Transformers for Failure Prediction in High-Performance Computing System Logs

Halil Ozgur Demir Allison Austin Amey Gohil Tejas Patil

ECS 289L, Spring 2024

Motivation

High-Performance Computing (HPC) is widely used in problem-solving for many different fields and applications. HPC systems are composed of thousands of connected hardware components running in parallel to support applications solving large problems in science. These complex systems require effective monitoring techniques to ensure successful application runs and save computation time and resources. Current efforts to predict anomalies rely on human intervention, and there is a substantial demand for machine learning (ML) frameworks that support online anomaly detection [1]. Due to their popularity, transformer-based models, particularly large-language models (LLMS), have led the most recent efforts to close this gap in HPC system monitoring. **The aim of this project is to explore the applications of LLMs in predicting anomalous log events and detecting failures within large, complex systems.**

Background

Transformer-based models are designed to handle sequential data more effectively and efficiently than previous models like RNNs and LSTMs [13]. This is because of Transformers' ability to more effectively handle long-term relations in sequential data using its self-attention mechanism. This feature of Transformers makes them suitable for log-based prediction [2] because HPC failures typically result from degraded performance over long periods of time [7]. Large language models (LLMs) employ transformer architectures, which leverage self-attention mechanisms to effectively process language based on the context provided by the input data [11]. Assessing the health of HPC systems requires analyzing several log messages and determining relationships between events, hardware components, and jobs. LMs are highly suited for this task because they demonstrate rich skill sets in natural language processing (NLP) of time series data.

There are two LLMs that we will be experimenting with on our datasets. The first is minGPT, which is a simpler GPT-2 model implementation that is available on GitHub [9]. It uses standard transformer blocks which are transformers followed by feed-forward layers. The second transformer-based model we will be using is a Bidirectional Encoder Representation from Transformers (BERT) model. Anomaly detection by applying BERT on HPC log data has been done previously, and we will be following the implementation of the state-of-the art LogBERT model [8], which is also available on GitHub. BERT captures the contextual information of whole log sequences by fusing

left and right contexts, which we predict will achieve better results for predicting anomalies in the system logs [5].

Methods

Data Preprocessing

We used two different HPC log time series datasets for training our LLMs. The first dataset is smaller and low-frequency, and was collected from the System 20 of the high performance computing cluster at the Los Alamos National Laboratories [12]. There are 432,260 log messages and 80 unique log events. Of these 80 log events, we deemed 21 of them to indicate failure states. We also flagged an event as 'failure' if the State of the log message contains 'error'. The second dataset was collected from a BlueGene/L (BGL) supercomputer system at Lawrence Livermore National Labs [6]. This dataset is larger and is more densely sampled. The BGL logs contain alert and non-alert messages, and the alert messages are considered as anomalous. Out of the 4,747,963 total log messages in the BGL dataset, 348,460 are anomalous. Table 1 shows the error event IDs and templates we used to determine failures in the System 20 dataset.

To represent log messages, we used the Drain log parser algorithm to extract log keys from log messages [10]. These log keys are used to build a vocabulary for training. We then built feature vectors representing sequences of logs contained within specific time windows. For BGL data, we defined a time sliding window as 5 minutes and a step size of 1 minute to generate log sequences, where the average length is 562. This works well for this dataset since the data is high-frequency. For the System 20 dataset, we used a hybrid window approach to generate log sequences. For this approach, the logs are split into time-based and event-based windows based on a fixed window size of 15 minutes and 10 log events. After processing the System 20 logs, we split the dataset into 40 percent train and 60 percent test; totalling 20,256 training log sequences and 30,384 testing log sequences with an average length of 9 log events in each sequence. A summary of the log sequences can be shown in Table 2.

GPT

We utilized the minGPT model, a simplified implementation designed for educational purposes and based on the GPT-2 architecture [11]. This model comprises 12 transformer and feedforward layers with an embedding size of 768. To tailor the model to our specific needs, we created new vocabularies for each dataset, mapping event IDs to unique tokens. This approach allowed us to handle log data more effectively, converting log events into token sequences suitable for the model. Figure 1 provides an overview of our GPT model architecture.

For training, the dataset was prepared to predict a single new token given a block of preceding tokens of a fixed size. During validation, we predicted a sequence of tokens equal to the block size, starting with an initial block of tokens. The model generated each future token iteratively, using its previous predictions to inform the next step.

BERT

Following the open-source LogBERT implementation, we constructed a Transformer encoder by using four Transformer layers. Each transformer layer includes a multi-head self-attention with 4 heads and a position-wise feed forward sub-layer of size 256. The multi-head attention uses a parallel of self-attentions to jointly capture information from different log keys. After concatenating the parallel heads and applying ReLU activation to the feed forward sub-layer, the transformer layer returns the contextualized embedding. This is passed into a fully connected layer and softmax is used to get the probability distribution of the token over the vocabulary. Cross-entropy loss is then used for the masked log key prediction task. Figure 4 provides an overview of our BERT model architecture.

The total training time for the BERT model with the BGL dataset was around 50 minutes using a T4 GPU in Google Colab. The training time for the System 20 dataset was much slower, so we trained BERT for only 50 epochs, which took approximately 55 minutes to complete using MPS backend on an Apple M2 Max chip. While the loss decreased while training BERT for both datasets, the model was able to converge on the BGL dataset. This is likely due to the differences in data granularity, size, sequencing, and leaving the model parameters unchanged for each dataset.

Probabilistic Inference

In classical autoregressive inference, GPT predicts each token sequentially, appending each predicted token to the end of the input sequence before predicting the next token. This method is effective when exact token sequences are necessary, but it may not be optimal for tasks where overall prediction quality is more important than exact token sequences. For anomaly prediction, where the goal is to foresee potential errors rather than predict specific tokens, we can use a probabilistic inference method. This method feeds probabilities instead of token predictions during inference. To do this, we first set the probabilities of known tokens to 1 at their respective indices. After predicting a token, we use softmax probabilities instead of selecting the highest probability token. By using generated probability distributions, we are maintaining the probabilistic sequence. After completing the sequence prediction, we select the token with the highest probability for error prediction.

Attention Mechanism Variation

To further improve prediction results, we tested whether variations of the standard attention mechanism would be able to capture complex dependencies in the input sequences. We explored two different variations of attention mechanism–additive attention [3] and hierarchical attention [4]. When training using additive attention, we replaced the simple attention module in the multi-headed attention block with an attention module that uses a single-layer feedforward network with hyperbolic tangent nonlinearity to compute the weights. We denote W_1 and W_2 as the matrices corresponding to the linear layer and v as a scaling factor. h is the size of the embeddings and s is the sequence length. The query and key vectors from the transformer block are passed into the linear layer, and we add the option to use an additional attention mask and dropout.

For the hierarchical attention mechanism, we replaced the multi-headed attention in each trans-

former block with a Hierarchical Attention Network (HAN) which processes sequences at two hierarchical levels: words and sentences (aka log events and sequences). First, it computes word-level attention within each sequence, creating a summary vector. The sequence of summary vectors is used to compute sentence-level attention, which produces a final representation of the sequence. We trained the additive attention model using the same number of epochs as the baseline models (200 for BGL and 50 for Sys20), but reduced the number of epochs for the hierarchical attention model to 20 for BGL and 10 for Sys20 due to resource usage limits.

Evaluation

The baseline models and variations were given the task of predicting anomalous sequences in each dataset. For the GPT models, we predicted future logs and according to the predicted sequence, marked the actual sequence accordingly. We checked the appearance of error logs in the predicted sequence, and if the sequence included an error, then we marked the actual sequence as anomalous as it led to error generation. For the BERT models, we derived the anomalous score of the log sequence prediction from the mask tokens. The probability distributions tell us the likelihood of a log key appearing in the position of each mask token. Using the top *num_candidates* highest likelihoods, if the real log key is in the candidate set, then the key is normal. If the observed log key is not in the candidate set, it is abnormal. If the log sequence consists of r anomalous log keys, then the sequence is labeled anomalous. We evaluated the task performance of all models based on accuracy, precision, recall, and f-1 score of anomalous sequence predictions.

Results

Evaluation metrics for each model trained on both datasets is shown in Table 3. The acronyms PI, AA, and HA denote models with probabilistic inference, additive attention, and hierarchical attention, respectively. The baseline BERT model performed the best on the BGL dataset, and the worst for the System 20 dataset. We see a decrease in all performance metrics for BGL data when AA is added to the BERT model, but results improve with HA and are comparable to baseline. The baseline BERT model achieved a 95.61 accuracy score and an 88.75 F-1 score during evaluation, which was the highest F-1 score for all models across both datasets. For the System 20 dataset, changing the attention mechanism substantially improved results. Using HA resulted in an anomalous log sequence prediction accuracy of 81.61 and a precision of 82.53. Using BERT with AA achieved the highest recall and F-1 score for the System 20 dataset. We attribute this difference in prediction results to the nature of the datasets, as the benefits of AA and HA might be more pronounced in a sparser time series.

Conclusion

Overall, the GPT model performed worse than the BERT model. In log anomaly prediction, the number of anomalies forms a small portion of datasets (4.5% of Sys20, 6.8% of BGL). This leads to predictions of anomalous logs less frequently. On the other hand, as the BERT model is trained on normal data, it doesn't have this problem. Therefore, this leads to high precision and low recall, as

it predicts errors less frequently but more confidently. For future work, we could experiment with a BERT model that uses a combination of hierarchical attention network and additive attention.

Probabilistic inference increased the performance in the Sys20 dataset while decreasing the performance in the BGL dataset. One of the reasons for this may be because of the number of log keys in each dataset. When there are more possible tokens, probabilities are distributed more compared to less possible tokens. This leads to very close probabilities after some iterations. Because of this close probability distribution, the model becomes less confident about each token after each iteration. Therefore we are losing the advantage of probabilistic inference when we have more log keys. This limitation can be fixed by selecting top_k probabilities before feeding those for a new iteration.

Contribution

Allison provided the datasets, handled the preprocessing of the logs, generated train and test sets, worked on BERT model code, and worked on the additive and hierarchical attention mechanisms. Ozgur worked on the minGPT training and evaluation code with our datasets, came up with model improvement methods, and worked on the probabilistic inference method for GPT.

References

[1] O. Tuncer et al. "Online Diagnosis of Performance Variation in HPC Systems Using Machine Learning". In: *IEEE Transactions on Parallel and Distributed Systems* (2019).

[2] Khalid Ayedh Alharthi. "Clairvoyant: a log-based transformer-decoder for failure prediction in large-scale systems". In: *Proceedings of the 36th ACM International Conference on Supercomputing* 35 (2022), pp. 1–14.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *arXiv preprint arXiv:1409.0473* (2014). URL: https://arxiv.org/abs/1409.0473.

[4] Hongyu Bi, Lijun Lu, and Yu Meng. "Hierarchical attention network for multivariate time series long-term forecasting". In: *Applied Intelligence* 53 (2023), pp. 5060–5071.

[5] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[6] Om Duggineni. *Loghub BGL Log Data*. https://www.kaggle.com/datasets/omduggineni/loghub-bgl-log-data. Accessed: 2024-05-27. 2024.

[7] Haryadi Gunawi and Riza Suminto. *Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems*. 2018.

[8] Haixuan Guo. *LogBERT*. https://github.com/HelenGuohx/logbert. Accessed: 2024-05-20. 2021.

[9] Andrej Karpathy. *minGPT*. https://github.com/karpathy/minGPT. Accessed: 2024-05-17. 2024.

[10] LogPAI. *LogParser: Log File Parsing*. https://github.com/logpai/logparser/tree/main. Accessed: 2024-05-20. 2022.

[11] et al Radford A. "Language Models are Unsupervised Multitask Learners". In: *IEEE International Symposium on Software Reliability Engineering* (2019).

[12] *Ultra-scale Systems Research Center (USRC) Failure Data Repository*. https://lanl.gov/projects//ultrascale-systems-research-center/data/failure-data.php.

[13] Ashish Vaswani et al. "Attention is all you need". In: *Advances in Neural Information Processing Systems*. 2017.

Figures

Table 1: System 20 Error Events

EventID	**Template**
e7ede03f	NIFF: node node-<*> detected a failed network connection on network <*> via interface <*>
e07f96ea	ServerFileSystem: An ServerFileSystem domain panic has occurred on <*>
3dff51fb	Command Failed on: <*>
bd82b62d	Failed subcommands <*>
562e2cee	Link error on broadcast tree Interconnect-<*>:<*>
b02b9741	not responding
74e8333a	down
252f035f	not-responding
683ade8b	fan<*> has failed
7e85bcb6	critical
5cdfd2f3	<*> (command <*>) Error: Unknown We failed to determine the state of this node
7b80ea70	<*> (command <*>) Error: <*> <*> <*> <*> <*>
0cc39f34	<*> (command <*>) Error: <*> <*> <*> <*> <*> <*> <*> <ABORT code completed>
c0fcff27	member_delete (command <*>) Error: node-<*> is NOT a member of cluster node-D<*>
a84ecf37	bootGenvmunix (command <*>) Error: node returned to srm while booting: <ABORT code completed>
20bdc066	<*> (command <*>) Error: <*> <*> <*> <*> <*> <*>
b38fa951	boot (command <*>) Error: timeout while booting.
8c1777d1	boot (command <*>) Error: node went to SRM while trying to boot
dfc857c8	halt (command <*>) Error: Couldn<*>t Connect to console (state = REFUSED)
6eff2f7f	haltcluster (command <*>) Error: haltcluster: bad spawn id
03b9852a	boot (command <*>) Error: CMF-Port This node<*>s port is not connected

Table 2: Dataset Statistics

Dataset	Log Messages	Anomalies	Log Keys	Log Sequences in Test Dataset	
				Normal	Anomalous
Sys20	432,260	20,118	80	30,384	3,620
BGL	4,747,963	348,460	334	20,579	3,018

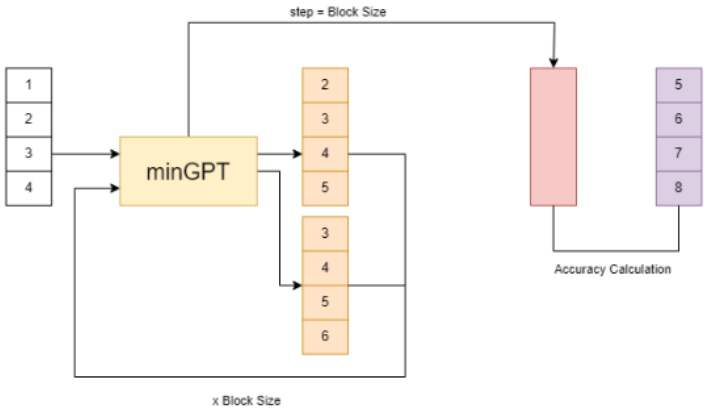

Figure 1: Overview of GPT model for anomalous log prediction.

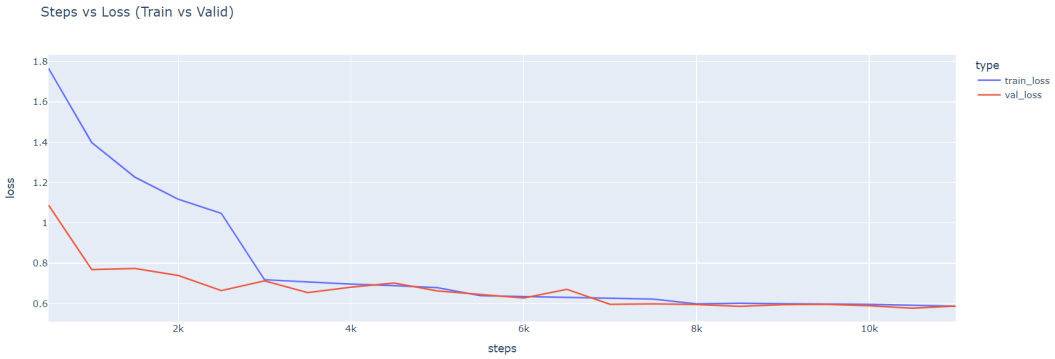

Figure 2: GPT model loss while training on HPC dataset.

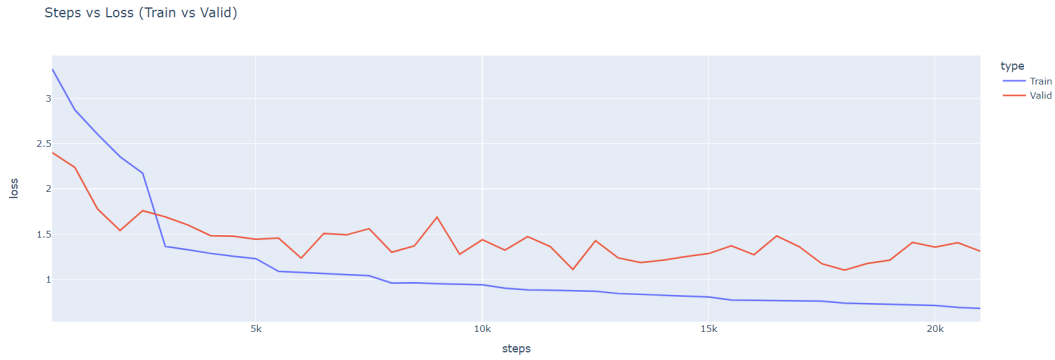

Figure 3: GPT model loss while training on BGL dataset.

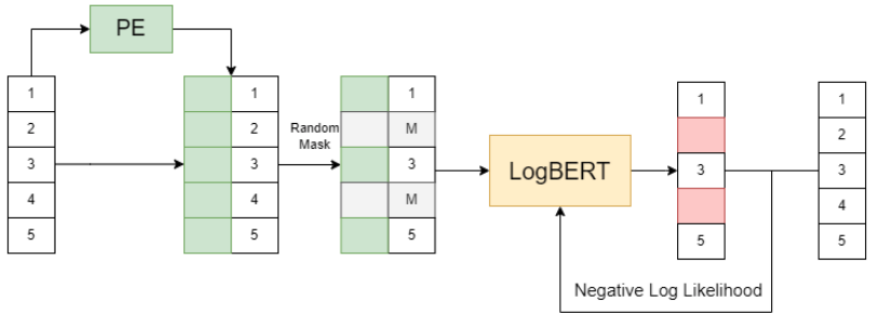

Figure 4: Overview of BERT model for anomalous log prediction.

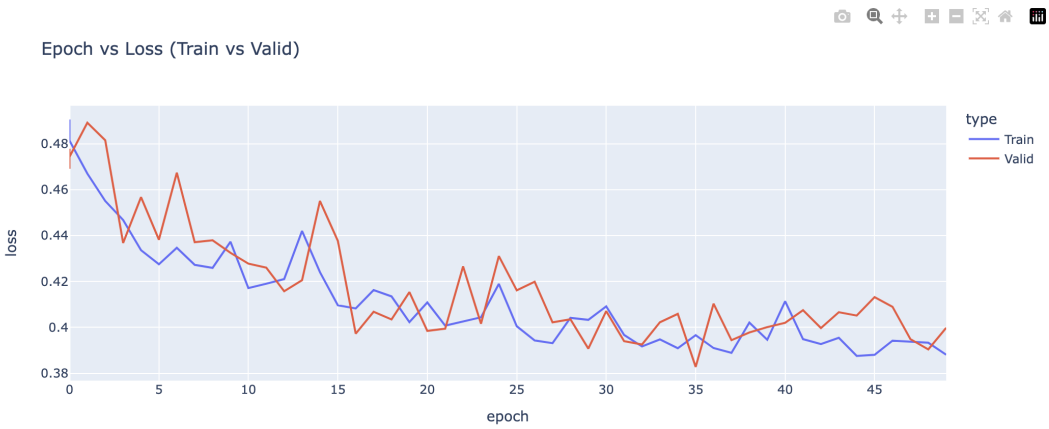

Figure 5: BERT model loss while training on System 20 dataset.

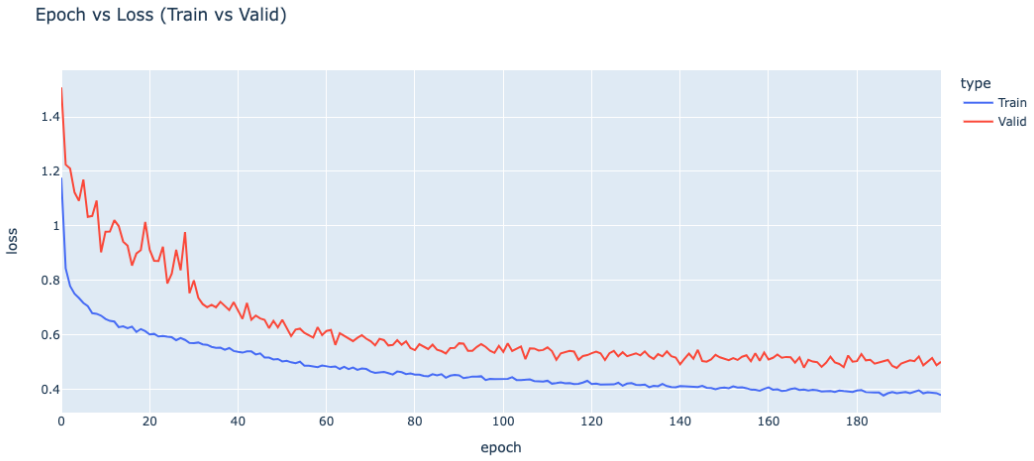

Figure 6: BERT model loss while training on BGL dataset.

$$f_{\text{att}}(h_i, s_j) = v_a^\top \tanh(W_1 h_i + W_2 s_j) \tag{1}$$

Figure 7: Formula for additive attention mechanism.

$$u_{i,t} = \tanh(W_w h_{i,t}) \tag{2}$$

$$a_{i,t} = \frac{\exp(u_{i,t}^\top u_w)}{\sum_t \exp(u_{i,t}^\top u_w)} \tag{3}$$

$$s_{i,t} = \sum_t a_{i,t} h_{i,t} \tag{4}$$

Figure 8: Word-level attention in HAN.

$$u_{i,t} = \tanh(W_s h_{i,t}) \tag{5}$$

$$a_{i,t} = \frac{\exp(u_{i,t}^\top u_s)}{\sum_t \exp(u_{i,t}^\top u_s)} \tag{6}$$

$$s_{i,t} = \sum_t a_{i,t} h_{i,t} \tag{7}$$

Figure 9: Sentence-level attention in HAN.

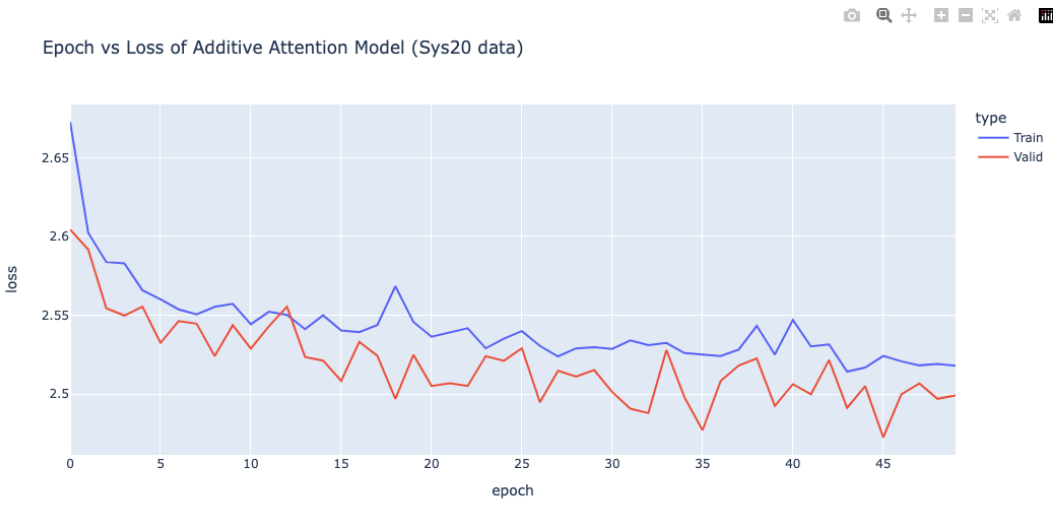

Figure 10: Additive Attention BERT model loss while training on System 20 dataset.

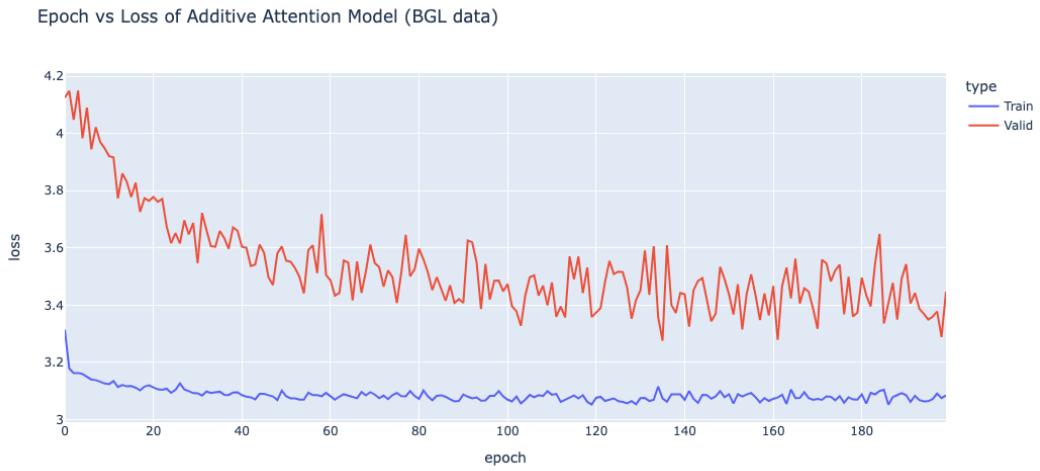

Figure 11: Additive Attention BERT model loss while training on BGL dataset.

Table 3: Performance of each model on anomalous log sequence prediction task.

	Sys20				BGL			
	Accuracy	Precision	Recall	F-1 score	Accuracy	Precision	Recall	F-1 score
GPT	45.62	98.28	9.86	17.92	83.65	80.61	36.41	50.16
GPT + PI	48.65	95.28	19.90	32.93	83.75	75.28	33.33	46.21
GPT + AA	45.62	100	9.84	17.92	-	-	-	-
BERT	80.12	68.97	1.54	3.02	**95.61**	**94.61**	**83.57**	**88.75**
BERT + AA	78.14	43.81	**31.89**	**36.91**	88.28	72.34	70.27	71.29
BERT + HA	**81.61**	**82.53**	10.47	18.59	93.17	86.30	79.69	82.86