Technical Report: R&D Winter 2024

Marco Ottavio Podesta Vezzali, Felíx, F. de J., Hérnandez G., A.,
Nieto, N., Morales, B., Gómez B., J. A., L. A. Munoz

School of Engineering and Sciences
Tec de Monterrey, Monterrey, N.L., México
marcoopodesta@gmail.com, ffelix3ro@gmail.com,
alexhergomz@gmail.com, nezihng@gmail.com,
a01661598@tec.mx, a01736171@tec.mx, amunoz@tec.mx

December 1, 2024

Abstract

In this paper, we introduce XAE^2, a double autoencoder architecture designed for text obfuscation and deobfuscation. The model consists of two autoencoders: the Obfuscator, which converts plaintext into an encrypted latent representation, and the Deobfuscator, which reconstructs the original text from this encoded form. A novel Intermediate Teacher forcing mechanism is applied during training to enhance deobfuscation accuracy, aligning the training process with real-world inference conditions. Our results demonstrate the efficacy of this approach, achieving a deobfuscation accuracy of up to 98%.

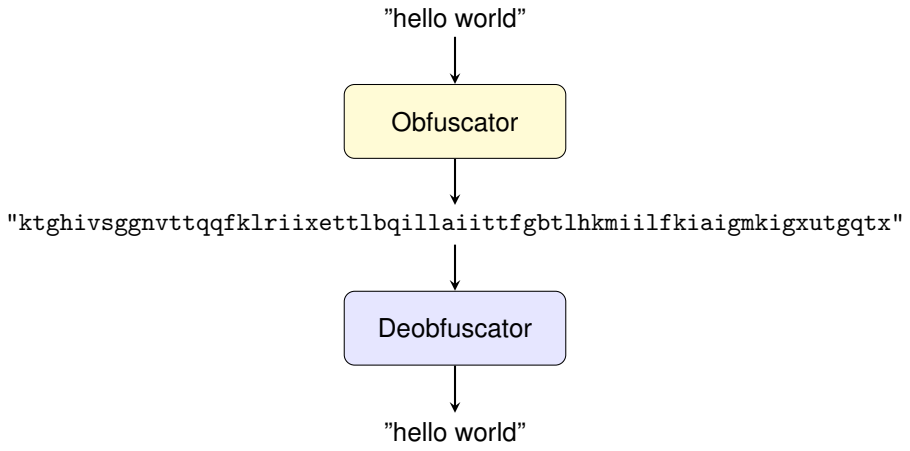

Figure 1: Example of model usage. The input text is obfuscated, then reconstructed.

0.1 Introduction

In today's digital age, data security and privacy are critical concerns [4]. Obfuscation plays an essential role in safeguarding sensitive information by transforming plaintext into unreadable formats [5]. Advancements in deep learning and neural networks can potentially encode and decode information without the need for traditional algorithms [6].

This work introduces a double autoencoder [8] architecture designed to perform text obfuscation and deobfuscation of text. The first autoencoder (Obfuscator) transforms text into obfuscated discrete latent representations, while the second autoencoder (Deobfuscator) reverses the process, recovering the original text. By combining autoencoder-based obfuscation with a twist on the Teacher Forcing [2] mechanism, we achieved high accuracy for real-world inference.

Our contributions are twofold:

- **Expansive Autoencoders**: We present a double autoencoder architecture for text obfuscation and deobfuscation with high accuracy.
- **Discrete Intermediate Teacher Forcing**: A method to enhance model confidence by feeding discrete one-hot representations during training.

0.2 Hypothesis

We hypothesize that our double autoencoder architecture can Obfuscate and Deobfuscate text accurately. The design transforms text into discrete latent representations through the first autoencoder (Obfuscator) and reverses it with the second (Deobfuscator).

0.3 State of the Art

0.3.1 Autoencoders

Autoencoders [1] are neural networks often used for data compression and feature extraction. They consist of an **encoder** $\mathrm{E} : X \to Z$ that maps input data X to a latent representation Z, and a **decoder** $\mathrm{D} : Z \to \hat{X}$ that reconstructs an approximation of a desired output, in this case $\hat{X}$ approximates the original input X. Their ability to transform input data into latent representations that are compact and non-interpretable makes them appealing for secure encoding and obfuscation methods.

$$X \xrightarrow{\mathrm{E}} Z \xrightarrow{\mathrm{D}} \hat{X}$$

0.3.2 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) [7] are neural architectures designed for sequential data processing by maintaining a hidden state h_t that evolves over time. This state depends on the current input x_t and the previous hidden state h_{t-1}, allowing RNNs to model temporal dependencies. The hidden state update is typically defined as $h_t = f(W_h h_{t-1} + W_x x_t + b)$, where W_h and W_x are weight matrices, b is a bias term, and f is a non-linear activation function.

Despite their ability to capture context from sequential data, traditional RNNs struggle with vanishing and exploding gradient issues, limiting their capacity to learn long-term dependencies. In our obfuscation-deobfuscation framework, RNNs serve as a base for encoding and decoding sequential patterns securely. To overcome their limitations with longer sequences, we explore Gated Recurrent Units (GRUs), which improve performance and stability through enhanced mechanisms for managing hidden states.

0.3.3 Gated Recurrent Unit (GRU)

The Gated Recurrent Unit (GRU) [3] is a variant of Recurrent Neural Networks (RNNs) designed to address the vanishing gradient problem and improve the efficiency of sequential data processing. GRUs achieve this by introducing two gating mechanisms: the *update gate* and the *reset gate*.

The *update gate* determines how much of the past information is carried forward to the current step, while the *reset gate* decides how much of the previous hidden state should be forgotten.

$$\begin{aligned} z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\ r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\ \tilde{h}_t &= \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \\ h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \end{aligned}$$

Here, z_t is the update gate, r_t is the reset gate, $\tilde{h}_t$ is the candidate hidden state, and h_t is the final hidden state at time t. The sigmoid activation σ and element-wise product $\odot$ ensure smooth gating operations.

0.3.4 Teacher Forcing

Teacher forcing is a training strategy often utilized in recurrent neural networks (RNNs) where the model's previous output is replaced by the actual target output from the training dataset for the next step prediction [2]. This approach stabilizes training and accelerates convergence by avoiding error accumulation across time steps.

0.4 Materials and Methods

0.4.1 Double Autoencoders

Double autoencoders, an extension of traditional autoencoders, consist of two sequentially or jointly trained autoencoder architectures. The primary goal of a double autoencoder is to enhance data representation capabilities or introduce additional transformations for specific tasks, such as obfuscation.

A double autoencoder operates by using the output generated by the first autoencoder as the input to a second autoencoder, reconstructing an approximation of a desired output. Formally, the first autoencoder encodes and decodes as $E_1 : X \to Z_1$ and $D_1 : Z_1 \to Y$, while the second autoencoder maps $E_2 : Y \to Z_2$ and $D_2 : Z_2 \to \hat{X}$, attempting to reconstruct the original input X.

$$X \xrightarrow{\mathrm{E_1}} Z_1 \xrightarrow{\mathrm{D_1}} Y \xrightarrow{\mathrm{E_2}} Z_2 \xrightarrow{\mathrm{D_2}} \hat{X}$$

0.4.2 Proposed Model Architecture

The proposed architecture consists of a double autoencoder that is divided into two main components: the **Obfuscator** and the **Deobfuscator**. The input text X is one-hot encoded, embedded, and processed through the first autoencoder (Obfuscator). The intermediate representation Y, also refered to as the obfuscated text, is a discrete, one-hot encoded format resulting from the obfuscation process. This representation is embedded, and processed through the second autoencoder (Deobfuscator) producing $\hat{X}$, the reconstruction of the original input text.

It is important to note that using a softmax activation function to produce Y resulted in poorer performance, with higher reconstruction error and a less favorable loss curve. Consequently, we opted to apply the argmax operation directly to the raw logits without normalization via softmax to improve the model's performance in terms of reconstruction accuracy and convergence.

The process begins as the **Obfuscator** takes the input text $X \in \mathbb{R}^{S_X \times V_X}$, which is represented as a one-hot encoded matrix where S_X is the sequence length, and V_X is the vocabulary size. This one-hot encoded input is passed through an embedding layer, producing an embedded representation $X_{\text{emb}} \in \mathbb{R}^{S_X \times d}$, where each token is mapped to a continuous vector of dimension d.

The embedded input X_{emb} is then processed by a GRU-based encoder, which outputs a latent representation $Z_1 \in \mathbb{R}^{S_X \times h}$, where h is the hidden state dimension of the GRU. This latent representation Z_1 is passed to a GRU-based decoder, generating an intermediate continuous representation $Y_{\text{pre}} \in \mathbb{R}^{S_X \times h}$.

The decoder output Y_{pre} is then processed through two fully connected layers to adjust its size to the desired intermediate text representation's vocabulary and length. The first fully connected layer maps Y_{pre} to the intermediate vocabulary size, resulting in $Y_{\text{voc}} \in \mathbb{R}^{S_X \times V_Y}$, where V_Y is the intermediate vocabulary size. The second fully connected layer adjusts the sequence length, producing $Y_{\text{len}} \in \mathbb{R}^{S_Y \times V_Y}$, where S_Y is the intermediate sequence length.

Finally, the intermediate vocabulary logits Y_{len} are discretized by applying the argmax operation along the vocabulary dimension, producing the obfuscated text $Y \in \mathbb{R}^{S_Y \times V_Y}$. This results in a one-hot encoded, discrete representation suitable for further processing as text.

The **Deobfuscator** repeats a similar process as the Obfuscator: It takes the obfuscated text Y as input and passes it through an embedding layer, producing an embedded representation $Y_{\text{emb}} \in \mathbb{R}^{S_Y \times d}$.

Y_{emb} is then processed through a GRU-based encoder, resulting in a latent representation $Z_2 \in \mathbb{R}^{S_Y \times h}$. The latent representation Z_2 is then passed to a GRU-based decoder, producing $\hat{X}_{\text{pre}} \in \mathbb{R}^{S_Y \times h}$.

The decoder output $\hat{X}_{\text{pre}}$ is processed through two fully connected layers. The first fully connected layer maps $\hat{X}_{\text{pre}}$ to the original vocabulary size, resulting in $\hat{X}_{\text{voc}} \in \mathbb{R}^{S_Y \times V_X}$. The second fully connected layer adjusts the sequence length to match the original input, producing $\hat{X}_{\text{len}} \in \mathbb{R}^{S_X \times V_X}$.

Finally, the raw logits $\hat{X}_{\text{len}}$ are discretized using the argmax operation along the vocabulary dimension, producing $\hat{X} \in \mathbb{R}^{S_X \times V_X}$, which is a one-hot encoded reconstruction of the original input text X.

0.4.3 Hyperparameter Selection and Design Choices

The model's embedding dimension d, hidden dimension h, and sequence lengths were chosen through initial experimentation to balance obfuscation accuracy with computational efficiency.

The input sequence length S_X was set to 16 tokens. The vocabulary size of the input text V_X is 26, corresponding to the alphabet size for the input character-based representation. The intermediate sequence length S_Y was set to 128 tokens, and the vocabulary size V_Y was set to 26.

The embedding dimension d, which maps one-hot encoded tokens to continuous vectors, and the hidden state dimension h were set to 64 to capture meaningful relationships between tokens while keeping the embedding size and

latent space and lightweight.

The GRU (Gated Recurrent Unit) was chosen for the encoder and decoder due to its efficiency and ability to handle sequential dependencies effectively. Two fully connected (FC) layers were used in the obfuscation and deobfuscation processes. The first FC layer maps the GRU decoder's output to the desired vocabulary size (V_Y for the obfuscator and V_X for the deobfuscator). The second FC layer adjusts the sequence length as needed (S_Y for the obfuscator and S_X for the deobfuscator). These layers provide the necessary transformations to align the outputs with the required format at each stage.

The final discretization step applies an argmax operation to the raw logits of both the Obfuscator and Deobfuscator to produce a one-hot encoded representation.

0.4.4 Training

As depicted in Figure 3, the training begins with the initialization phase: The Obfuscator and Deobfuscator are instantiated with the predefined architecture and hyperparameters, the training and test datasets are preprocessed and loaded into their respective data loaders, and an Adam optimizer is initialized for both the Obfuscator and Deobfuscator.

The input text X, represented as a one-hot encoded sequence, is processed through the Obfuscator and processed in batches of 32 sequences.

Once Y_{raw} is obtained, a uniformly random value sampled from $[0, 1)$ is evaluated against a threshold p. If the value is less than p, an `argmax` is applied to Y_{raw} to extract one-hot encoded representations, obtaining Y. Otherwise, Y_{raw} is passed directly to the Deobfuscator. This process, referred to as the **Intermediate Teacher Forcing** (ITF) mechanism, is designed to improve model performance, as continuous values often yielded suboptimal results during inference, where only discrete values are provided to the Deobfuscator.

The reconstruction error between the input text X and its reconstructed output $\hat{X}$ is computed using Cross-Entropy Loss. This loss is then backpropagated through the Obfuscator and Deobfuscator networks to update their respective weights.

The above steps are iteratively repeated over multiple epochs. After each epoch, the model evaluates the reconstruction accuracy and intermediate representations using validation metrics. The accuracy is computed as the proportion of correctly reconstructed tokens in $\hat{X}$. The training continues until the predefined number of epochs is completed or a satisfactory convergence in loss is observed.

0.4.5 Intermediate Teacher Forcing

The **Intermediate Teacher Forcing** (ITF) mechanism is a training strategy employed to improve model performance, particularly in scenarios where discrete representations are required for inference. This method dynamically

Algorithm 1 Training Process for Sequential Autoencoders with Intermediate Teacher Forcing

Data: X: Input text in one-hot encoding
Models: $Obfuscator$, $Deobfuscator$
Hyperparameters: Embedding dimension d, hidden dimension h, teacher forcing probability p
for each batch of input sequences X **do**
  $Y_{\text{len}} \leftarrow Obfuscator(X)$
  Intermediate Teacher Forcing
  Sample $r \sim \text{Uniform}(0, 1)$
  for each token in Y_{len} **do**
    if $r < p$ **then**
      $Y \leftarrow \text{argmax}(Y_{\text{len}})$ // Convert to one-hot encoding
    else
      $Y \leftarrow Y_{\text{len}}$ // Pass as continuous raw logits
    end if
  end for
  $\hat{X} \leftarrow Deobfuscator(Y)$
  Compute Loss and Update Model
  Calculate reconstruction loss between X and $\hat{X}$ using cross-entropy
  $\theta_{E_1,D_1,E_2,D_2} \leftarrow \text{Update}(\theta_{E_1,D_1,E_2,D_2} - \nabla \mathcal{L}_{\text{reconstruction}})$ // Update model parameters with gradient descent
end for
Inference Phase: Disable teacher forcing; always use the argmaxxed one-hot discrete representations of Y

alternates between using discrete and continuous data representations during training, controlled by a probability p.

As seen in Figure 4, during the training phase, the continuous logits Y_{raw}, produced by the Obfuscator, are processed as follows: A random probability is sampled from a uniform distribution $[0, 1)$. If this value is less than the threshold p, Y_{raw} undergoes an `argmax` operation to yield a one-hot encoded, discrete representation Y. Otherwise, the continuous logits Y_{raw} are passed directly to the Deobfuscator.

This probabilistic switching aims to balance two competing objectives. First, the **Preservation of Gradient Flow** as Continuous logits Y_{raw} allow gradients to flow uninterrupted through the network during backpropagation, enabling efficient parameter updates for both the Obfuscator and Deobfuscator. Discrete representations, being non-differentiable, break this gradient flow and hinder effective training. And **Preparation for Inference**, since during real-world scenarios, the Deobfuscator processes discrete representations exclusively, training the model with discrete inputs periodically ensures that the Deobfuscator learns to handle such data effectively, bridging the gap between training and inference.

Without ITF, training the model entirely with continuous representations Y_{raw} leads to a mismatch between training and inference conditions, which results in suboptimal performance. Conversely, training solely with discrete representations causes gradient disconnection, stalling learning.

0.5 Results

The performance of XAE^2, was evaluated using discrete and continuous data being fed to the Deobfuscator. We performed 7 runs of 15 epochs with $30,000$ sequences of text during training. The results provide insights into how the architecture responds to different configurations of the ITF probability p, as well as its behavior when the Deobfuscator is exposed to continuous versus discrete evaluation inputs, simulating real world inference applications.

The values tested for p were 0.0, 0.005, 0.01, 0.02, 0.05, 0.1, 0.5, 0.9, and 1.0. We found that the model functioned significantly better with probabilities between 0 and 1, specifically the closer they got to 1 seemed to handle discrete data inputs for the Deobfuscator better, but as soon as it reaches 1, the results declined.

The loss graphs (Figure 5 and Figure 6) provide additional context for the training dynamics. The variations in loss curves among different p values suggest differences in overfitting tendencies. A lower p (e.g., $p = 0.0$ and $p = 0.005$) led to better loss minimization during training but compromised inference accuracy when dealing with discrete data. Conversely, higher p values resulted in slightly elevated training losses but better generalized performance during inference, but once it reached $p = 1.0$ we can see its performance worsening during the training. The differences in loss are likely due to the model overfitting to learn that what goes in, must come out. It is effectively learning to map input to

output.

As depicted in Figures 7 and 8, when continuous data was passed to the Deobfuscator, the model achieved consistently high accuracy, as long as p was below 1. This demonstrates that the architecture is inherently capable of reconstructing the input text effectively when the Deobfuscator is provided with high-dimensional, differentiable latent representations. But once the data we passed to the Deobfuscator was always discrete, it broke, providing declining accuracy when dealing with continuous data, presumably because it was never trained on data that was not a one-hot representation.

Switching to discrete data inputs, a stark contrast in performance was noted depending on whether ITF was employed during training. Without ITF, the model's accuracy dropped significantly to an average of 63%. This substantial decline is attributed to the mismatch between training and inference conditions, as discrete data was not adequately represented in the training process. Introducing ITF improved accuracy across all probabilities, bridging the gap between training and inference by enabling the model to generalize better to discrete representations.

Figures 9 and 10 illustrate these results. The average accuracy trends upwards as ITF probability p is more than 0.0, showcasing the importance of ITF in preparing the model for real-world inference, where discrete data inputs for the Deobfuscator are the norm.

0.6 Applicability and Extensions

The XAE^2 architecture directly addresses text obfuscation and deobfuscation, but its design is modular and generalizable, making it highly adaptable to a wide range of applications. This flexibility allows its application in numerous domains while also presenting opportunities for further research and implementation.

One major area of applicability is **secure text-based communication**. The XAE^2 framework can be used to encrypt sensitive messages in chat applications or obfuscate personally identifiable information (PII) in emails and documents. By providing a neural alternative to traditional cryptographic techniques, it is especially suited for scenarios requiring adaptive or lightweight solutions. Additionally, in environments with strict content moderation or censorship, XAE^2 can transform sensitive text into encoded representations that are less likely to trigger filtering mechanisms. This enables freer communication while ensuring that the original content can still be recovered.

Another promising domain involves secure data transmission. XAE^2 can act as a neural substitute for traditional encryption techniques, offering a keyless and adaptive alternative for secure data sharing. For resource-constrained devices in the Internet of Things (IoT) ecosystem, lightweight versions of XAE^2 could be used to obfuscate sensitive telemetry data before transmission, with a centralized server deobfuscating this data for analysis, thus ensuring privacy across the network. Similarly, in wireless communication systems, XAE^2 can obfuscate transmitted messages, reducing the risk of interception and unautho-

rized access. This application is particularly relevant for low-latency systems, such as those used in autonomous vehicle networks.

The architecture also has potential in handling biomedical data, addressing privacy concerns in this highly regulated field. For example, hospitals and healthcare providers could implement XAE^2 to obfuscate patient records before storage or transmission to third-party researchers. This would ensure compliance with privacy regulations such as HIPAA and GDPR while safeguarding sensitive information. Furthermore, the flexibility of XAE^2 makes it an attractive choice for developing privacy-preserving solutions tailored to specific biomedical and healthcare use cases.

0.7 Future Research

Future research for the XAE^2 architecture highlights several promising directions to enhance its capabilities and broaden its applications. One significant opportunity involves extending its design beyond text data to handle multimodal inputs, such as images, audio, and video. This would require the development of advanced encoder-decoder mechanisms capable of processing and obfuscating multimodal data while ensuring compatibility of latent representations across different modalities.

Another key direction is the implementation of personal private keys or customizable model configurations to enable user-specific obfuscation. By introducing unique keys or allowing users to adjust the model parameters, the results of obfuscation could be tailored to individual users. This customization would ensure that obfuscation outcomes are different for each user, adding an additional layer of security and privacy. Such an approach could also facilitate the development of personalized security protocols, making the architecture more adaptable to diverse requirements.

The expansion of evaluation metrics is another area of potential development. While current assessments focus on accuracy and loss, future efforts could develop more comprehensive evaluation frameworks. These frameworks would integrate metrics such as privacy guarantees, robustness under adversarial conditions, computational efficiency, and usability. Such enhancements would offer a more holistic perspective on the model's performance and its practical applicability in real-world scenarios.

Lastly, advancing the theoretical understanding of the Intermediate Teacher Forcing (ITF) mechanism could provide insights into its broader utility. Investigating optimal probability parameters for ITF across diverse datasets, tasks, and architectures may generalize its applicability beyond XAE^2, contributing to the broader field of neural network research.

By exploring these directions, including multimodal processing, customizable security features, advanced evaluation metrics, and theoretical advancements, XAE^2 has the potential to become a robust and versatile framework for secure, privacy-preserving data processing across a wide range of domains.

0.8 Discussion

In this work we introduced XAE^2, a novel double autoencoder architecture designed for text obfuscation and deobfuscation. By leveraging the capabilities of autoencoders, particularly GRU-based encoders and decoders, the model achieved a high deobfuscation accuracy of up to 98%, demonstrating its effectiveness in transforming plaintext into secure latent representations and reconstructing it with minimal loss. The proposed Intermediate Teacher Forcing (ITF) mechanism proved instrumental in bridging the gap between training and real-world inference by alternating between continuous and discrete representations, ensuring robust performance even under discrete inference conditions.

The modular design of XAE^2 makes it highly adaptable, with immediate applications in secure communication, privacy-preserving analytics, and encrypted data transmission. Furthermore, its potential extends beyond text processing to other modalities, such as image, audio, and video, paving the way for multimedia obfuscation and neural steganography. The architecture's adaptability also enables use cases in resource-constrained environments, such as IoT networks, where lightweight and efficient obfuscation methods are essential.

The evaluation of XAE^2 highlighted the critical role of ITF in enhancing model accuracy when discrete obfuscated data is passed to the Deobfuscator. Experiments revealed that higher ITF probabilities improved the model's ability to generalize to real-world conditions, reducing the performance disparity between training and inference.

Future research directions include extending XAE^2 to multimedia data obfuscation, optimizing its design for single-autoencoder implementations, and integrating it with existing security protocols. Additionally, the development of adaptive obfuscation strategies and privacy-preserving federated learning setups offers promising avenues for further exploration.

Acknowledgment

I want to say thank you to everyone who helped me on this journey. Thank you, Tec de Monterrey, for providing a platform for research. Thank you to my advisor, Alberto Muñoz Ubando, for providing guidance when I was lost and needed direction. Thank you, friends, it's been a wonderful semester. Thank you to the library staff for keeping the library so nice. Thank you, Manuel and Mojama, for calling me every once in a while to catch up. Thank you, Leo, for coming to Monterrey. Thank you to a special friend who showed me kindness in moments when I needed it most; where have you been, loca? Thank you, Mamma e Papà, for giving me the chance to live such an amazing experience.

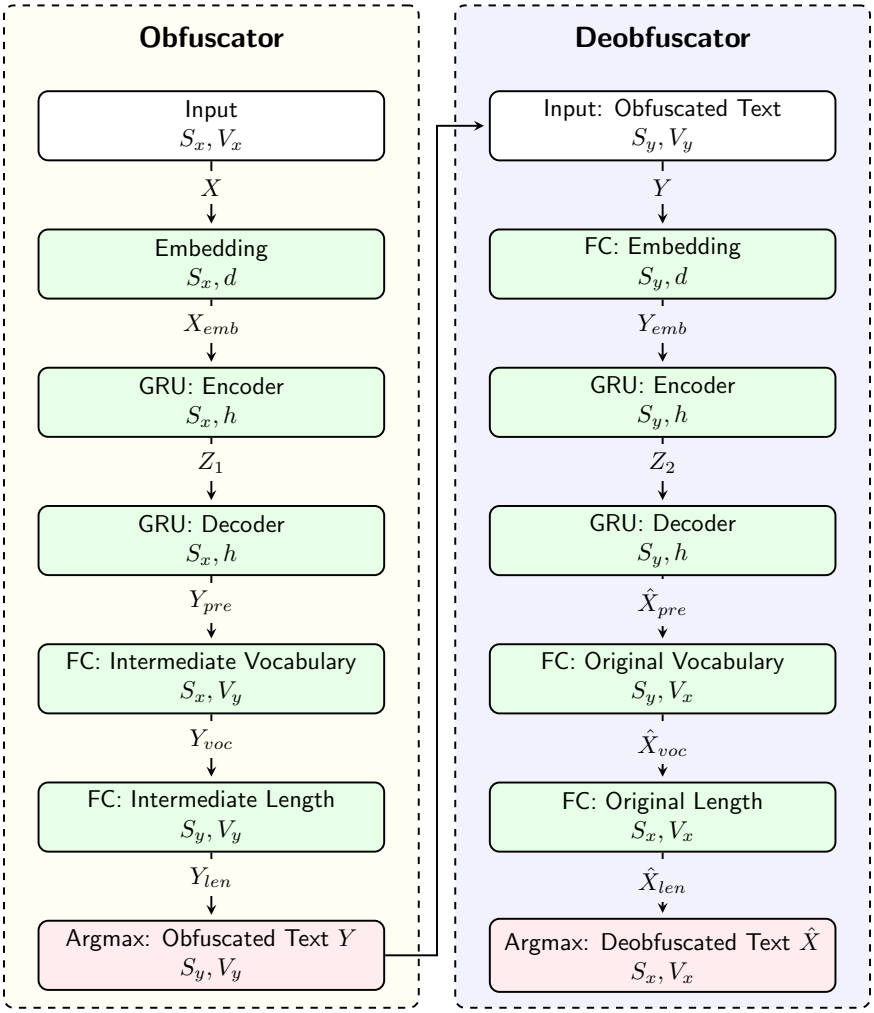

Figure 2: Model Diagram

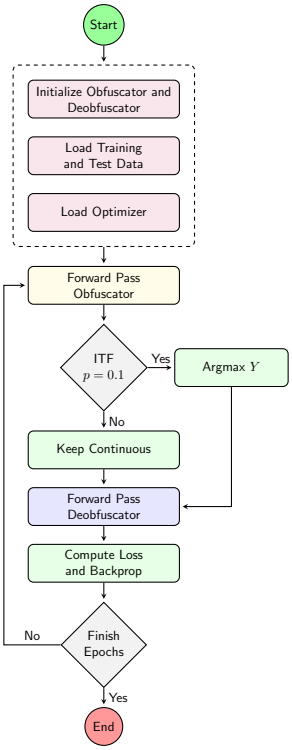

Figure 3: Training representation. Focus on the Intermediate Teacher Forcing.

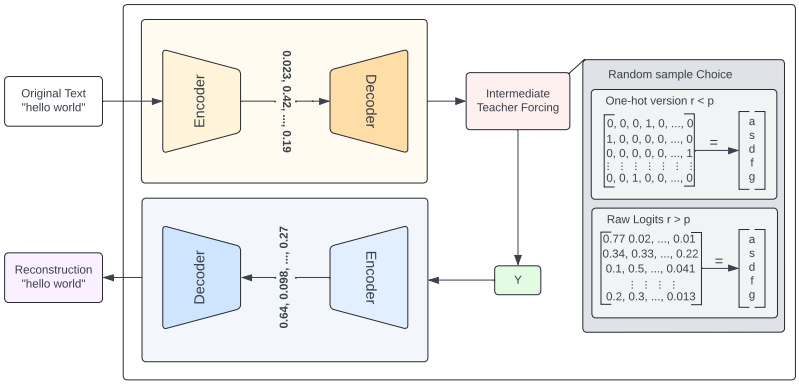

Figure 4: Intermediate Teacher Forcing representation, notice how if the sample obtained is less than p, the values are discretized into a one-hot representation instead of continuous data.

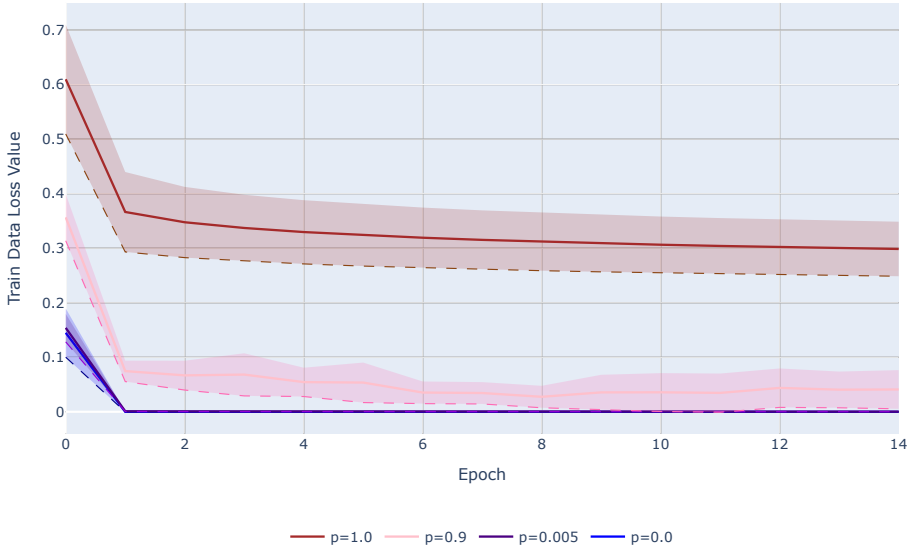

Figure 5: Loss with examples for p in some extreme values. We can see there is almost no difference in loss between 0.0 and 0.005, they both overfit, while $p = 1.0$ is not learning correctly.

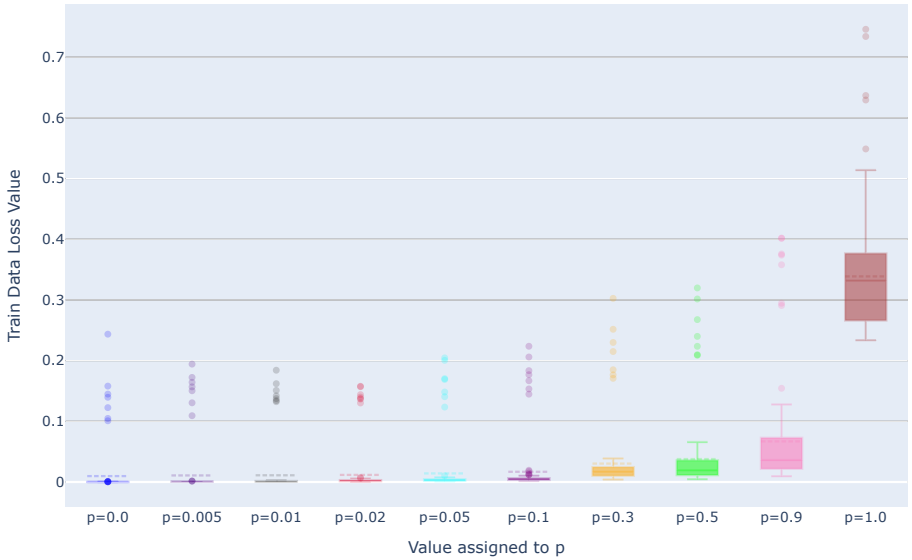

Figure 6: Average loss for all testing values of p at each run. All testing values for p. Notice how the closer p gets to 1, the worse its loss gets

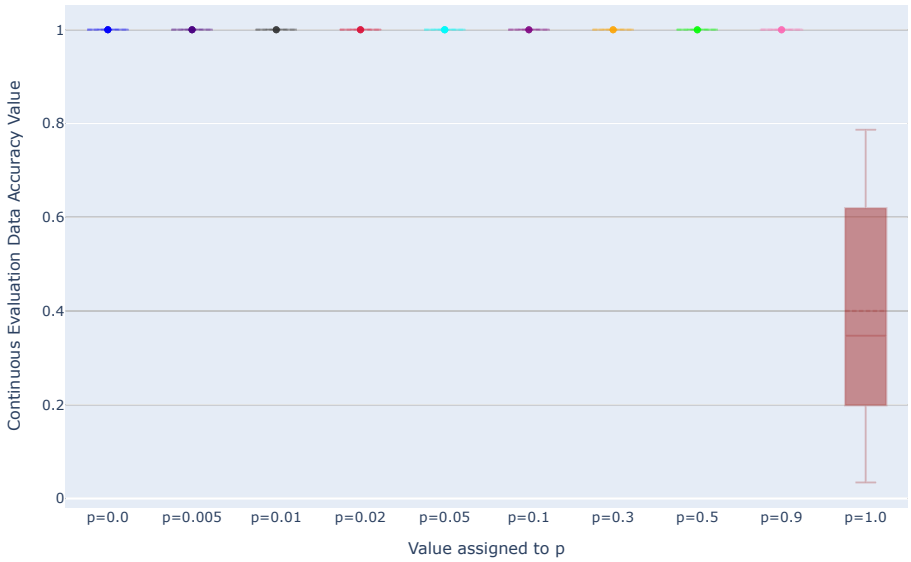

Figure 7: Model performance with continuous evaluation data being passed to the Deobfuscator from the Obfuscator during training, regardless if ITF was activated.

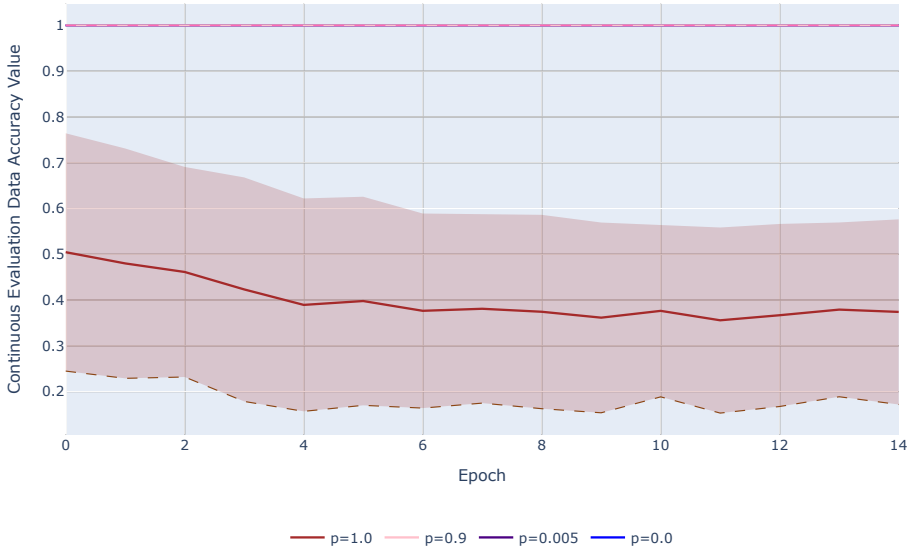

Figure 8: Model performance with continuous evaluation data being passed to the Deobfuscator from the Obfuscator during training, regardless if ITF was activated.

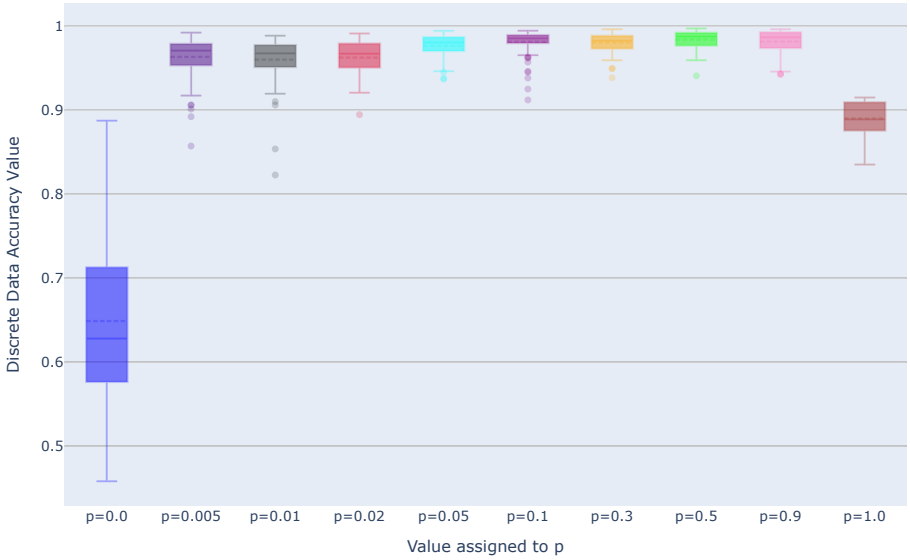

Figure 9: Model performance with the discrete argmaxxed one-hot values from the Obfuscator being passed to the Deobfuscator. Pay attention to the decline in accuracy once p reaches 1

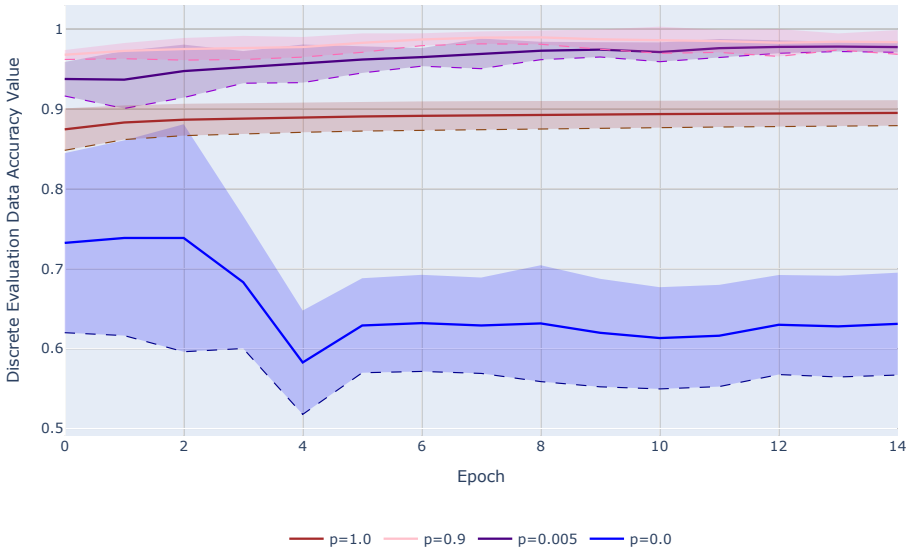

Figure 10: Model performance with the discrete argmaxxed one-hot values from the Obfuscator being passed to the Deobfuscator. Representative values for p were chosen.

Bibliography

[1] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *CoRR*, abs/2003.05991, 2020.

[2] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[4] Widia Febriyani, Tien Fabrianti Kusumasari, and Muharman Lubis. Data security: A systematic literature review and critical analysis. In *2023 International Conference on Advancement in Data Science, E-learning and Information System (ICADEIS)*, pages 1–6, 2023.

[5] Mandy Knöchel and Sebastian Karius. Text steganography methods and their influence in malware: A comprehensive overview and evaluation. 2024.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[7] Robin M. Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv preprint arXiv:1912.05911*, 2019.

[8] Ha Son Vu, Daisuke Ueta, Kiyoshi Hashimoto, Kazuki Maeno, Sugiri Pranata, and Sheng Mei Shen. Anomaly detection with adversarial dual autoencoders. *arXiv preprint arXiv:1902.06924*, 2019.