

Challenges in Stabilizing Training for Diachronic Text Style Transfer

Adrian Niedziółka-Domański
0009-0003-4797-7484
Maria Curie-Skłodowska University
Plac Marii Curie-Skłodowskiej 5
20-031 Lublin, Poland
Email: niedziolkadrian@gmail.com

Abstract—This paper analyzes the challenges associated with training an architecture inspired by CycleGAN for diachronic text style transfer between Early Modern English (EMO) and modern English of the XXI century. Unlike the classic CycleGAN, a single Transformer generator conditioned by the FiLM mechanism for the target language period was used, which reduces the number of parameters but significantly complicates the learning process.

The main training challenges include the instability of adversarial learning, error accumulation in autoregressive generation, loss of semantics, token repetition, and high memory requirements. This paper proposes solutions that effectively address these issues through a series of mechanisms, such as the use of Gumbel-Softmax, adaptive penalty for token repetition, adaptive control of the discriminator training, or two-stage approach to autoregressive training.

Thanks to these mechanisms, it was possible to increase training stability and improve the quality of the generated sequences.

I. INTRODUCTION

TEXT style transfer is the task of transforming an input text into a different linguistic style while preserving its original semantic meaning. Unlike traditional machine translation, the goal is not to change the language, but to modify the text's stylistic features, such as vocabulary, spelling, syntax, or form of expression.

In the context of historical variants of the language, this issue becomes particularly challenging. Older forms of English (e.g., Early Modern English, EMO) differ significantly from modern English in terms of spelling, the use of special characters, sentence structure, and characteristic lexical constructions. At the same time, preserving the original meaning of the text during style conversion remains a key requirement of a generative system.

This paper analyzes the challenge of style transfer between texts in Early Modern English and present-day English used in the XXI century.

A. Challenges in Training CycleGAN Architectures for Text

The presented approach uses an architecture inspired by CycleGAN [1], adapted for the task of historical style transfer. Unlike the classic CycleGAN architecture, a single generator is used to perform bidirectional transfer by conditioning the model on the target language epoch. This solution reduces the

number of model parameters but also complicates the training process, as the generator must simultaneously:

- preserve the semantics of the text,
- generate features characteristic for the target style,
- enable reconstruction of the original sentence within the framework of cycle consistency.

Training such an architecture involves a number of problems characteristic for both GAN [2] models and autoregressive language models.

One of the main challenges is the instability of adversarial learning. The discriminator can very quickly learn to distinguish between real and generated samples, especially in the early stages of training when the generator is still producing low-quality text. As a result, the generator stops receiving a useful gradient signal, which causes learning progress to stop.

An additional difficulty is the autoregressive nature of text generation. Each subsequent token depends on the previously generated content, causing errors to accumulate during sequence generation. Combined with adversarial learning, this leads to problems such as:

- repetition of tokens and generator collapse,
- loss of semantics during style transfer,
- unstable oscillations of the loss function,
- over-dominance of the discriminator,
- high memory requirements resulting from the need to maintain the computational graph for autoregressive generation.

An additional challenge in CycleGAN architecture is the need to simultaneously reconcile two partially conflicting goals:

- changing the style of the text,
- and preserving its original meaning.

Too strong an emphasis on adversarial learning can lead to a loss of meaning and the generation of stylistically correct but semantically unrelated sentences. Meanwhile, excessive dominance of cycle consistency can cause the model to simply copy the input text without any actual style transfer.

B. Aim of the work

The aim of this work was to analyze and stabilize the training process of a CycleGAN-inspired architecture for the

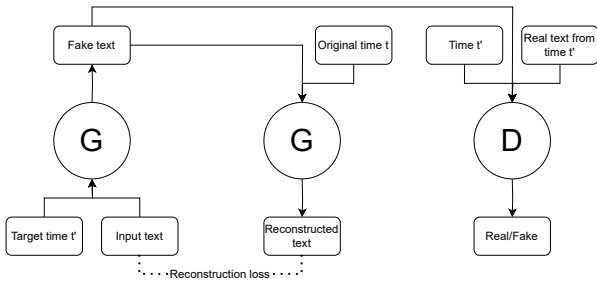


Fig. 1. General model architecture, from [5].

task of language style transfer between Early Modern English and modern English.

This work proposes an approach utilizing:

- a Transformer-based encoder–decoder generator [3],
- a discriminator based on the Transformer Encoder architecture,
- style conditioning using the FiLM mechanism [4],
- a cycle consistency mechanism enabling the preservation of text semantics,
- methods for stabilizing the training of the generator and discriminator.

Particular emphasis was placed on problems which arise during the training of adversarial language models, primarily in the context of autoregressive text generation. Among other things, the study analysed the impact of the discrete nature of text data, the instability of adversarial learning, memory issues, and the preservation of semantics during style transfer.

The goal of the proposed solutions was to increase the stability of the learning process, improve the quality of generated sequences, and mitigate common issues encountered during the training of text-based GAN/CycleGAN architectures.

II. SYSTEM ARCHITECTURE

The proposed system is based on the CycleGAN architecture, which has been specifically adapted to address the problem of text style transfer, as presented in Fig. 1 [5]. The system consists of two main components: a generator G responsible for transforming text between styles, and a discriminator D that evaluates the stylistic correctness of the generated sequences.

The generator receives input text along with information about the target language style and generates a new sequence of tokens corresponding to the selected language period. This paper considers two styles:

- Early Modern English (EMO),
- and modern English from the XXI century.

The discriminator acts as a classifier trained to distinguish real samples from generated ones. The model analyzes both the content of the sequence and the style information, allowing it to assess whether the text corresponds to the specified linguistic period.

Additionally, the cycle consistency mechanism was used during training. After generating text in the target style, the generator attempts to reconstruct the original sequence by transferring it back to the initial style. This helps minimize the loss of semantic information and reduces the risk of generating texts that are stylistically correct but semantically inconsistent with the input.

A. Generator Architecture

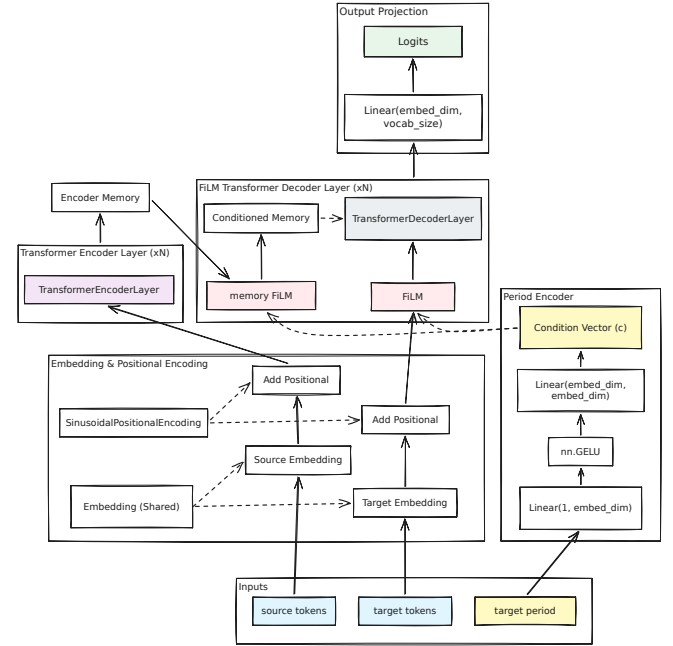


Fig. 2. Generator architecture.

The generator was implemented as an autoregressive encoder–decoder model based on the Transformer architecture. The generator’s task is to transform the input text into a style characteristic of the target linguistic era while preserving semantic information.

The model consists of:

- token embedding layer,
- Transformer encoder,
- Transformer decoder with FiLM conditioning,
- dictionary space projection layer.

The whole architecture is shown in Fig. 2.

In the experiments, 128-dimensional embeddings were used. Both the encoder and the decoder consisted of 3 layers. The full model configuration is summarized in Table I.

TABLE I
GENERATOR CONFIGURATION USED IN THE EXPERIMENTS.

Component	Value
Embedding dimension ($embed_dim$)	128
Vocabulary size ($vocab_size$)	32000
Encoder layers	3
Decoder layers	3

1) *Representation of input tokens*: Input tokens (x) were represented using an embedding layer (*Embedding*) with a dimension of $(vocab_size, embed_dim)$. In order to keep information about token order, sinusoidal positional encoding (*PE*) was added to the embeddings, following the original Transformer architecture [3].

The embeddings were also scaled by a factor of $\sqrt{embed_dim}$, in accordance with the original Transformer architecture, which stabilized the scale of the input representations after adding positional encoding.

So, the final representation is given by:

$$x_{embed} = \sqrt{embed_dim} \cdot Embedding(x) + PE(x)$$

2) *Encoder*: The encoder component was responsible for creating a semantic representation of the input text. The encoder consisted of three layers of Transformer Encoders that utilized a multi-head self-attention mechanism.

The encoder processed the entire input sequence in parallel, enabling the model to build representations that depend on the global context of the sentence.

3) *Style representation and FiLM conditioning*: Information about the target language style was represented using period encoding (embedding layer of shape $(1, embed_size)$). In the experiments, a binary representation was used to specify the target language style:

- Early Modern English - 0,
- modern English of the XXI century - 1.

The style embedding was then passed to the FiLM (Feature-wise Linear Modulation) mechanism [4], which modulated the model's latent representations.

The FiLM mechanism calculated scaling factors and shifts of the hidden representation vector depending on the target style:

$$FiLM(x) = \gamma(c) \odot x + \beta(c)$$

where:

- x denotes the hidden representation vector,
- c represents the style embedding,
- γ and β denote the modulation parameters.

4) *Decoder*: The generator's decoder was implemented as a Transformer Decoder consisting of 3 layers.

During generation, each token in the sequence could only use information from previously generated tokens. This constraint was enforced by using causal masking, which prevented access to future tokens in the sequence.

A FiLM conditioning mechanism was applied before each layer of the decoder to modify token representations according to the target language style.

5) *Projection onto the dictionary space*: The decoder's output was mapped to the vocabulary space using a linear projection layer:

$$\mathbb{R}^{embed_dim} \rightarrow \mathbb{R}^{vocab_size}$$

The resulting logits represented the probability distribution of the subsequent tokens in the generated sequence.

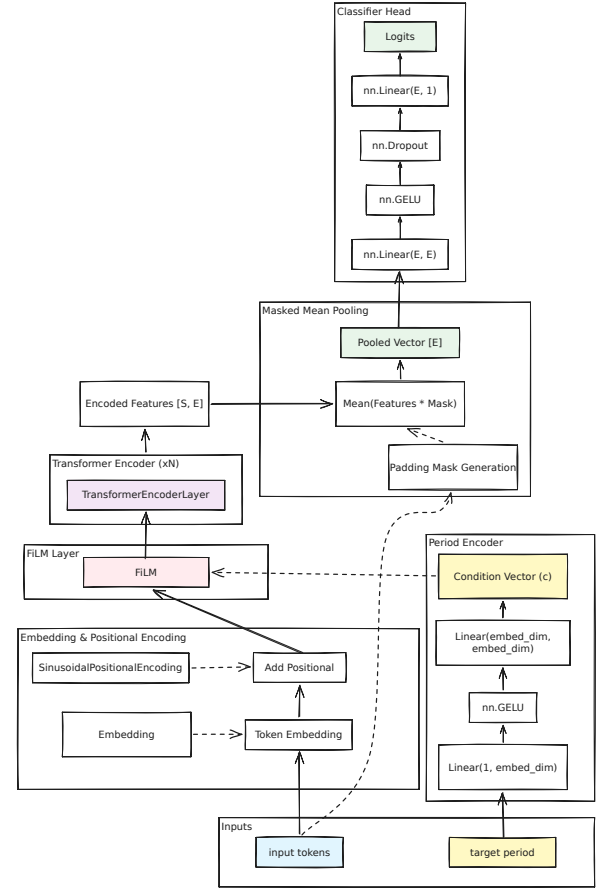


Fig. 3. Discriminator architecture.

B. Discriminator Architecture

The discriminator was implemented as a model based on the Transformer Encoder architecture, inspired by BERT-type models [6]. The discriminator's task was to distinguish real texts from sequences generated by the generator and to analyze the text's consistency with the target historical style. The model analyzed the entire sequence. This allowed for the use of the global context of the sentence during language style classification.

The discriminator's architecture consisted of:

- token embedding layer,
- period encoding mechanism and FiLM conditioning,
- Transformer encoder,
- binary classifier.

Discriminator architecture is shown in Fig. 3. In the experiments, 128-dimensional embeddings were used. The encoder consisted of 3 layers.

The representation of input tokens was implemented in the same way as in the generator. A token embedding layer with dimensions $(vocab_size, embed_dim)$ was used.

Information about historical style was represented using a period encoding mechanism, analogous to the one used in the generator. The style embedding was then used in the

FiLM (Feature-wise Linear Modulation) mechanism, which modulates token representations before feeding them to the Transformer encoder.

1) *Transformer Encoder*: The main component of the discriminator was a Transformer Encoder consisting of three layers.

The encoder-only architecture allowed the entire sequence to be analyzed simultaneously, enabling the model to leverage relationships between distant parts of the sentence during style classification.

Unlike the generator, the discriminator did not use causal masking, as the model had access to the entire sentence at once.

2) *Pooling and classification*: The encoder's output took the form of a representation of all tokens in the sequence: $(seq_len, embed_dim)$. To obtain a single representation of the sentence, mean pooling was applied, taking into account the masking of padding tokens, and the result was a vector of dimension $(embed_dim)$.

The sequence representation was calculated as the average of the representations of all unmasked tokens:

$$h = \frac{1}{N} \sum_{i=1}^N x_i$$

where x_i is a vector of size $embed_dim$ representing a token.

The resulting vector was then fed into a binary classifier consisting of:

- linear layer,
- GELU activation function,
- final linear layer generating a single logit.

C. Training loop

The model was trained iteratively in batches of 32 using the PyTorch framework [7]. In each iteration, separate steps were performed to update the parameters of the discriminator and the generator.

For each batch, the following data was collected:

- input sequences,
- information about the source time period,
- the target historical style set as the opposite time period.

The whole training loop is presented in Alg. 1.

1) *Generating Training Samples*: During training, the generator was used to create sequences styled after the target historical era. Generation was performed using an autoregressive model, with each subsequent token depending on:

- input text,
- previously generated tokens,
- target historical period.

The generation process began with the start of sequence (SOS) token and ended after the end of sequence (EOS) token was generated or the maximum sequence length was reached. In the experiments, the maximum length was set to 192 tokens.

Algorithm 1 Training loop

```

1: for each epoch do
2:   for each batch  $(x, p_{src})$  do
3:     // Train Discriminator
4:     Generate fake samples:  $\hat{x} \leftarrow G(x, p_{tgt})$ 
5:     Concatenate real and generated samples:  $X \leftarrow (x, \hat{x})$ 
6:     Compute discriminator predictions:
        $(D_{real}, D_{fake}) \leftarrow D(X)$ 
7:     Compute discriminator loss:
        $L_D \leftarrow BCE(D_{real}, 1) + BCE(D_{fake}, 0)$ 
8:     Update discriminator parameters
9:     // Train Generator
10:    Generate stylized sequence:  $\hat{x} \leftarrow G(x, p_{tgt})$ 
11:    Compute adversarial loss (generated samples as real):
        $L_{adv} \leftarrow BCE(D(\hat{x}, p_{tgt}), 1)$ 
12:    Reconstruct original sequence using teacher forcing:
        $x_{rec} \leftarrow G(\hat{x}, p_{src})$ 
13:    Compute cycle consistency loss:
        $L_{cyc} \leftarrow CE(x_{rec}, x)$ 
14:    Compute final generator loss:
        $L_G \leftarrow L_{adv} + \lambda_{cyc} L_{cyc}$ 
15:    Update generator parameters
16:   end for
17: end for

```

2) *Training Discriminator*: During the training of the discriminator, a generator was used to produce sequences styled to match the target language era. The generated samples were treated as fake data, while the original sentences were treated as real data.

The discriminator was trained using the BCE (Binary Cross Entropy) loss function, which was designed to distinguish real sequences from generated ones.

3) *Training Generator*: The generator was trained using a combination of two loss functions: adversarial loss and cycle consistency loss. The detailed training process, including discriminator updates and the training schedule, is presented in Alg. 1.

The goal of adversarial loss is to teach the generator to create sequences that are indistinguishable from real historical text samples. The generator learns to “trick” the discriminator by generating text in a given style, conditioned on the target period.

The second component, cycle consistency loss, is responsible for preserving semantic information during style transfer. To this end, the generated sequence is reprocessed by the generator to recreate the original input style. During this stage, a teacher forcing mechanism is used to accelerate training. It works by feeding the model ground-truth tokens from the target sequence as input instead of its own predictions, which stabilizes the learning process. The reconstruction error is calculated using the cross-entropy function.

The final loss function of the generator takes the form of a weighted sum of two components: adversarial loss and cycle

loss, where the first component is responsible for the realism of the generated sequences, and the second for maintaining semantic consistency between the input and the generated sequence.

$$L_G = L_{adv} + \lambda_{cyc} L_{cyc}$$

During training, the value of the λ_{cyc} coefficient was dynamically adjusted. This allowed to gradually increase the influence of cycle consistency loss during later stages of training.

III. TRAINING ISSUES AND CYCLEGAN STABILIZATION

Training CycleGAN for text generation proved to be significantly more unstable than in the case of models generating continuous data, such as images. During the experiments, a range of problems were observed related to the discrete nature of tokens, autoregressive sequence generation, as well as the dynamic balance between the generator and the discriminator.

This chapter discusses the most important issues that arise during training and introduces applied stabilisation mechanisms.

A. Problem of non-differentiability of the $\arg\max$ operation

In classical text generation, token selection is often performed using the $\arg\max$ operation, which leads to a loss of differentiability due to the discrete nature of sampling. In GAN-based architectures, this limits the ability to propagate gradients from the discriminator to the generator, as the token selection operation interrupts the continuous flow of training information (gradient).

This problem becomes even more significant in CycleGAN-type architectures for text, where an additional reconstruction loop ($x \rightarrow \hat{x} \rightarrow x_{rec}$) exists alongside the standard generation path. In this setup, the lack of differentiability can disrupt both the adversarial signal and the signal resulting from the consistency cycle, limiting gradient flow through the full style transformation cycle.

To preserve gradient propagation, the Gumbel-Softmax mechanism [8] was employed, which approximates discrete token sampling using a continuous probabilistic representation:

$$y_i = \frac{\exp\left(\frac{\log(\pi_i) + g_i}{\tau}\right)}{\sum_{j=1}^k \exp\left(\frac{\log(\pi_j) + g_j}{\tau}\right)}, \quad \text{for } i = 1, \dots, k.$$

where:

- y_i denotes the i -th component of the resulting Gumbel-Softmax distribution,
- π_i denotes the probability of selecting the i -th token,
- g_i is a random variable drawn from the Gumbel distribution,
- τ is the temperature parameter controlling the sharpness of the distribution,
- k is the number of categories (vocabulary size).

B. Discriminator Shortcut Learning

After applying Gumbel-Softmax, another problem was observed: the discriminator very quickly learned to distinguish between real samples and those generated based solely on the structure of the probabilistic representations.

Real tokens were represented as one-hot vectors, while the generator produced soft probability distributions. In practice, the discriminator did not analyze the linguistic features of the text but detected numerical artifacts characteristic for the generated data.

This problem led to:

- very rapid increase in the discriminator's accuracy,
- disappearance of useful gradients for the generator,
- destabilization of adversarial learning.

To limit shortcut learning, the following measures were taken:

- setting the `hard=True` parameter in Gumbel-Softmax,
- lowering the distribution temperature.

The `hard=True` parameter caused the representation to be projected into one-hot encoding during the forward pass while preserving gradients during the backward pass. Lowering the temperature further reduced the blurring of the probability distributions.

This solution prevented the discriminator from detecting representation artifacts instead of actual linguistic style features.

C. Token Repetition and Generator Collapse

One of the most frequently observed issues during generator training was the generation of repetitive tokens or short, looping text fragments. This problem was especially noticeable during unstable phases of adversarial training.

In extreme cases, the generator produced identical tokens at consecutive positions in the sequence, leading to a degradation in the quality of the generated text (i.e. mode collapse).

To mitigate this phenomenon, an additional loss function component was introduced in the form of a *repeat penalty*, calculated at the level of the probability distributions of generated tokens (Gumbel-Softmax), to preserve the gradient graph. For this purpose, the soft distributions of tokens for subsequent positions in the sequence p_t and p_{t+1} were compared, and their similarity was then calculated as a scalar product:

$$sim_t = p_t \cdot p_{t+1}$$

These values are calculated for the entire sequence, excluding padding tokens, by applying a mask that depends on the probability of the PAD token occurring. The result is a normalized penalty value (between 0 and 1):

$$L_{rep} = \frac{\sum_{t=1}^{T-1} sim_t \cdot m_t}{\sum_{t=1}^{T-1} m_t + \epsilon}$$

where m_t denotes the mask of valid token pairs (non-padding) and ϵ is added for computational stability.

The raw value of this penalty was then transformed using a nonlinear scaling function based on a sigmoid curve, which

allowed for a controlled increase of its impact depending on the level of repetition.

$$\tilde{s}(x) = A \cdot \frac{\sigma(k(x - x_0)) - \sigma(-kx_0)}{\sigma(k(1 - x_0)) - \sigma(-kx_0) + \epsilon}$$

where:

- A denotes the maximum weight (scale) of the repeat penalty in the final loss function,
- k controls the steepness of the sigmoid function's transition (the larger k is, the more "threshold-like" the behavior)
- x_0 defines the activation point of the function, i.e., the input value at which the penalty begins to increase sharply,
- ϵ is a numerical constant ensuring the stability of the calculations.

The function has been normalized relative to two reference points corresponding to the values $x=0$ and $x=1$. This process, known as anchoring, ensures that the function's output remains limited to the interval $[0,A]$ in practical terms, regardless of the range of input values typically observed during training.

This approach allowed for adaptive punishment of repetition cases: at low repetition rates, the effect of punishment was limited and did not interfere with the learning process. However, once a threshold (x_0) was crossed, its effect became a major component of the loss function, stabilizing the generation and increasing the diversity of results.

D. Discriminator dominance over the generator

In the early stages of training, the discriminator achieved high accuracy very quickly, which led to a situation where the generator stopped receiving a useful gradient signal. This problem is characteristic for GAN training and stems from the difficulty imbalance between:

- real/fake classification,
- and the generation of realistic text sequences.

In early experiments, there were attempts to limit the discriminator's dominance by updating its parameters less frequently. The discriminator was trained only every N batches (in the final configuration, updates were performed every 10 batches).

$$global_step \bmod N = 0$$

Despite the reduction in training frequency, the discriminator still achieved nearly perfect classification accuracy after some time. As a result, the generator received an increasingly weaker gradient signal, which led to a stagnation of adversarial learning.

To further stabilize the learning process, adaptive control of the discriminator's training was applied, based on the exponential moving average (EMA) of accuracy:

$$EMA_t = \beta EMA_{t-1} + (1 - \beta) acc_t$$

The accuracy value used to update the EMA was calculated periodically, every N batches, which helped reduce measurement variance and stabilize the decision to activate or

deactivate discriminator training. If the EMA accuracy value exceeded a specified upper threshold, discriminator training was temporarily disabled; once it fell below a lower threshold, it was reactivated.

This approach allowed for dynamic balancing of the generator and discriminator strengths during training and limited instances of discriminator dominance.

Additionally, updating the discriminator less frequently reduced computational overhead, since discriminator forward and backward passes were not performed at every training step. This resulted in more efficient training without affecting the overall optimization dynamics of the model.

IV. MEMORY ISSUES AND LIMITATIONS OF AUTOREGRESSION

The autoregressive nature of text generation resulted in very high memory requirements during training. In the classical approach, one would need to maintain the full computational graph for all generation steps to enable gradient propagation across the entire generated sequence. In practice, the memory cost of such a solution increases very rapidly with the length of the sequence, as each generation step requires storing additional intermediate representations needed during the backward pass.

During the experiments, three possible gradient propagation strategies presented in Table II were analyzed.

TABLE II
TRAINING STRATEGIES.

Approach	Gradient scope	Memory cost	Sequence context
Full autoregressive graph	full through every generation step	very high	full
Detached generation + full-sequence forward	full for final sequence only	moderate	full
Single-token optimization	last token only	low	limited

The full autoregressive graph approach involves maintaining the full computational graph for all generation steps. This solution ensures full gradient propagation across the entire sequence; however, memory consumption increases very rapidly with the length of the sequence and the number of autoregressive steps.

The other extreme solution would be single-token optimization, in which the gradient is propagated only for the single last generated token without considering the full context of the generated sequence. This approach significantly reduces memory requirements, but leads to a loss of information about dependencies between earlier tokens and destabilizes the process of generating longer sequences.

The final version of the system employed a two-stage approach to autoregressive training as shown in Alg. 2. In the first stage, the sequence was generated autoregressively without maintaining the full computational graph. After a

token was generated, it was detached from the graph, which prevented memory costs from accumulating during subsequent generation steps.

After generation was complete, an additional forward pass was performed on the entire generated sequence. It was only at this stage that the soft representations of tokens were computed, which were used during training and for the backward pass.

This approach represents a compromise between:

- full autoregressive gradient propagation through all generation steps,
- and the complete elimination of information about the global context of the sequence.

Thanks to the second forward pass, the model continued to receive a gradient signal dependent on the entire generated sequence, but without the need to maintain the full computational graph for each autoregressive step. This significantly reduced memory consumption while preserving information about the global context of the text.

This approach was made possible by the use of causal masking in the Transformer decoder. When computing the representation for position t , the model has access only to tokens prior to t .

In practice, this means that the representation of the generated token depends only on the previously generated context, just as in classical token-by-token autoregressive generation.

As a result, the forward pass performed on the entire generated sequence remains consistent with the autoregressive generation process. The representations computed for individual positions correspond to the same contextual dependencies that existed during step-by-step generation.

Therefore, autoregressive masking allowed us to separate:

- the sequence generation process,
- from the gradient calculation process,

without loss of information about the local linguistic context.

V. SUMMARY

This work focused on identifying and addressing key challenges related to training stability in a CycleGAN-inspired architecture for the task of diachronic text style transfer between Early Modern English and modern English of the XXI century.

Key challenges, such as the instability of adversarial learning, the problem of non-differentiability of token sampling, the repetition of generated sequences, the dominance of the discriminator, and the high memory requirements of autoregressive generation, were effectively addressed through a series of mechanisms proposed above.

GENERATIVE AI DISCLOSURE STATEMENT

During the preparation of this work the author(s) used ChatGPT in order to improve the language and clarity of the manuscript. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

Algorithm 2 Generating sequence

```

1: Initialize target sequence with SOS token:  $tgt \leftarrow [SOS]$ 
2: for each generation step  $t$  do
3:   Compute next tokens:  $logits \leftarrow G(src, tgt, p_{tgt})$ 
4:   Sample last token distribution using Gumbel-Softmax:
      $p_t \leftarrow \text{GumbelSoftmax}(logits_t, \tau)$ 
5:   Select next token:  $x_t \leftarrow \arg \max(p_t)$ 
6:   Append generated token to target sequence:
      $tgt \leftarrow tgt \oplus x_t$ 
7:   Detach generated sequence:  $tgt \leftarrow detach(tgt)$ 
8:   Update EOS mask
9:   if all sequences finished then
10:     break
11:   end if
12: end for
13: Perform second forward pass on full generated sequence
    (w/o last token):  $logits_{full} \leftarrow G(src, tgt[: -1], p_{tgt})$ 
14: Compute differentiable soft token representations:
     $soft\_tokens \leftarrow \text{GumbelSoftmax}(logits_{full}, \tau)$ 

```

REFERENCES

- [1] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks,” in *2017 IEEE International Conference on Computer Vision (ICCV)*. Venice: IEEE, Oct. 2017. doi: 10.1109/ICCV.2017.244. ISBN 978-1-5386-1032-9 pp. 2242–2251. [Online]. Available: <http://ieeexplore.ieee.org/document/8237506/>
- [2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative Adversarial Nets,” in *Advances in Neural Information Processing Systems*, vol. 27. Curran Associates, Inc., 2014.
- [3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is All you Need,” in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html
- [4] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville, “FiLM: Visual Reasoning with a General Conditioning Layer,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. doi: 10.1609/aaai.v32i1.11671. [Online]. Available: <https://ojs.aaai.org/index.php/AAAI/article/view/11671>
- [5] A. Niedziółka-Domański and J. Bylina, “Adapting CycleGAN architecture for Unpaired Diachronic Text Style Transfer,” Kraków, Poland, Oct. 2025. doi: 10.15439/2025F4661 pp. 81–86. [Online]. Available: https://annals-csis.org/Volume_44/drp/4661.html
- [6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019. doi: 10.18653/v1/N19-1423 pp. 4171–4186. [Online]. Available: <https://aclanthology.org/N19-1423/>
- [7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://papers.nips.cc/paper_files/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html
- [8] E. Jang, S. Gu, and B. Poole, “Categorical Reparameterization with Gumbel-Softmax,” arXiv, Aug. 2017. doi: 10.48550/arXiv.1611.01144 ArXiv:1611.01144 [stat.ML]. [Online]. Available: <http://arxiv.org/abs/1611.01144>