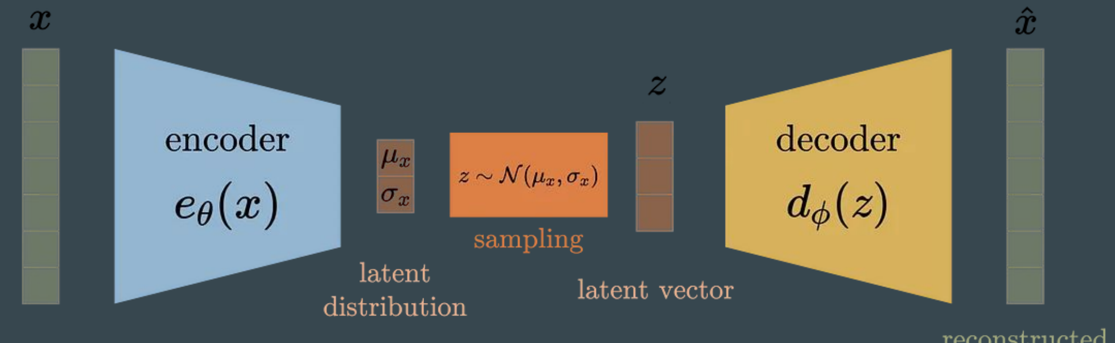
Methodology: Variational Auto Encoders (VAEs)

Structure: VAEs learn parameters of a probability distribution (mean and variance) for latent vectors.

Regularization: Kullback-Leibler (KL) divergence for normal distribution approximation.

Training: Minimize reconstruction errors, flag high errors as potential anomalies.

Hyperparameter Tuning: Used Optuna for optimization.

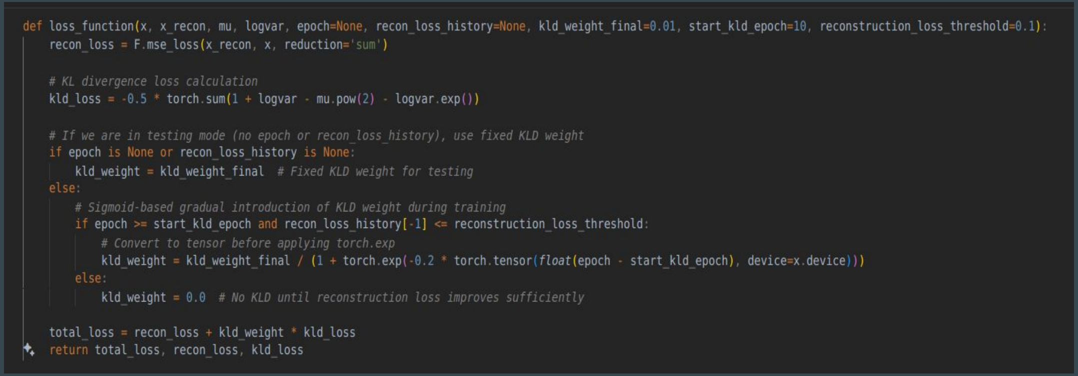


$$\text{reconstruction loss} = \|x - \hat{x}\|_2 = \|x - d_\phi(z)\|_2 = \|x - d_\phi(\mu_x + \sigma_x \epsilon)\|_2$$

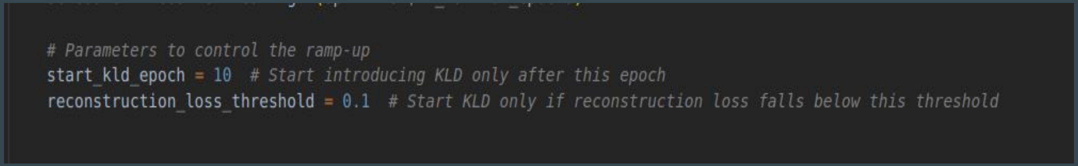
$$\mu_x, \sigma_x = e_\theta(x), \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\boxed{\textit{similarity loss} = \textit{KL Divergence} = D_{KL}(\mathcal{N}(\mu_x, \sigma_x) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I}))}$$

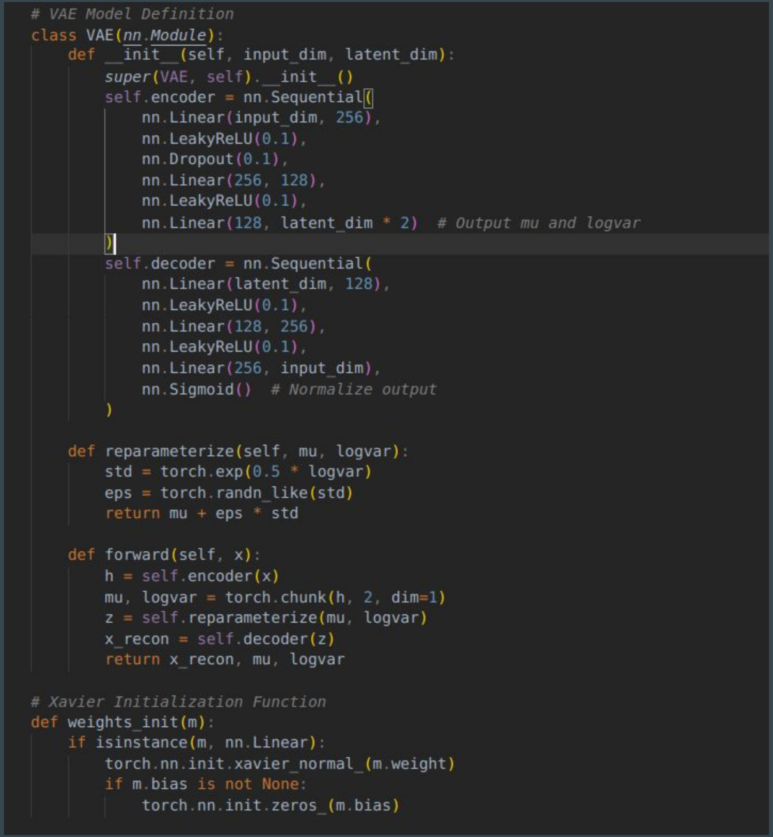
$$\boxed{\textit{loss} = \textit{reconstruction loss} + \textit{similarity loss}}$$



```python
def loss_function(x, x_recon, mu, logvar, epoch=None, recon_loss_history=None, kld_weight_final=0.01, start_kld_epoch=10, reconstruction_loss_threshold=0.1):
    recon_loss = F.mse_loss(x_recon, x, reduction='sum')

    # KL divergence loss calculation
    kld_loss = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

    # If we are in testing mode (no epoch or recon_loss_history), use fixed KLD weight
    if epoch is None or recon_loss_history is None:
        kld_weight = kld_weight_final  # Fixed KLD weight for testing
    else:
        # Sigmoid-based gradual introduction of KLD weight during training
        if epoch >= start_kld_epoch and recon_loss_history[-1] <= reconstruction_loss_threshold:
            # Convert to tensor before applying torch.exp
            kld_weight = kld_weight_final / (1 + torch.exp(-0.2 * torch.tensor(float(epoch - start_kld_epoch), device=x.device)))
        else:
            kld_weight = 0.0  # No KLD until reconstruction loss improves sufficiently

    total_loss = recon_loss + kld_weight * kld_loss
    return total_loss, recon_loss, kld_loss
```



```python
# Parameters to control the ramp-up
start_kld_epoch = 10  # Start introducing KLD only after this epoch
reconstruction_loss_threshold = 0.1  # Start KLD only if reconstruction loss falls below this threshold
```



```python
# VAE Model Definition
class VAE(nn.Module):
    def __init__(self, input_dim, latent_dim):
        super(VAE, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 256),
            nn.LeakyReLU(0.1),
            nn.Dropout(0.1),
            nn.Linear(256, 128),
            nn.LeakyReLU(0.1),
            nn.Linear(128, latent_dim * 2)  # Output mu and logvar
        )
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 128),
            nn.LeakyReLU(0.1),
            nn.Linear(128, 256),
            nn.LeakyReLU(0.1),
            nn.Linear(256, input_dim),
            nn.Sigmoid()  # Normalize output
        )

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def forward(self, x):
        h = self.encoder(x)
        mu, logvar = torch.chunk(h, 2, dim=1)
        z = self.reparameterize(mu, logvar)
        x_recon = self.decoder(z)
        return x_recon, mu, logvar

# Xavier Initialization Function
def weights_init(m):
    if isinstance(m, nn.Linear):
        torch.nn.init.xavier_normal_(m.weight)
        if m.bias is not None:
            torch.nn.init.zeros_(m.bias)
```

Results

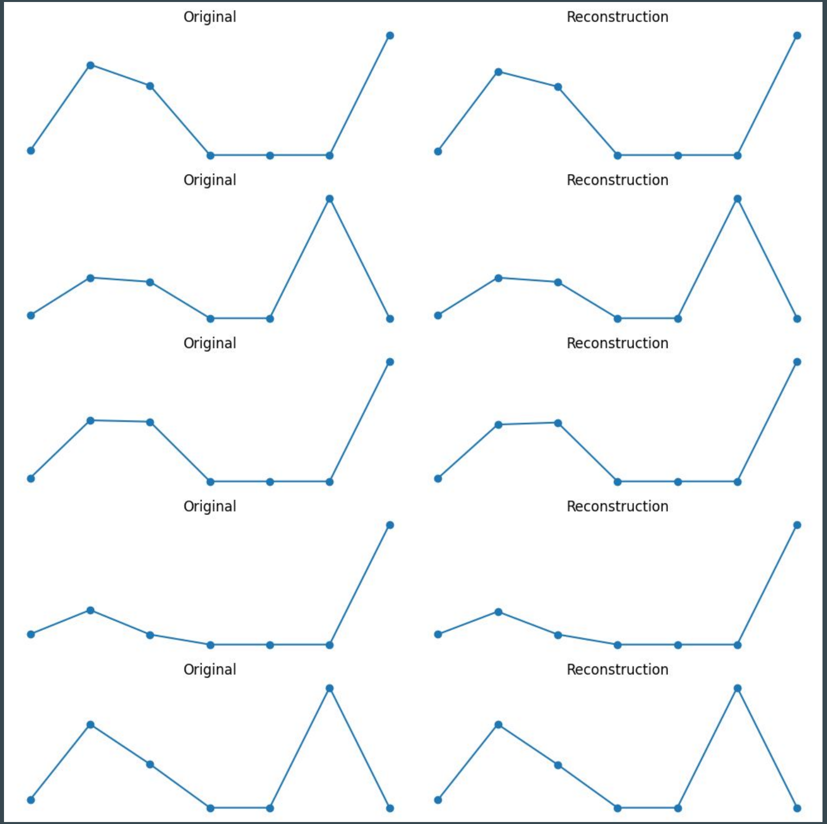
Anomaly Detection: Assessed by calculating reconstruction error for each event.

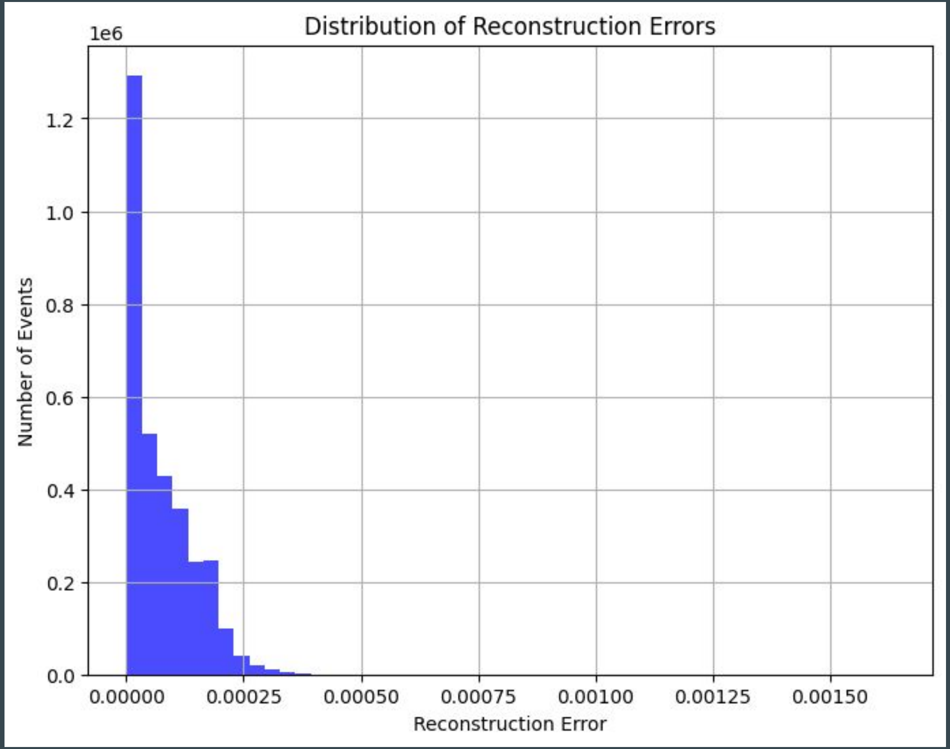
Blackbox Dataset: Used to test model's ability to detect unknown anomalies (BSM events).

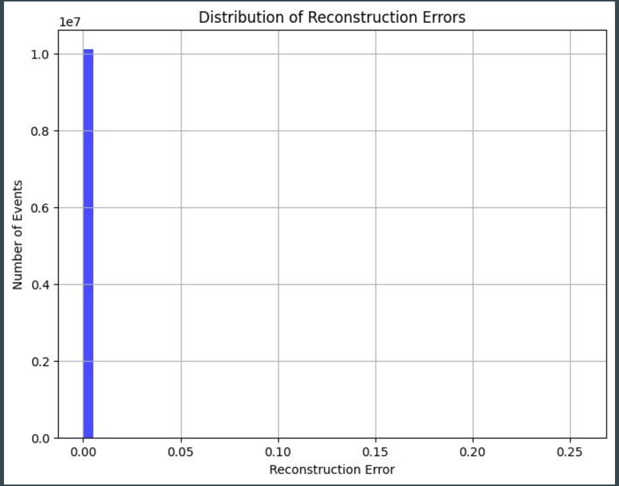
VAE Effectiveness: Reconstructed normal events, forming clusters in the latent space.

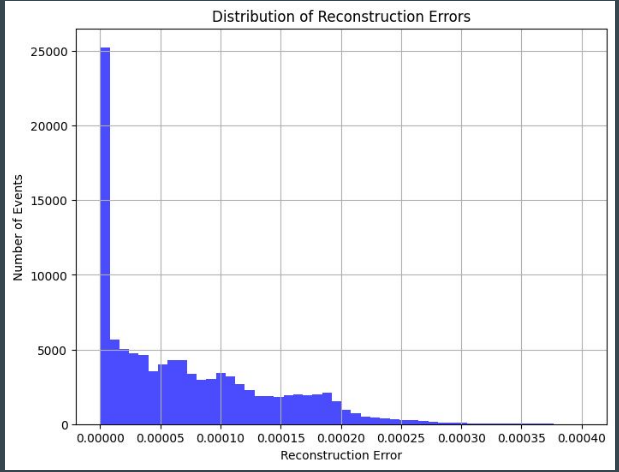
Anomalies: Scattered within clusters, indicating some level of detection but not entirely separated.

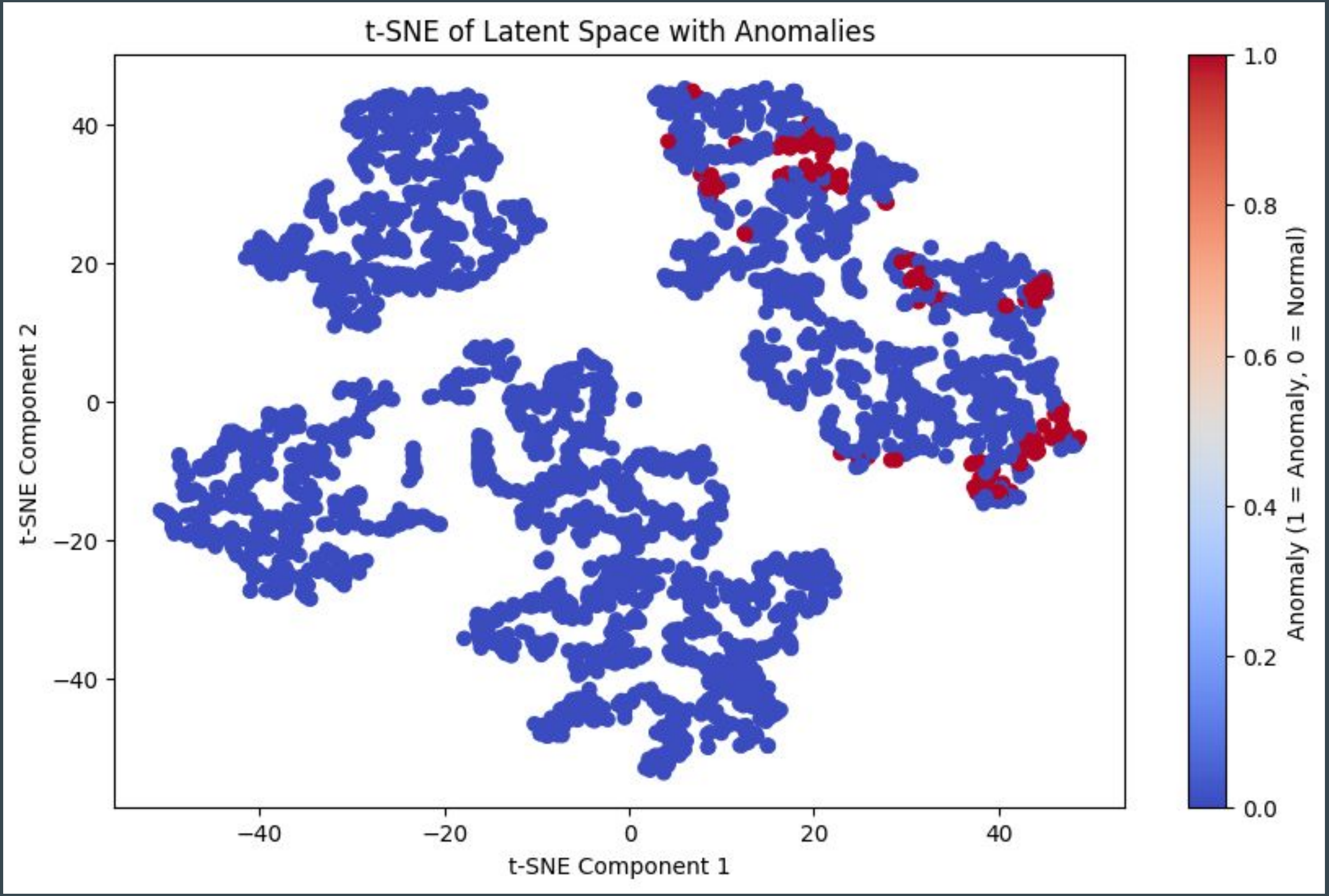
Limitation: No access to ground truth labels for blackbox data, limiting full evaluation of model performance.




Distribution of Reconstruction Errors
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Number of Events
0.00000
0.00025
0.00050
0.00075
0.00100
0.00125
0.00150
Reconstruction Error


Distribution of Reconstruction Errors
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Number of Events
0.00
0.05
0.10
0.15
0.20
0.25
Reconstruction Error


Distribution of Reconstruction Errors
25000
20000
15000
10000
5000
0
Number of Events
0.00000
0.00005
0.00010
0.00015
0.00020
0.00025
0.00030
0.00035
0.00040
Reconstruction Error


t-SNE of Latent Space with Anomalies
t-SNE Component 1
t-SNE Component 2
Anomaly (1 = Anomaly, 0 = Normal)
−40
−20
0
20
40
1.0
0.8
0.6
0.4
0.2
0.0