Progress Evaluation

Literature Review

Introduction

Diffusion-LM (Li et al., 2022) [1] identifies a key bottleneck in text diffusion: the model often fails to commit its predictions x_0 to valid discrete tokens, instead producing embeddings that lie off the word-vocabulary manifold . To compensate, it reparametrizes the loss to predict x_0 directly (summing $\|f(x_t,t)-x_0\|^2$ over timesteps) and applies a "clamping trick" at generation time (mapping each predicted vector to its nearest word embedding) [1]. These fixes suggest the original ε–based loss does not strongly enforce discrete fidelity. This review surveys related work on loss re-parametrization and discrete alignment, to inform rounding-aware loss designs that may improve fluency, control, and efficiency in Diffusion-LM.

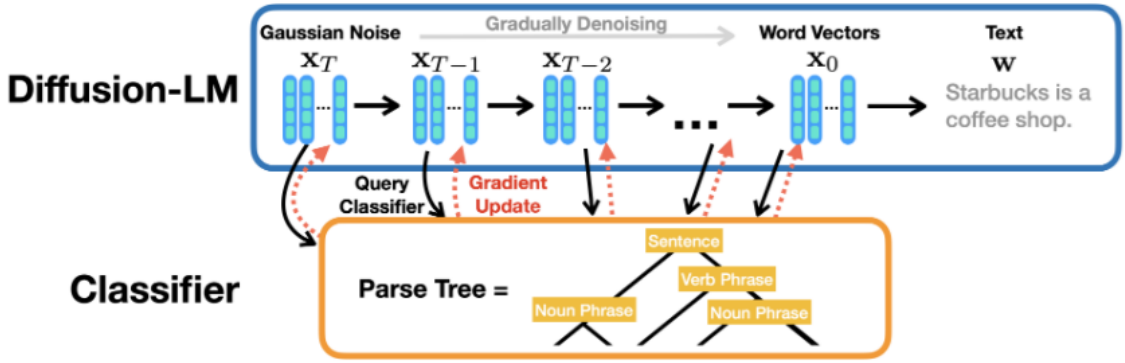


Figure 1: *Diffusion-LM iteratively denoises a sequence of Gaussian vectors into word vectors, yielding a intermediate latent variables of decreasing noise level $x_T \cdots x_0$. For controllable generation, we iteratively perform gradient updates on these continuous latents to optimize for fluency (parametrized by Diffusion-LM) and satisfy control requirements (parametrized by a classifier) [1]*

Loss Function Re-parametrization in Diffusion Models

- **Standard vs. x_0-loss.** Traditional DDPM training predicts the noise (ε) or the denoising mean μ_t (which is a linear combination of x_0) at each step. Li et al. (2022) [1] show that the standard μ_t-loss lacks enough "pull" to force outputs onto true word embeddings. They propose instead to **predict x_0 at every timestep** via a simple loss

$$L_{\text{simple}} = \sum_{t=1}^{T} \mathbb{E}\left[\|f_\theta(x_t, t) - x_0\|^2\right]$$

which drives the network to denoise directly to the original embedding. This x_0-

prediction loss yields faster convergence and ensures that even early predictions nudge the model towards valid tokens.

- **Unifying the objective.** A survey by Li et al. (2023) notes that this x_0-parameterized loss "unifies" the denoising objective across timesteps, effectively simplifying training [2]. Many follow-up text-diffusion models adopt the x_0–loss for this reason. In vision tasks, similar insights led to the use of alternative parameterizations (e.g. predicting the original image x_0 or the "v" prediction of Nichol & Dhariwal, 2021) for better stability. Overall, **shifting the target from noise to data** tends to improve discrete fidelity by directly supervising the network on the desired embedding manifold [2].

Rounding and Discrete Recovery in Diffusion and VAE Models

- **Vector-Quantized autoencoders (VQ-VAE).** VQ-VAE (van den Oord et al., 2017) introduces an explicit discrete latent codebook. It uses a **quantization loss** with two terms: a codebook commitment loss that penalizes the encoder output for straying from its assigned discrete code (to "commit" it to a quantized value), and a reconstruction loss [3]. Gradients are passed through the non-differentiable quantizer via a **straight-through (STE) estimator** [3]. The STE simply copies gradients from the quantized vector to the encoder output, enabling end-to-end training despite hard discretization [3]. These techniques show that adding quantization-aware losses can enforce discrete structure in latent spaces.

- **Gumbel-Softmax and Concrete distributions.** The **Gumbel-Softmax** trick (Jang et al., 2017) replaces a categorical sample with a differentiable soft sample that anneals to one-hot. Jang et al. demonstrate that this reparametrization lets the model learn discrete distributions by backpropagating through a "softmax temperature" that approaches zero [4]. In practice, Gumbel-Softmax (and the related Concrete distribution) is widely used in discrete VAEs and latent models to allow gradient-based training of categorical variables.

- **Additional regularization losses.** In embedding-diffusion models, an auxiliary **rounding loss** (cross-entropy from predicted embedding to true token) is often added. Li et al. (2022) introduced such a loss $-\log p_\theta(w|x_0)$ to ground embeddings, but found it encouraged a degenerate, anisotropic space [2]. Gao et al. (2024) propose instead an **anchor loss** $-\log p_\theta(w|\hat{x}_0)$ using the model's own prediction $\hat{x}_0$ which pushes $\hat{x}_0$ further from its token (increasing separation among embeddings). This anchor loss proved more effective at maintaining a well-distributed embedding space [5]. Such quantization-aware terms (commitment or anchor) are crucial for preventing the collapse of embeddings and improving discrete recoverability.

Discrete Latent Models in Text Generation

- **Non-autoregressive latent variables.** Kaiser et al. (2018) [6] introduce discrete latent variables $l_1, \ldots, l_m$ (with $m < n$) to compress target sequences. These latents are trained via techniques like Gumbel-Softmax and VQ-VAE to ensure they are discrete [6]. The model then predicts the short latent sequence autoregressively and decodes all target tokens in parallel from these latents. This showcases one approach to using discrete bottlenecks for faster generation.
- **Discrete VAEs for linguistic factors.** Recent text-VAE work uses discrete latents to capture interpretable language factors. For example, Zhao et al. (2021) [9] define a set of discrete latent variables (one per linguistic attribute) modeled with Gumbel-Softmax, forming a differentiable "soft" one-hot that eventually collapses to discrete values [7]. These models explicitly enforce discreteness in the encoder and decoder, illustrating one route to infusing generative NLP models with discrete structure.
- **Plug-and-play control.** Plug-and-play approaches (Dathathri et al., 2020 [8]) steer language generation by applying gradients from an attribute model to the LM's hidden states. PPLM does not alter the model's tokens but adjusts its latent activations via backpropagation. While not a discrete latent model per se, PPLM exemplifies an external-control strategy that "wraps" around a pretrained LM for attribute-based generation, and it can be viewed as a latent-modification technique.
- **Latent diffusion models for language.** Lovelace et al. (2023) propose **latent diffusion** in a pretrained language-model latent space: text is encoded into continuous latents, diffused, and then decoded. Relatedly, Lou et al. (2024) and others explore **discrete diffusion** processes for text by defining forward noising on token sequences or embeddings. Although these methods still rely on continuous relaxations during training, they reflect the broader trend of learning (often discrete) latent representations specifically for text generation tasks.

Rounding-Sensitive Evaluation and Sampling Tricks

- **Clamping to vocabulary.** To enforce discrete outputs, Diffusion-LM explicitly **clamps** each predicted embedding to the nearest word vector before the next step. In practice, they map $f_\theta(x_t, t)$ to its closest vocabulary embedding at each denoising step [1]. This ensures that even if the network's raw prediction is off the manifold, the sample is snapped to a valid token, which empirically reduces cumulative rounding errors.
- **Training losses for discretization.** As noted, auxiliary losses help guide embeddings. The rounding loss $-\log p(w|x_0)$ (Li et al., 2022 [1]) and its anchor variant (Gao et al., 2024 [10]) are critical. These losses can be viewed as decoder-supervision terms that **supervise the embedding–token mapping**. They encourage the network to output embeddings that score highly on the correct token's softmax. Gao et al. (2024) [10] show that anchor loss

yields more distinguishable embeddings and mitigates “collapse” better than the naive rounding loss [5] [2].

- **Evaluation metrics for token fidelity.** Feng et al. (2025) [11] introduce complementary metrics to assess discrete accuracy in diffusion text models. They define a **token error rate (TER)** (essentially perplexity) for token-level fluency and a **sequence error rate (SER)** requiring an exact match of the entire sequence. These metrics capture different aspects of rounding fidelity: TER assesses how well each output token matches the reference (lower is better), while SER penalizes any mistakes in the full sequence. Using such metrics highlights whether a model’s errors are sporadic or systemic.
- **Sampling and projection heuristics.** Beyond clamping, some approaches adjust sampling temperature or perform a final projection step. For example, one could sample an embedding and then choose the argmax token (similar to using straight-through at inference). Others propose iterative refinement (e.g. classifier-guided updates) to enforce constraints. In practice, simple tricks like outputting the highest-probability token at the final step (rather than sampling) are often used to ensure coherence. Such heuristic sampling strategies, along with the training losses above, collectively aim to maximize “commitment” of the diffusion output to valid vocabulary items.

Methodology Outline

1. Baseline Setup

Diffusion-LM (Li et al., 2022) [1] is a continuous-diffusion, non-autoregressive language model that generates text by iteratively denoising a sequence of Gaussian noise vectors into continuous word embedding vectors [1]. Each input token is first mapped to an embedding vector, and the forward diffusion process adds Gaussian noise to these embeddings (with $q_\phi(x_0|w) = \mathcal{N}(\mathrm{EMB}(w), \sigma_0^2 I)$) [1]. The model then learns to reverse this process: at each timestep it predicts the noise-free embedding (denoted x_0) or the parameters of the denoising transition. The training loss includes a standard simplified diffusion loss: a mean-squared-error (MSE) on the predicted denoising mean (as in [1]). In practice Li et al. optimize a surrogate $"L_{\text{simple}}"$ term (squared error between predicted and true Gaussian means) [1]. In addition, Diffusion-LM adds a discrete *rounding* loss: a cross-entropy term $-\log p_\theta(w|x_0)$ training a softmax classifier $p_\theta(w|x_0)$ that maps the continuous prediction x_0 back to the nearest vocabulary token [1]. In summary, the baseline objective combines the diffusion MSE loss with an embedding-distance term and the rounding (cross-entropy) loss (see Eq.2 of Li et al.) [1]. Importantly, Diffusion-LM uses an x_0-*parameterization*: the neural network is trained to predict the original embedding x_0 at each step (via a function $f_\theta(x_t, t)$) [1]. At generation time, after the final denoising step, the model rounds the predicted x_0 to the nearest word embedding (taking $p_\theta(w|x_0)$ to produce discrete output tokens.

To further reduce rounding errors, the authors introduce a **clamping trick** during sampling: at each intermediate step the predicted x_0 vector is "clamped" to its nearest embedding before adding noise [1]. This enforces that the denoised vector stays close to some valid token embedding, improving token commitment in the reverse process.

- **Architecture:** A Transformer-based diffusion model (80M parameters in Li et al.) maps noise to word embeddings. During training, each word *w* is encoded as $\mathrm{EMB}(w) \in \mathbb{R}^d$ (with d=16 for E2E, d=128 for ROCStories) [1].
- **Loss:** The loss is the sum of a diffusion MSE term $L_{\text{simple}}(x_0)$ plus terms enforcing x_0 consistency and discreteness. In particular, the training objective includes $||f_\theta(x_t, t) - x_0||^2$ (implicitly via the standard diffusion loss) plus the log-probability $-\log p_\theta(w|x_0)$ for the correct token [1].
- **x_0 reparameterization:** The network $f_\theta(x_t, t)$ is trained to predict the clean embedding x_0 directly [1]. This "x_0-parameterization" ensures that the model focuses on the final embedding throughout denoising.
- **Clamping:** At decode time, after each denoising step the predicted x_0 is snapped to the nearest token embedding (via $\mathrm{Clamp}(f_\theta(x_t, t))$) [1]. This "clamping" ensures the latent commits to a valid word and mitigates rounding errors.

Together, these components (continuous embeddings, diffusion MSE loss, and a rounding classifier with clamping) comprise the original Diffusion-LM baseline [1].

2. Proposed Loss Enhancement

We introduce a **rounding-aware loss term** designed to encourage the model's continuous prediction x_0 to lie close to a valid token embedding even before clamping. In effect, this new term acts as a regularizer pushing the output distribution toward the discrete vocabulary manifold. Two motivating formulations are:

- **Anchor-type MSE loss:** Inspired by Gao et al. (2024) [10] and VQ-VAE ideas [10] [12], we could add an MSE that penalizes the distance between the predicted x_0 and the *nearest* token embedding. Concretely, let $\hat{w} = \arg\min_{w'} |\mathrm{EMB}(w') - f_\theta(x_t, t)|$. Then an "anchor" loss term $L_{\text{anchor}} = |f_\theta(x_t, t) - \mathrm{EMB}(\hat{w})|^2$ encourages the model to predict exactly the embedding of $\hat{w}$. This is analogous to the "anchor loss" proposed by Gao et al. to prevent embedding collapse [10]. Intuitively, by using the model's own prediction to define the loss target, gradients flow through the predicted point and pull it toward the correct embedding, jointly regularizing the embeddings and model.
- **Quantization-aware cross-entropy:** Alternatively, we can formulate a loss based on vocabulary probabilities. For example, compute a softmax over cosine similarities or

negative distances between $f_\theta(x_t, t)$ and all embeddings, then add a cross-entropy that encourages assigning highest probability to the true token or its nearest neighbor. In effect, this "quantization-aware" cross-entropy reinforces commitment to discrete codes. This idea is analogous to the vector-quantized VAE (van den Oord et al., 2017 [12]) which adds a loss to align continuous outputs with a codebook [12].

These terms can be used individually or combined, but both serve the same purpose: making x_0 predictions lie close to real embeddings. For example, one could define

$$L_{\text{quant}} = -\log \frac{\exp(\langle f_\theta(x_t, t), \text{EMB}(w) \rangle)}{\sum_{w'} \exp(\langle f_\theta(x_t, t), \text{EMB}(w') \rangle)}$$

as a cross-entropy with the true token *w*, or use the distance-based anchor MSE L_{anchor} above. The key is that this additional loss is "rounding-aware" and penalizes fractional embedding predictions before the sampling stage.

3. Integration Strategy

The new term(s) are added **on top of** the original training objective. Concretely, if L_{base} denotes the original Diffusion-LM loss (diffusion MSE plus existing embedding consistency and cross-entropy terms [1]), we optimize

$$L_{\text{total}} = L_{\text{base}} + \lambda L_{\text{new}}$$

where L_{new} is the proposed rounding-aware loss (e.g. L_{anchor} or L_{quant}) and λ is a hyperparameter weighting its influence. In practice, we will tune λ on validation control tasks: for example testing values such as {0,0.01,0.1,1.0} to find the best balance. A small λ checks if the new term can improve rounding without degrading fluency, while a large λ heavily biases outputs toward embeddings. Importantly, this addition is **only during training**; the sampling (generation) process remains exactly the same as the baseline Diffusion-LM. That is, we do *not* alter the forward or reverse diffusion steps or use any extra decoding-time tricks beyond the standard clamping (though ideally clamping will become less necessary if x_0 already lies near a valid code). The goal is to bake the rounding commitment into training, so that at inference time the model naturally produces token embeddings.

- **Objective:** $L_{\text{total}} = L_{\text{diffusion}} + L_{\text{rounding}} + \lambda L_{\text{new}}$, where L_{new} (anchor or quantization loss) is weighted by λ.
- **Hyperparameter tuning:** Select λ by grid search on held-out dev data (for example optimizing overall control-task success). We may also tune any internal scale of the new term if needed.

- **No changes to sampling:** During generation, we **do not** modify the sampling algorithm or use additional heuristics; all improvements must come from better training. Clamping to the nearest embedding is still applied as in Li et al. (2022), but we hypothesize that with the new loss it will be applied less often (i.e. the model will already commit).

4. Experimental Setup

We follow the original Diffusion-LM experimental protocol for training and evaluation, to enable direct comparison.

- **Datasets:** Train and evaluate on the same datasets used by Li et al.: the *E2E* NLG dataset and the *ROCStories* corpus [1]. E2E consists of ~50K restaurant descriptions labeled by fields (food, price, etc.) [1], and ROCStories is a collection of 5-sentence commonsense stories (larger, more diverse).
- **Model architecture:** Use the public Diffusion-LM code (GitHub release of Li et al.) as the base. The model is a Transformer diffusion model with roughly 80M parameters. We set embedding dimension d=16 for E2E and d=128 for ROCStories (as in the original paper) [1]. All other hyperparameters (number of layers, noise schedule, etc.) are kept the same as in the Li et al. implementation.
- **Diffusion schedule:** Use a forward noise schedule similar to Li et al. (we may replicate their "√ schedule" from Appendix A). For consistency, we use 200 diffusion steps for E2E and 2000 steps for ROCStories during training/decoding [1]. (At decoding time, Diffusion-LM uses these full trajectories; any acceleration tricks such as step skipping are applied equally to baseline and new model.)
- **Optimization:** Following Li et al., we train with AdaGrad on the diffusion objective. (Li et al. tuned learning rates from 0.05–0.2 for AdaGrad [1]; we will similarly tune the optimizer learning rate and any weight decay as needed.) We train until convergence or early stopping on validation perplexity, ensuring both models see the same number of updates. Training is done on GPU(s) as in the baseline (e.g. using an NVIDIA A100; Li et al. report ~3 days on one GPU for similar experiments [1]).
- **Evaluation tasks:** We evaluate control performance on the same six tasks from Li et al. (2022) [1]. These tasks are:
    - *Semantic content* (ensure certain slot values appear exactly)
    - *POS tags* (match a target POS tag sequence)
    - *Syntax tree* (match a target constituency parse, measured by F1)
    - *Syntax spans* (enforce a particular labeled span within the parse)
    - *Length* (generate within ±2 of a target length)[1]

- *Infilling* (generate a sentence connecting left/right contexts; evaluated via standard GenIE metrics) [1].
  For each task we compute the **success rate** according to the criteria in Li et al. (e.g. exact-match or F1) [1]. In addition, we measure fluency of the generated outputs by the **lm-score** metric: the perplexity of the generated text under a fixed pretrained GPT-2 (lower is better) [1]. Both success rate and lm-score will be reported and compared between the baseline Diffusion-LM and the model with the new loss.

5. Ablation Plan

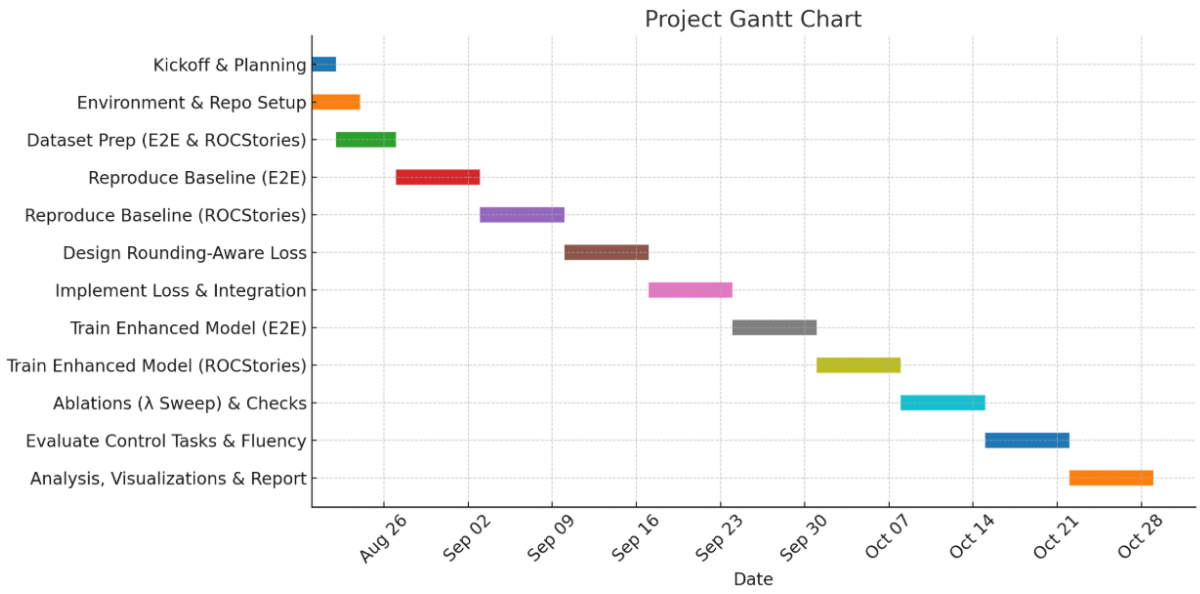
To isolate the effect of the new loss, we will perform controlled ablation experiments:

- **Baseline vs. enhanced:** Compare the original Diffusion-LM (no new loss) to the model trained with L_{new} (using a default λ). This directly measures whether adding the rounding-aware term improves control accuracy or fluency.
- **Varying lambda:** Train multiple variants with different λ (e.g. λ=0.0,0.01,0.1,1.0). Smaller λ tests whether a slight encouragement helps, while larger λ tests how strong the regularization can be before harming language quality. For each λ, record control success rates, lm-score, and also **commitment metrics** (e.g. fraction of outputs where the predicted x_0 is within an ℓ_2 threshold of some vocabulary embedding).
- **Metric trade-offs:** We expect trade-offs: a higher-weight loss may improve control success (better constraint satisfaction) but could degrade fluency (higher perplexity) if it distorts the model's output space. By plotting success rate vs. lm-score for each λ, we can identify Pareto-optimal settings. We will also check whether the new loss reduces reliance on clamping: for example, we can report how often clamping changes the sample vs. original predictions.

Throughout, we keep all other factors fixed (same random seeds, data order, decoding steps) to ensure fair comparison. These ablations will show whether the proposed loss truly aids "token commitment" (by observing how often x_0 lands near a valid embedding) and how it affects the standard control and fluency metrics on the E2E and ROCStories tasks.

Our baseline description and evaluation protocol follow Li et al.2022) [1]. The idea of adding an anchor or quantization loss is motivated by Gao et al. (2024) [10] and vector-quantized VAE techniques (van den Oord et al., 2017) [12], both of which emphasize pulling model outputs toward discrete embeddings. All experiments will build on the open-source Diffusion-LM implementation to ensure consistency with the original results.

Project Timeline



References

[1] Li, X., Thickstun, J., Gulrajani, I., Liang, P. S., & Hashimoto, T. B. (2022). Diffusion-lm improves controllable text generation. Advances in neural information processing systems, 35, 4328-4343.

[2] Li, Y., Zhou, K., Zhao, W. X., & Wen, J. R. (2023). Diffusion models for non-autoregressive text generation: A survey. arXiv preprint arXiv:2303.06574.

[3] Understanding Vector Quantization in VQ-VAE

[4] Jang, E., Gu, S., & Poole, B. (2016). Categorical Reparameterization with Gumbel-Softmax. *ArXiv*. https://arxiv.org/abs/1611.01144

[5] Nosaka, R., & Matsuzaki, T. (2025, July). Timestep Embeddings Trigger Collapse in Diffusion Text Generation. In Proceedings of the 29th Conference on Computational Natural Language Learning (pp. 397-406).

[6] Kaiser, L., Bengio, S., Roy, A., Vaswani, A., Parmar, N., Uszkoreit, J., & Shazeer, N. (2018, July). Fast decoding in sequence models using discrete latent variables. In International Conference on Machine Learning (pp. 2390-2399). PMLR.

[7] Mercatali, G., & Freitas, A. (2021). Disentangling generative factors in natural language with discrete variational autoencoders. arXiv preprint arXiv:2109.07169.

[8] Dathathri, S., Madotto, A., Lan, J., Hung, J., Frank, E., Molino, P., Yosinski, J., & Liu, R. (2019). Plug and Play Language Models: A Simple Approach to Controlled Text Generation. *ArXiv*. https://arxiv.org/abs/1912.02164

[9] Zhao, Q., Peng, B., Wang, S., Liu, H., Che, W., & Liu, T. (2021). *Disentangling Generative Factors in Natural Language with Discrete Variational Autoencoders*. In *Findings of EMNLP 2021*, pages 3500–3516.

[10] Gao, Z., Guo, J., Tan, X., Zhu, Y., Zhang, F., Bian, J., & Xu, L. (2024). *Empowering Diffusion Models on the Embedding Space for Text Generation*. In *Proceedings of NAACL-HLT 2024* (Long Papers), pages 4664–4683.

[11] Feng, G., Geng, Y., Guan, J., Wu, W., Wang, L., & He, D. (2025). Theoretical benefit and limitation of diffusion language model. arXiv preprint arXiv:2502.09622.

[12] Oord, A. V., Vinyals, O., & Kavukcuoglu, K. (2017). Neural Discrete Representation Learning. *ArXiv*. https://arxiv.org/abs/1711.00937