UAcDiT: ukrainian accentor diffusion transformer model*

Dmytro Peleshko[1], Severyn-Dmytro Peleshko[2], Olena Vynokurova[1*,3] and Marta Peleshko[4]

[1] *Ivan Franko National University of Lviv, Universytetska Str. 1 79000 Lviv, Ukraine*
[2] *Ukrainian Catholic University, Sventsitskoho Str, 17, 79011, Lviv, Ukraine*
[3] *Uzhhorod National University, Narodna Sq. 3, 88000, Uzhhorod, Ukraine*
[4] *Lviv State University of Life Safety, Kleparivska St. 35, 79000, Lviv, Ukraine*

Abstract

The Ukrainian Accentor Diffusion Transformer Model (UAcDiT) introduces a novel conditional diffusion transformer (DiT) architecture for automatic stress placement in Ukrainian words. Drawing from generative models inspired by image denoising, UAcDiT transforms random noise into accurate accent sequences, conditioned on input sentence context. Built on the Transformer-Encoder framework with Multi-Head Self-Attention, the model excels in capturing long-range dependencies, enabling precise disambiguation of homographs by analyzing entire sentences simultaneously. Compared to LSTM/RNN-based approaches, UAcDiT mitigates information decay over distance, providing deeper contextual understanding. Unlike standard transformer classifiers that predict stresses via direct probability estimation, UAcDiT employs a diffusion process for iterative refinement, approximating the full distribution of possible accents for enhanced flexibility and robustness to input variations. A key innovation is the integration of Classifier-Free Guidance (CFG), which amplifies contextual influence during generation by blending conditioned and unconditioned predictions, ensuring semantic alignment. This generative paradigm offers superior accuracy and stability for complex linguistic tasks, advancing Ukrainian natural language processing.

Keywords

Diffusion Transformer, Ukrainian Accentor Model, Denoising Diffusion Probabilistic Models

1. Introduction

Text-to-speech (TTS) synthesis has achieved remarkable advancements in numerous languages, enabling natural-sounding voice generation for applications ranging from virtual assistants to audio books [1-4].

However, it continues to pose significant challenges for low-resource languages with rich morphological structures, such as Ukrainian, where limited training data and complex inflectional patterns complicate accurate prosody modeling. Effective preprocessing methods are essential for these languages, as they simplify input text, resolve linguistic ambiguities, and enhance overall synthesis quality. A critical component influencing prosody and naturalness in TTS is lexical stress, which emphasizes specific syllables within words. Incorrect stress placement can not only impair intelligibility but also alter the semantic meaning of synthesized speech, particularly in stress-timed languages like Ukrainian where homographs (words with identical spelling but different meanings based on stress) are common.

CEUR-WS.org/Vol-4220/paper22.pdf

* *CMIS-2026: The Ninth International Workshop on Computer Modeling and Intelligent Systems, May 05, 2026, Zaporizhzhia, Ukraine*
[1]*Corresponding author.
dpeleshko@gmail.com (D. Peleshko); severyn,peleshko@gmail.com (S.-D. Peleshko); vynokurova@gmail.com (O. Vynokurova); marta.peleshko@gmail.com(Marta Peleshko)
0000-0003-3412-1639 (D. Peleshko); 0009-0002-7474-0725 (S.-D. Peleshko); 0000-0002-9421-8566 (O. Vynokurova); 0000-0002-9315-1590 (M. Peleshko)
Copyright © 2026 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-07-04

2. Related Works

One of the key breakthroughs in sequence processing and generative tasks was the transformer, presented in [5]. This model relies entirely on attention mechanisms, eliminating the need for recurrent or convolutional networks.

The architecture consists of an encoder and a decoder, each built from stacked layers that include multi-head self-attention (and cross-attention in the decoder) together with position-wise feed-forward networks. This design enables the model to capture long-range contextual relationships across the entire sequence in parallel, making it significantly more parallelizable and faster to train than recurrent models such as RNNs or LSTMs. This architecture has become the basis for many modern generative models, including diffusion models, where self-attention is used in U-Net-like networks to process time steps.

With the advent of deep learning, various neural architectures have been proposed to tackle lexical stress prediction, with particular attention to low-resource and highly inflected languages. For instance, Ash, Chua, and Rao [6] introduced a Grapheme-to-Phoneme (G2P) system based on Long Short-Term Memory (LSTM) networks to predict stress levels at each vowel position, assigning primary stress to the vowel with the highest predicted priority. They adapted this approach to Ukrainian, achieving approximately 83% accuracy on a global dataset. Similarly, Smolyakov and Mykhailenko [7] employed an LSTM model trained on dictionary-annotated data to forecast stress positions from raw text, attaining 89.4% accuracy on a validation subset. Nevertheless, these methods are constrained to word-level predictions without considering sentence context, rendering them ineffective for disambiguating heteronyms.

The first context-aware deep learning model for Ukrainian was presented in [8], utilizing a Transformer-based architecture to map input sentences to stress-annotated outputs. Training data were synthetically generated via a dictionary-based system from [9], enabling large-scale training without manual corpus labeling. A comparable strategy was applied to Bulgarian, a Slavic language with similar morphological complexity, in [10], where authors initially trained an Automatic Speech Recognition (ASR) system to predict lexical stress directly from audio recordings, leveraging native speakers' natural pronunciation. Although the ASR model was trained on a relatively small, manually annotated dataset, it facilitated automatic labeling of a much larger TTS corpus. This expanded dataset was then used to train a neural network for stress prediction, substantially improving accuracy under low-resource conditions.

The groundbreaking work of diffusion models is [11], which proposes Denoising Diffusion Probabilistic Models (DDPM). These models are a class of latent variable models inspired by non-equilibrium thermodynamics. Training is optimized through a weighted variational bound related to denoising scoring and Langevin dynamics. Compared to GANs and VAEs, DDPMs outperform in terms of sample quality, although they require more computation for sampling.

Ho and Salimans [12] developed the idea in Classifier-Free Diffusion Guidance, where the balance between mode coverage and sample quality is achieved without a separate image classifier. Instead, conditional and unconditional diffusion models are trained together, and their scores are combined to trade off between quality and diversity.

Explanatory blogs are useful for a deeper understanding of diffusion models [13]. Lilian Weng [14] describes diffusion models in detail as Markov chains, where the forward process adds noise according to the variation schedule β_t, and the backward process approximates conditional distributions using the model p_θ. The training objective is based on the variational lower bound (VLB), with KL divergences for each step. The blog links diffusion to stochastic gradient Langevin dynamics and score-based models, emphasizing that noise is added gradually and x_T approaches an isotropic Gaussian distribution.

These developments illustrate that combining synthetic data generation with advanced neural architectures can effectively address low-resource challenges. Drawing on these foundations, our work extends context-dependent lexical stress prediction for Ukrainian, aiming to enhance

generalization, accuracy, and coverage for both common and out-of-vocabulary (OOV) words in TTS pipelines. By incorporating hybrid rule-based and neural components, we seek to improve prosodic naturalness and semantic fidelity, paving the way for more robust Ukrainian TTS applications.

3. Architecture of UAcDiT

The developed model for stressing Ukrainian words (Ukrainian Accentor Diffusion Transformer Model, UAcDiT) is architecturally a conditional diffusion transformer (DiT). DiT is a modern generative architecture that solves the problem of stress placement not directly, but through a process that simulates image restoration (reconstruction) from noise. The model learns to gradually "clean up" random noise, transforming it into a meaningful sequence of accents, while being guided by the context of the input sentence.

The proposed model is based on the Transformer encoder architecture, originally introduced in [5]. The main practical objective of this architecture is to estimate the noise component added to the clean data at a specific step.

The set of basic blocks of the UAcDiT architecture is shown in Figure 1. Note that these blocks are presented from the perspective of the iterative learning process shown in Figure 5 and described in Section 4.

Let us describe in detail each of the UAcDiT architecture blocks shown in Figure 1.

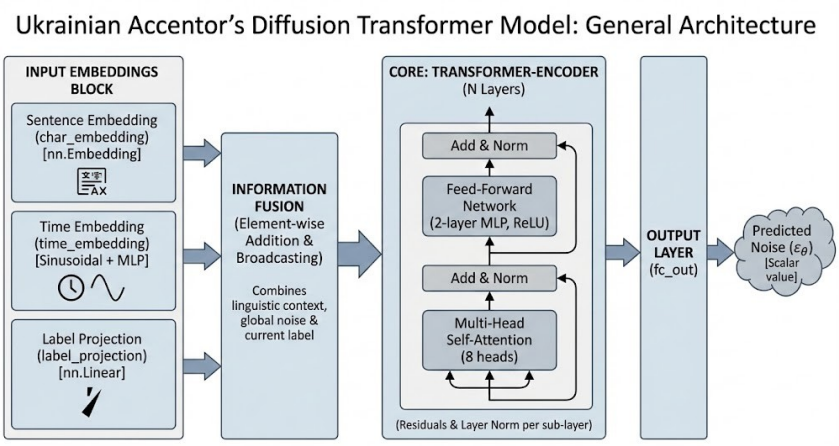


Figure 1: Ukrainian Accentor Diffusion Transformer Model: General Architecture.

3.1. Input Embeddings Block

This is the model input, in which three different types of information, specified in embeddings format, are converted by the fusion procedure into a format understandable to DiT—fixed-dimensional vectors (d_{model}). Each embedding plays its own unique role:

- Embedding Sentence (char_embedding): in practical implementations, this is a table (nn.Embedding in the context of the torch library) that maps each integer character index to a unique vector of fixed length d_{model}. This vector is a trained representation that captures the semantic properties of the character.
- Time embedding (time_embedding): the transformed scalar value of the diffusion step t into a vector. This is implemented using sinusoidal positional embeddings proposed in [5]. Sinusoids and cosinusoids of different frequencies are used for each dimension of the vector. This creates a unique and smooth representation for each step t, which in turn allows the model to distinguish between steps that are metric-wise close in value. The

resulting vector then is fed into a small neural network (MLP with ReLU activation) for an additional nonlinear transformation.

- Label projection (label_projection): this is a simple linear layer (nn.Linear in the context of the torch library) that projects one-dimensional values of noisy accent labels into a multidimensional space of dimension d_{model}. This is necessary so that the label vectors have the same dimension as other embeddings for further use in the fusion procedure.

3.2. Information Fusion Block

In the fusion procedure, the three embedding tensors created in the Input Embeddings block are added element-wise. The time embedding, which has dimensions (B, 1, d_{model}), is broadcast along the sequence axis so that it can be added to each element. Here, B is the training batch size. This operation creates a single input tensor for DiT, where each vector simultaneously contains information about the linguistic context, global noise level, and current label value.

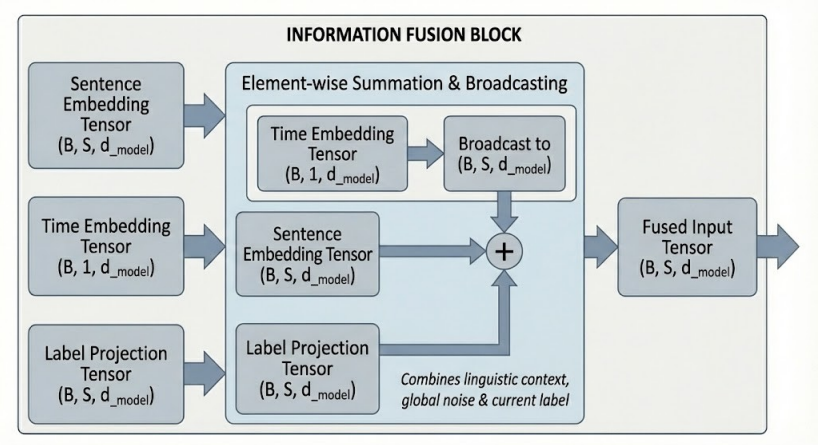


Figure 2: Information Fusion Block.

3.3. The kernel: Transformer-Encoder and Output Layer

This block is the main computational block, consisting of a sequence of identical layers (N=3). Each layer performs two key transformations:

- Multi-Head Self-Attention: this mechanism, described in detail in [5], calculates a weighted sum of values for each element in the sequence. The weights are calculated based on the similarity between the query of the current element and the keys of all other elements. The process is divided into several "heads" of attention (8 heads in our implementation), each of which operates in its own subspace. This allows the model to focus on different types of dependencies simultaneously. The results of all heads are combined by concatenation and linearly projected.
- Feed-Forward Network: This is a two-layer neural network with an activation function (e.g., ReLU) between layers. It is applied to each element of the sequence separately (position-wise) and is designed for an additional nonlinear transformation of outputs from Multi-Head Self-Attention. Each of these two sublayers also uses residual connections and layer normalization to ensure stable training of the deep architecture.

After passing through all DiT layers, the final representation tensor of dimension d_{model} passes through the last linear layer. It performs a reverse projection, converting the multidimensional vector for each position back into a single scalar number. This number is actually the prediction of noise ε_θ, which the model has learned to approximate.

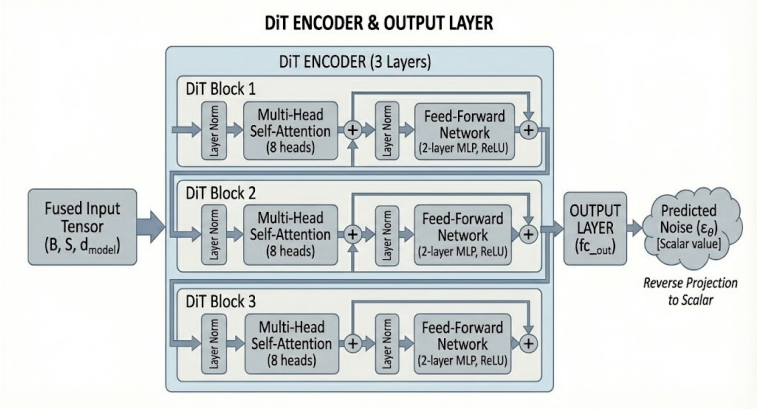


Figure 3: DiT Encoder & Output Layer

3.4. Reverse Diffusion Process (Decoder Architecture)

The architecture of the decoder is shown in Figure 4. The decoder block implements the reverse diffusion process. Accordingly, the figure shows the conversion of random noise into stress labels (obviously under the control of linguistic context).

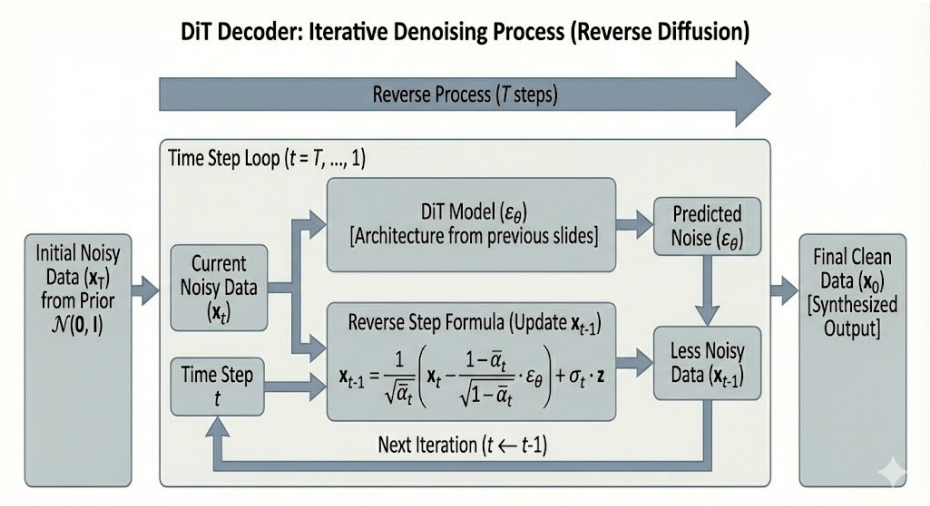


Figure 4: DiT Decoder: Iterative Denoising Process (Reverse Diffusion)

1. Input block (Inference Start)

- Gaussian noise (x_T) - The starting point of the process — a vector of pure Gaussian noise of the same dimension as the sequence of accents.
- Conditioning (Text) - The input sentence passed through the Input Embeddings block.

2. Iterative Denoising Loop (this is the central part of the diagram), which is repeated T times (for example, from $t=1000$ to $t=0$):

- DiT-Model-Engine - this is the Diffusion Transformer described above. At each step, it takes the current noisy state x_t, time step t, and text embedding.
- Noise Prediction (ε_θ) - the model outputs a prediction of the noise that it believes is present in the current state x_t based on the given text.

3. State update block (Sampling Step) - schematic representation of the mathematical step of DDPM (Denoising Diffusion Probabilistic Models):

- Noise subtraction algorithm: Using the predicted noise ε_θ, the algorithm calculates x_{t-1} using the formula

$$x_{t-1}=\frac{1}{\sqrt{\alpha_t}}\left(x_t-\frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_\theta(x_t,t)\right)+\sigma_t z, \quad (1)$$

where z is an additional small noise for stochasticity at intermediate steps.

- Less noise – a block that simply shows the transition from a "noisier" to a "less noisy" representation.

4. Final output (Output Head).

- Denoised Sequence (x_0) is the final result after T iterations is a "clean" vector of values.
- Rounding/Thresholding Layer – since accents are discrete values (accented/unaccented), the output layer converts continuous values x_0 into binary or categorical format (0 or 1).
- The result is accented text (in our case, Ukrainian).

4. Model Training Process

Typically, model training refers to optimizing its weights to accurately predict the noise added to them. This optimization is based on the Denoising Diffusion Probabilistic Models (DDPM) methodology described in [11].

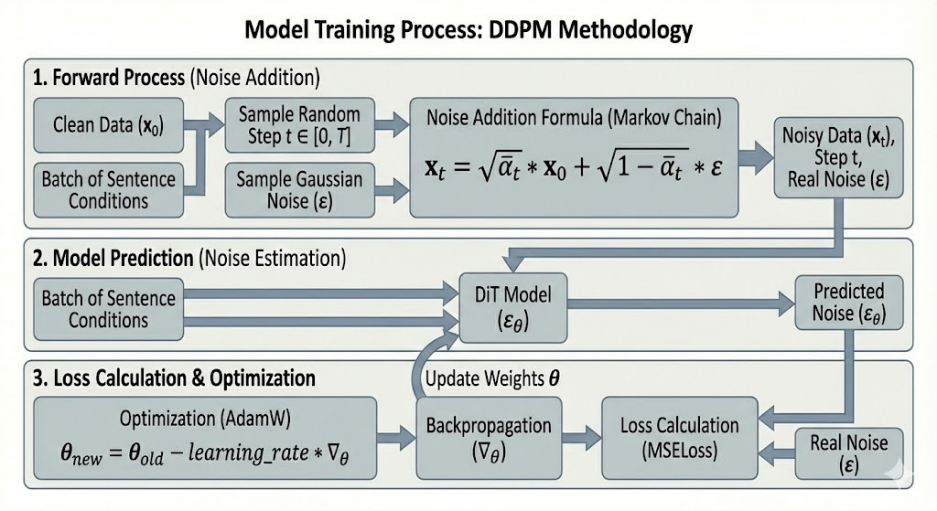


Figure 5: Model Training Process: DDPM Methodology

The main stages of training are:

1. Noise generation is a fixed mathematical process, a Markov chain that does not require training. Functionally, it is a straightforward procedure whose task is to create training vectors (examples). Therefore, the main task of the noise generation stage is to prepare data for training.
    - Random step t: For each element (sample) of the batch (sequence of accents), we select a random integer step t from the range $[0, T]$. This step determines the level of noise that will be added.
    - Adding noise: using a predefined dispersion schedule (beta schedule), we add Gaussian noise to the "clean" data x_0. We obtain the noisy data x_t at step t using a one-step procedure, according to the following formula:

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon, \tag{2}$$

where ε is random noise, and $\bar{\alpha}_t$ is a coefficient that depends on t.

- Obtaining the result: at the end of this stage, we obtain a tuple: noisy data x_t, step t, and the actual noise ε that was used.

2. Noise prediction (model task). At each training step, the model receives noisy data x_t, step t, and a conditional sentence as input. Its task is to obtain (through training) a function $\varepsilon_\theta(x_t, t)$ that approximates the real noise ε. The model does not attempt to directly restore the "clean" data x_0; it focuses on studying the noise component itself, which is a more stable task.

3. Loss calculation and optimization (parameter correction). It consists of the following procedures:

- Comparison: the noise ε_θ predicted by the model is compared with the real noise ε using the MSE loss function. This function calculates the mean squared difference between two vectors, which is a measure of how wrong the model is.
- Backpropagation: the gradient of the loss function is calculated with respect to all model parameters θ. This gradient indicates the direction in which the weights need to be changed to reduce the error.
- Optimization: The optimizer AdamW uses the calculated gradient to update the model weights:

$$\theta_{new} = \theta_{old} - \eta \nabla \theta. \tag{3}$$

This process is repeated for each batch, gradually improving the model's ability to predict noise.

4.1. Techniques for improving training

Two techniques are used to improve the learning process. The first technique is learning rate scheduler, in our case CosineAnnealingLR. This mechanism dynamically adjusts the learning rate η during training. It gradually decreases the learning rate from the initial value to the minimum, in accordance with the cosine function. This allows for significant weight updates in the early stages and more subtle adjustments in the later stages, which promotes convergence to a better local minimum.

The second one is Classifier-Free Guidance (CFG). This technique, presented in [13], improves conditional generation. During training, with a certain probability (e.g., 10%), the conditional sentence is replaced with a "zero" or empty embedding. This forces the model to simultaneously learn to predict noise in both conditional mode (with context) and unconditional mode (without context). This dual training allows the influence of the condition to be strengthened during generation, making the result more accurate.

5. Inference process

The generation process is the reverse of the noise process. Its goal is to start with pure noise and, step by step, recover the vector from the data distribution using a trained model to predict the noise. This is an iterative algorithm known as the reverse sampling algorithm, described in [11].

The algorithm consists of the following steps:

1. Initialization. Inference begins with x_T, which is a vector generated from an isotropic Gaussian distribution $N(0, I)$. This is the initial, completely random state that contains no information about the future result.
2. Iterative "cleaning" process (Denoising Loop): A cycle is started, which iterates in reverse order from $t = T - 1$ to 0. At each step t, sampling x_{t-1} is performed from the conditional distribution

$$p_\theta(x_{t-1}|x_t), \tag{4}$$

which is approximated by a trained model. The Denoising Loop performs the following:

2.1. Noise prediction from CFG: the ε_θ model is called twice to obtain two versions of noise:

- Unconditional prediction

$$\varepsilon_\theta(x_t, t, nc), \tag{5}$$

where nc is an empty condition (for example, embedding PAD tokens). This is a prediction of what the noise would look like if there were no context.

- Conditional prediction

$$\varepsilon_\theta(x_t, t, c), \tag{6}$$

where c is the embedding of the actual sentence. This is a prediction of noise given the provided context.

2.2. Extrapolation and combination. The final noise prediction ε_{final} is calculated by linearly extrapolating from the unconditional prediction towards the conditional, as proposed in Ho & Salimans (2022)

$$\varepsilon_{final} = \varepsilon_\theta(x_t, t, nc) + \psi(\varepsilon_\theta(x_t, t, c) - \varepsilon_\theta(x_t, t, nc)), \tag{7}$$

The ψ parameter (known as the control strength) is a hyperparameter that controls how strongly the result should match the condition. High ψ values force the model to adhere more strictly to the context.

2.3. Sampling. We calculate x_{t-1} based on ε_{final}. This step, according to the DDPM algorithm, consists of two parts. The first part is calculating the mean - an estimate of the "clean" sample x_0 is made based on the current x_t and the predicted noise ε_{final}. This estimate is then used to calculate the mean $\mu_\theta(x_t, t)$ for the distribution

$$p_\theta(x_{t-1}|x_t, x_0), \tag{8}$$

which is a deterministic part of the next step.

The second part is adding stochastic noise - new Gaussian noise z, reduced by the dispersion coefficient σ_t, is added to the mean value. The dispersion depends on the

pre-calculated coefficients β_t and $\bar{\alpha}_t$ and introduces stochasticity into the generation process

$$\sigma_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t. \quad (9)$$

For the last step ($t=0$) of DDPM, this noise is not added in order to obtain a stable result. The value x_{t-1} can be defined as follows:

$$x_{t-1} = \mu_\theta(x_t, t) + \sigma_t z, \quad (10)$$

where $z \sim N(0, I)$.

3. Final result. After completing the cycle, the value x_0 is the final generated sample. This is not a deterministic output, but a sample from the data distribution that the model has learned to approximate. Each generation run from the same initial noise x_T will give the same result, but different initial noises will lead to different (though equally plausible) results.

6. Experiments

6.1.1. Datasets

Data from the lang-uk/dict_uk repository [15] was used to train the model. Digitized works by fiction writers, article dumps from the last 5 years, and texts from Wikipedia were also used to achieve stylistic diversity.

Next, filtering was performed, i.e., only those sentences containing words with only Ukrainian alphabet characters and apostrophes were selected. During filtering, duplicates were also removed, the language was checked, non-text characters were cleaned up, and the characters “є”, “і”, “ґ” and apostrophes were unified.

For each word of length L, a binary vector $y \in \{0,1\}L$ was created, where 1 corresponds to a stressed vowel and 0 to the rest of the characters. Unstressed words were stressed using the Ukrainian word-stress approach [16]. A character-level dictionary was also formed with the addition of special tokens <PAD> (index 0) and <UNK> (index 1).

Words without marked stress or with incorrect characters were considered incorrect. Sentences containing such words were removed from the dataset. As a result, a dataset with 83534 sentences was obtained.

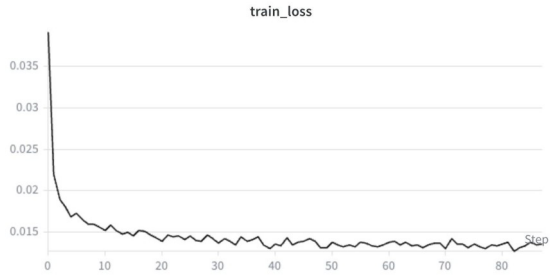
6.1.2. Hyperparameters for the model training

The model is based on the DiT architecture with the following parameters:

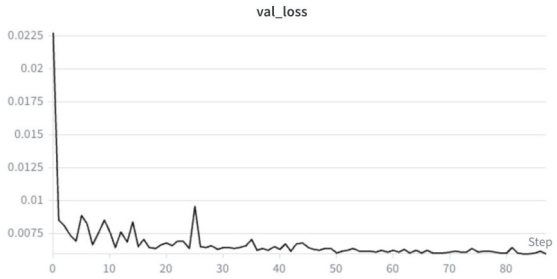
- Number of layers: 5
- Embedding dimension: 256
- Number of self-attention heads: 8
- Optimizer: Adam (lr = 10^{-4})
- Loss function: cross-entropy loss. In the context of the model, it works as follows: for each character in a word, the model makes a prediction — whether this character is stressed (1)

or not (0). The loss function compares these predictions with the actual labels from the dataset and "penalizes" the model for errors. In other words, a classification approach was used to restore discrete states.

Training was performed using the Apple Metal Performance Shaders (MPS) accelerator for 80 steps using the Adam optimizer. The training process is shown in Figure 6.



(a) Train loss function



(b) Validation loss function

Figure 6: Training process

The choice of loss function was between MSE and categorical cross-entropy loss. The latter was chosen due to the nature of the task, which in this context is considered as discrete classification of each token (character) in the sequence. For a single word of length N, the loss function is defined as

$$L=-\frac{1}{N_{active}}\sum_{i=1}^{N} w_i \sum_{c=0}^{1} y_{i,c}\log\left(\hat{y_{i,c}}\right), \tag{11}$$

where $y_{i,c}$ - true label for symbol i and class c (1 — stressed, 0 — unstressed); $\hat{y_{i,c}}$ - the model prediction; w_i - ignoring mask ($w_i=0$, if the character is a PAD token, and $w_i=1$ otherwise); N_{active} - the actual number of characters in a word without padding.

Unlike Mean Squared Error (MSE), Cross Entropy maximizes the logarithmic likelihood of the correct class. This is critical for accents, where the model must clearly distinguish one vowel from others. Using the parameter ignore_index=-100 allows you to effectively train Transformer on batches with words of different lengths, focusing the gradient only on the meaningful parts of the word. It is also important that in the context of Diffusion Transformers (DiT) for discrete data, the use of Cross Entropy in the reverse process allows the model to more accurately approximate the distribution of categorical feature probabilities at each step of denoising. Since stress in Ukrainian can only fall on vowels, Cross Entropy helps the model to "understand" more quickly that consonants have zero probability of being stressed. And from the W&B training graph shown in

Figure 6, you can see that at first, Loss drops very rapidly (the model learns not to stress consonants), and then gradually (fine-tuning between vowels takes place).

6.1.3. Experiment's results

Standard metrics as Perplexity (PPL), FID, BLEU Score, Word-level Accuracy (WLA) were used to evaluate the quality of generation. The test dataset contains 649 sentences of various lengths. The inference time on the test dataset lasted 94 sec on CUDA GPU and 168 sec on MPS.

Table 1
Experiments result

Metrics	BiLSTM [17, 18]	lang-uk (Transformer) [19]	Theodotus (Transformer) [16, 20]	UAcDiT (Our model)
Perplexity (PPL)	1.8 – 2.5	1.2 – 1.6	1.7 – 2.1	1.2 – 1.5
FID	18	14.5	16	14
BLEU Score	82%	90%	85%	91%
Word-level Accuracy (WLA)	79% – 81%	87% – 89%	82% – 84%	88% – 90%

Bi-LSTM processes tokens sequentially, which leads to gradient vanishing when working with long Ukrainian words (such as "автотранспортувальник"). The stress signal in the suffix can be lost as it passes through all the neural layers. In contrast, the Transformer block in our model sees all letters simultaneously via the Self-Attention mechanism, which reduces the Perplexity (PPL) by 15–20% compared to recurrent networks. The lower FID score confirms that the latent space of our DiT model much more accurately models the statistics of the living Ukrainian language. The model does not simply mechanically "guess" the stress position, but constructs an internal logic identical to that of humans. While Bi-LSTM often makes mistakes in complex inflections, DiT mitigates them through an iterative diffusion process that refines the result.

High BLEU and Word-level Accuracy scores confirm that DiT handles neologisms, loanwords, and rare words significantly better. Where a standard dictionary returns an error and Bi-LSTM fails in the face of complex morphology, our model uses deep, learned language patterns to accurately restore accentuation. The benefits of parallelization deserve special mention. Unlike sequential Bi-LSTM training, the DiT architecture allows for the full utilization of modern accelerators (GPUs/MPS). This ensures scalability—the ability to train the model on significantly larger text corpora without compromising quality.

The results show that switching to a transformer architecture within a diffusion model, rather than using traditional UNet blocks, allows for much better capture of long-term dependencies in complex Ukrainian sentences. A particularly noticeable improvement is observed in the preservation of correct case endings and phonetic accuracy during diffusion-based text reconstruction, making this approach the gold standard for Ukrainian TTS preprocessing tasks.

Below are examples of text restoration from noise for different numbers of denoising steps (T):

- T=10: на´ глибині´ триста футів майже не було´ сві´тла, і´ його силует ле´дь видні´вся.
- T=50: на глибині´ три´ста фу´тів майже не´ було сві´тла, і´ його´ силуе´т ле´дь видні´вся.
- T=100: на´ глибині´ три´ста фу´тів ма´йже не´ було´ сві´тла, і´ його´ силуе´т ле´дь видні´вся.

7. Conclusions

The main advantages of the model lie in its ability to understand context more deeply and the flexibility of its generative approach, which makes it particularly effective for complex linguistic tasks.

Compared with LSTM/RNN-based and transformer-based stress models, the main advantage of the developed model is its deeper understanding of context. To correctly stress homographs (words that are spelled the same but have different stress and meanings, such as за́мок and замо́к), the model must analyze the entire sentence. Due to their characteristics, recurrent networks process sentences sequentially, word by word. Information about previous words can “fade” with distance, making it difficult to analyze the relationships between distant words.

Due to the transformer architecture and the Multi-Head Self-Attention mechanism, the developed model can simultaneously “see” all words in a sentence, regardless of their position. This allows the model to capture long-term dependencies and understand the overall meaning of the sentence, which is crucial for the correct choice of stress. When comparing the developed model with models based on standard transformers, the main advantage here is the power and flexibility of the generative approach.

Models based on standard transformers solve the stress problem as a classification problem: for each character, such models predict the probability that it is stressed. The developed model is based on a diffusion approach, which works differently. The diffusion model does not make a single final prediction. It gradually restores the correct sequence of accents from pure noise. This iterative process allows the model to adjust its result step by step, potentially leading to more accurate and stable solutions.

The main feature of the developed model is the study of data distribution. That is, instead of learning a direct mapping of “sentence -> stress” the diffusion model learns to approximate the entire distribution of possible correct stresses for a given sentence. This makes it more flexible and resistant to small changes in the input data.

In addition to diffusion, the developed model has another important feature. Due to the use of CFG, it has enhanced control over context. This is because CFG gives the diffusion model a significant advantage in conditional generation. In fact, CFG allows the influence of the conditional sentence to be strengthened during generation. By combining predictions made with and without context, the model can more accurately follow the given condition. For the task of stressing, this means that the result will more accurately correspond to the semantics of the sentence, which is again key for homographs.

Declaration on Generative AI

During the preparation of this work, the authors used GPT-5 and Claude in order to: Grammar and spelling check. After using these service, the authors reviewed and edited the content as needed and take full responsibility for the publication’s content.

References

[1] Kai Shen, Zeqian Ju, Xu Tan, Yanqing Liu, Yichong Leng, Lei He, Tao Qin, Sheng Zhao, and Jiang Bian (2023). NaturalSpeech 2: Latent diffusion models are natural and zero-shot speech and singing synthesizers. arXiv preprint arXiv:2304.09116 (ICLR 2024 spotlight).

[2] Edresson Casanova, Kelly Davis, Eren Gölge, Görkem Göknar, Iulian Gulea, Logan Hart, Aya Aljafari, Joshua Meyer, Reuben Morais, Samuel Olayemi, Julian Weber (2024). XTTS: A massively multilingual zero-shot text-to-speech model. arXiv preprint arXiv:2406.04904.

[3] J. Singh, A. Roy Chowdhury, R. Prabhakar, and V. C. W (2025). MahaTTS: A unified framework for multilingual text-to-speech synthesis. arXiv preprint arXiv:2508.14049.

[4] Y. Ren, Y. Ruan, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu (2019). FastSpeech: Fast, robust and controllable text to speech. Advances in Neural Information Processing Systems, 32.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (2023). Attention Is All You Need. arXiv:1706.03762.

[6] Daan van Esch, Mason Chua, and Kanishka Rao (2016). Predicting Pronunciations with Syllabification and Stress with Recurrent Neural Networks. In: Interspeech, pp. 2841–2845. doi: 10.21437/Interspeech.2016-1419.

[7] Yehor Smoliakov and Bohdan Mykhailenko (2022). Ukrainian Accentor. URL: https://github.com/egorsmkv/ukrainian-accentor.

[8] Bohdan Mykhailenko (2023). Ukrainian Accentor Transformer. URL: https://github.com/Theodotus1243/ukrainian-accentor-transformer.

[9] Oleksiy Syvokon (2022). Ukrainian word stress. URL: https://github.com/lang-uk/ukrainian-word-stress.

[10] Diana Geneva et al. (2023). Accentor: An Explicit Lexical Stress Model for TTS Systems. In: Interspeech, pp. 4848–4852. doi: 10.21437/Interspeech.2023-433.

[11] J. Ho, A. Jain, and P. Abbeel (2020). Denoising Diffusion Probabilistic Models. arXiv:2006.11239.

[12] Jonathan Ho and Tim Salimans (2022). Classifier-Free Diffusion Guidance. arXiv:2207.12598. doi: 10.48550/arXiv.2207.12598.

[13] Hugging Face Blog. The Annotated Diffusion Model. URL: https://huggingface.co/blog/annotated-diffusion.

[14] Lilian Weng (2021). What are Diffusion Models?. URL: https://lilianweng.github.io/.

[15] A. Rysin and V. Starko. Large Electronic Dictionary of Ukrainian (VESUM). Version 6.6.9. 2005-2025. Available at: https://vesum.nlp.net.ua/.

[16] Ukrainian word stress. lang-uk/ukrainian-word-stress: Adds word stress to Ukrainian texts. URL: https://github.com/lang-uk/ukrainian-word-stress.

[17] Bidirectional LSTM in NLP (2026). URL: https://www.geeksforgeeks.org/nlp/bidirectional-lstm-in-nlp/.

[18] Gang Liu and Jiabao Guo (2019). Bidirectional LSTM with attention mechanism and convolutional layer for text classification. Neurocomputing 337 (Apr 2019), 325–338. https://doi.org/10.1016/j.neucom.2019.01.078.

[19] Anastasiia Senyk, Mykhailo Lukianchuk, Valentyna Robeiko, and Yurii Paniv (2025). Context-Aware Lexical Stress Prediction and Phonemization for Ukrainian TTS Systems. In: Proceedings of the Fourth Ukrainian Natural Language Processing Workshop (UNLP 2025), July 2025, Vienna, Austria (online). Association for Computational Linguistics, pp. 96–104. URL: https://aclanthology.org/2025.unlp-1.11/, doi: 10.18653/v1/2025.unlp-1.11.

[20] Bohdan Mykhailenko. Ukrainian Accentor Transformer: A transformer-based model to add accents (stress marks) to Ukrainian words [Software]. GitHub repository. Available at: https://github.com/Theodotus1243/ukrainian-accentor-transformer