On Encoding Spaces for Continuous Diffusion Language Models

Anonymous ACL submission

Abstract

Continuous generative models have been highly successful on continuous data, but applying them to language requires choosing an encoding space for discrete tokens. We show that encoding plays a fundamentally different role in continuous models than in discrete models: beyond serving as a neural representation, it has an additional probability-modeling role by inducing the continuous target distribution. Through controlled experiments, we show that this role leads encoding changes to affect discrete and continuous models in qualitatively different ways. We then study what makes a good encoding space for continuous language modeling and identify encoding geometry and encoding dimension as two key factors. These insights lead to improved real-vocabulary continuous language modeling by modifying pretrained T5 embeddings through dimensionality reduction and geometry-aware regularization. Our results suggest that encoding-space design is a central modeling problem for continuous language models.

1 Introduction

Continuous generative models, especially diffusion-based models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2020), have become a dominant paradigm for modeling high-dimensional continuous data such as images, audio, and video. Their success has naturally motivated growing interest in applying continuous generative modeling to language, a domain that is still largely dominated by discrete probabilistic models such as autoregressive language models (Achiam et al., 2023; DeepSeek-AI, 2025) and discrete diffusion models (Austin et al., 2021; Nie et al., 2025). Unlike discrete models, however, continuous generative models must define probability distributions over a real-valued encoding space. Existing continuous language models often reuse data representations inherited from discrete language models, such as one-hot vectors or pretrained token embeddings, without explicitly examining whether these representations are suitable for continuous models. This leads to a basic but underexplored question for language modeling: are representations designed for discrete language models also good encoding spaces for continuous probabilistic modeling?

In this work, we study this question by decomposing the role of encoding in discrete and continuous generative models. We use *encoding* to refer to the function that maps discrete data, such as tokens, into numerical vectors. Our key observation in Section 3 is that encoding plays fundamentally different roles in the two settings. In discrete generative models, the probability distribution is defined directly on the token space; the encoding mainly serves as a neural representation through which the model receives token identities. In continuous generative models, by contrast, the encoding has a dual role: it is still a neural representation, but it also has an additional probability-modeling role because it induces the continuous distribution on which noising, denoising, and sampling are defined. Through controlled experiments in Section 3.2, we show that this distinction leads encoding changes to affect discrete and continuous models in qualitatively different ways.

Motivated by this observation, we revisit the encoding problem for continuous language models in Section 4 without relying on intuitions inherited from discrete representation learning. We conduct systematic experiments across vocabulary sizes and identify two important properties of continuous codebooks: encoding geometry and encoding dimension.

We show that poor encoding geometry can create geometric attractors that distort denoising behavior, while overly high-dimensional codebooks, such as one-hot encodings at large vocabulary sizes, are difficult for continuous models to learn despite their symmetry. These findings guide the design of im-

proved encodings for real-tokenizer settings. In particular, we apply dimensionality reduction and mild geometry-aware regularization to pretrained autoregressive token embeddings, yielding substantially better training behavior for continuous language models than directly using the original high-dimensional embedding space.

Our contributions are summarized as follows:

- We provide a theoretical decomposition of the role of encoding in generative modeling, showing that encoding has an additional probability-modeling role in continuous generative models beyond its representation role. Through controlled experiments, we further show that this role leads encoding changes to affect discrete and continuous models in qualitatively different ways.
- We systematically study what makes a good encoding space for continuous language models, and identify encoding geometry and encoding dimension as two key factors that shape continuous-model performance.
- We use these insights to improve real-vocabulary continuous language modeling by modifying pretrained token embeddings through dimensionality reduction and geometry-aware regularization, showing that encoding-space design is a practical lever for improving continuous language models.

2 Preliminaries

2.1 Discrete Language Models

Let Ω denote a finite vocabulary with $|\Omega| = K$, and n be the sequence length. A discrete language model defines a probability measure over the discrete sample space[1] Ω^n:

$$\mathbb{P} : \Omega^n \to [0, 1]. \tag{1}$$

Directly modeling this high-dimensional joint distribution is often intractable, and different discrete generative models therefore introduce different factorizations of $\mathbb{P}$. Autoregressive (AR) language models (Brown, 2020; Achiam et al., 2023) factorize the distribution through left-to-right next-token prediction, while masked diffusion models (MDMs) (Lou et al., 2024; Ou et al., 2025) define a continuous-time, discrete-state Markov process and learn to reverse it in arbitrary token directions. Although AR models and MDMs employ different probabilistic factorizations, both define the target probability measure directly on the discrete token space Ω^n.

This probabilistic formulation is independent of any particular vector representation of tokens. Nevertheless, neural network estimation still requires tokens to be represented as numerical vectors. We denote this token-level encoding function by

$$\mathcal{E} : \Omega \to \mathbb{R}^D. \tag{2}$$

For standard discrete language models, this encoding is often implemented as a one-hot vector followed by the first linear layer of the model, which is equivalent to a learned embedding lookup table.

2.2 Continuous Language Models

Continuous generative models, such as continuous diffusion models (CDMs) (Dhariwal and Nichol, 2021; Rombach et al., 2022), have been highly successful on continuous-modality data such as images and videos. Mathematically, they model a different object from discrete models: a continuous distribution p over a vector space $\mathbb{R}^{nD}$. In diffusion models, this is achieved by defining a forward noising process in the vector space and learning a reverse denoising process (Sohl-Dickstein et al., 2015; Ho et al., 2020).

Recent work has also explored whether continuous generative processes can be adapted to text. Continuous language models generate discrete text by running a generative process in a continuous encoding space and decoding the final continuous state back to tokens, typically through nearest-codebook decoding or a learned decoder. As in discrete language models, text tokens must first be mapped by $\mathcal{E}$ into numerical vectors before they can be processed by a neural network.

From the perspective of the encoding space, existing methods can be roughly categorized by their choice of $\mathcal{E}$. Some models use one-hot encodings $\mathcal{E}_{\text{hot}} : \Omega \to \mathbb{R}^K$, where each token is mapped to a coordinate vector in the vocabulary simplex (Richemond et al., 2022; Graves et al., 2023). Others use pretrained token embeddings $\mathcal{E}_{\text{emb}} : \Omega \to \mathbb{R}^D$, often taken from the input embedding table of a pretrained discrete language model (Li et al., 2022; Wu et al., 2023; Gong et al., 2022). A third line of work maps entire

[1] Strictly speaking, a probability measure is defined on a sigma-algebra over Ω^n. Since Ω is finite in our setting, this distinction does not affect our discussion.

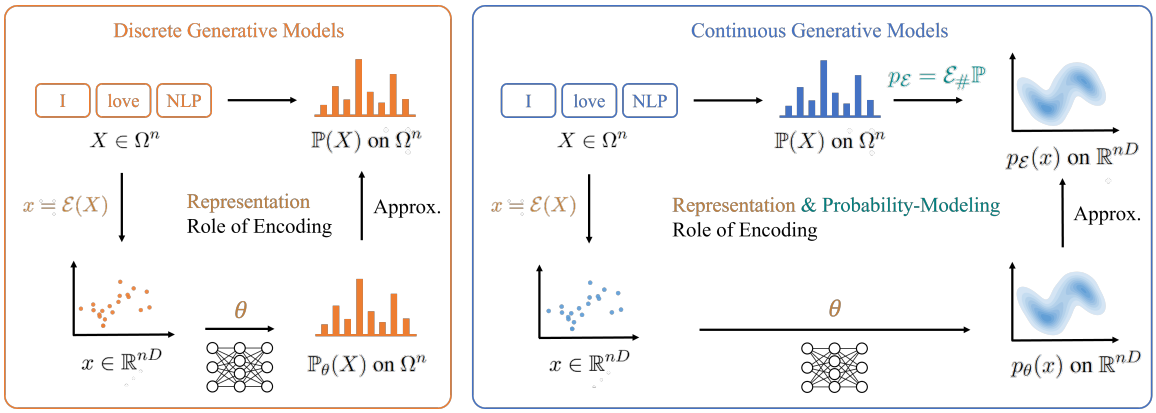

Figure 1: Encoding in discrete and continuous generative models. In discrete generative models, the data distribution $\mathbb{P}$ and the learned model distribution $\mathbb{P}_\theta$ are both defined on the token space Ω^n; the encoded vector $x = \mathcal{E}(X)$ mainly serves as a neural representation. In continuous generative models, the encoding function pushes the discrete distribution forward to an encoding-induced continuous target $p_\mathcal{E}$ on $\mathbb{R}^{nD}$, which is then approximated by p_θ. Thus, encoding has both a representation role and a probability-modeling role in continuous models.

sequences into contextual continuous representations $\mathcal{E}_{\text{ctx}} : \Omega^n \to \mathbb{R}^{nD}$ using a pretrained encoder (Zhou et al., 2025; Hu et al., 2026; Guo et al., 2026). These encoding choices are largely inherited from the representation-learning practice of autoregressive language models (Robinson et al., 2025). As we show in the following sections, however, encoding design requires separate consideration in continuous language models.

3 The Probability-Modeling Role of Encoding in Continuous Generative Models

Section 2 shows that both discrete and continuous language models require an encoding function to transform data into vector representations for neural networks. This apparent similarity hides a fundamental difference: in continuous generative models, encoding not only serves as a representation, but also participates in defining the probability modeling target itself. We refer to this additional role as the *probability-modeling role* of encoding. In this section, we first formalize this additional role theoretically and then use controlled experiments to show that it leads encoding changes to affect discrete and continuous models in qualitatively different ways. Figure 1 provides an intuitive illustration of this functional distinction.

3.1 Theoretical Analysis

In discrete generative models, the probability measure is defined directly on the token sample space Ω^n. The encoding function mainly serves as an input representation: it tells the neural network which token is present. Changing the input encoding changes how information is presented to the network and can affect optimization or representation learning. However, the discrete target $\mathbb{P}$ is defined on Ω^n without reference to any encoding.

For continuous generative models, the situation is different. The model does not directly approximate the discrete measure $\mathbb{P}$. Instead, it approximates a distribution over a real-valued space, where noising and denoising are defined in. This continuous modeling target does not exist until an encoding function is chosen. Given an encoding function $\mathcal{E} : \Omega^n \to \mathbb{R}^{nD}$, the discrete data measure is pushed forward to the encoding space:

$$p_\mathcal{E} = \mathcal{E}_\#\mathbb{P}. \tag{3}$$

Equivalently, for any measurable set $B \subset \mathbb{R}^{nD}$, the induced distribution assigns probability according to

$$p_\mathcal{E}(B) = \mathbb{P}\big(\{X \in \Omega^n : \mathcal{E}(X) \in B\}\big). \tag{4}$$

Here $p_\mathcal{E}$ denotes the encoding-induced distribution[2] that a continuous generative model is trained to approximate. Thus, $p_\mathcal{E}$ is not defined independently of the encoding: it is constructed from the discrete data distribution through $\mathcal{E}$.

Consequently, in continuous generative models, encoding has a dual role: it is both a neural representation and the mechanism that specifies the space where noising, denoising, and sampling are defined.

[2]For simplicity, we use $p_\mathcal{E}$ to denote both the encoding-induced continuous measure and the continuous distribution.

This distinction is central to our paper. In discrete models, changing the token encoding changes how information is presented to the neural network and can affect optimization or representation learning. However, it does not change the sample space or the probability measure being modeled. In continuous models, changing the encoding changes the induced target $p_{\mathcal{E}}$ itself, and therefore changes the probability modeling problem. Appendix A further clarifies this theoretical analysis by distinguishing several uses of one-hot vectors in discrete and continuous models.

3.2 Empirical Evidence

Having separated the representation role and the probability-modeling role of encoding in Section 3.1, we now ask whether this theoretical distinction leads to observable differences in practice. We construct controlled experiments in which the vocabulary size and model architecture are aligned across model families, and then compare how different encodings affect AR, MDM, and CDM.

We use a vocabulary of size $K = 27$. For text, we use character-level Text8 (Mahoney, 2011), consisting of the 26 English letters plus the space character. For images, we use CIFAR-10 (Krizhevsky, 2009) after converting images to grayscale and discretizing pixel intensities into 27 bins. Both modalities are therefore represented as sequences over a 27-token vocabulary. Each experiment uses $N = 5000$ training examples. The AR, MDM, and CDM models use aligned Transformer backbones, and CDM refers to a DDPM (Ho et al., 2020) model implemented on the specified codebook. More experimental details are given in Appendix B.1.

We compare three encodings. The first is a one-hot encoding, which maps each token to a 27-dimensional one-hot vector. The second is a scalar encoding, which maps each token to a one-dimensional scalar. For images, the scalar order follows grayscale intensity; for text, it follows alphabetical order. The third is a SCALAR-SHUFFLE encoding for images. It uses the same one-dimensional scalar space but applies a deterministic permutation that places visually distant grayscale bins close to each other in scalar coordinate. For CDM, all codebooks are normalized to $[-1, 1]$, and this normalized codebook is the space on which noising, denoising, and nearest-codebook decoding are performed. The construction of SCALAR-SHUFFLE and additional image-modality shuffle results are provided in Appendix C.1.

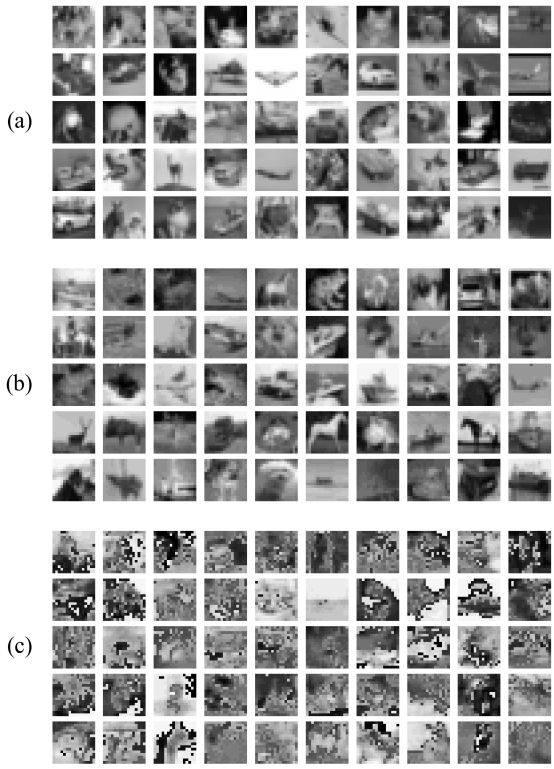

Figure 2: Samples from the controlled grayscale CIFAR-10 experiment in Section 3. (a) Validation images. (b) Samples generated by the AR model with the scalar-shuffle encoding. (c) Samples generated by the CDM model with the scalar-shuffle encoding.

Table 1 reports the results. The behavior of discrete and continuous models is markedly different. On Text8, AR and MDM remain relatively stable when replacing one-hot inputs with scalar inputs, whereas CDM suffers a substantial drop in spelling accuracy (SA). On grayscale CIFAR-10, the natural scalar encoding is competitive for CDM because the scalar order matches the semantic order of pixel intensity. However, once this order is shuffled, CDM generation collapses. In contrast, AR and MDM remain much less affected by the same scalar-shuffle perturbation. Figure 2 visualizes this contrast on generated samples.

These results support our central hypothesis. In the controlled setting above, discrete models can still learn the required token prediction tasks even when the raw input representation is substantially restricted. CDM, however, is directly affected because its continuous target distribution is induced by the codebook. The same scalar shuffle that is nearly harmless for discrete models becomes catastrophic for CDM, because it changes the encoding-induced distribution that the continuous model is

CIFAR-10 FID (↓)			
Encoding	AR	MDM	CDM
One-hot	17.00	21.43	24.11
Scalar	17.57	22.31	21.64
Scalar-shuffle	17.36	22.51	111.74
Text8 SA (↑)			
Encoding	AR	MDM	CDM
One-hot	89.29	71.82	71.73
Scalar	88.90	67.61	56.15

Table 1: Controlled comparison between discrete and continuous models under different encodings. In this controlled setting, AR and MDM remain relatively stable under these perturbations, while CDM is substantially more affected because the same encoding also participates in defining the continuous target distribution. Metrics are FID (Heusel et al., 2017) for CIFAR-10 and spelling accuracy (SA) for Text8.

trained to approximate.

These results also suggest that continuous language models may require codebook properties that are not emphasized in discrete representation learning. However, as discussed in Section 2, existing continuous language models often inherit one-hot vectors or token embeddings from autoregressive language modeling. Since these representations were not designed to define continuous target distributions, it is unclear whether they provide suitable codebooks for continuous probability modeling. This motivates the central question of Section 4: what properties should a good encoding space have for continuous language models?

4 What Makes a Good Encoding Space for Continuous Language Models?

Section 3 shows that encoding plays a fundamentally different probability-modeling role in continuous generative models. This raises a natural question: what properties should a good encoding space have for continuous language modeling?

We investigate this question empirically, moving from controlled small-vocabulary experiments to larger vocabularies and finally to a real-tokenizer setting. The main message is that two factors are especially important: encoding geometry and encoding dimension. A good codebook should avoid degenerate encoding geometry, but it should also avoid becoming overly high-dimensional and sparse. In real-vocabulary settings, these two factors must also be balanced with the semantic structure inherited from pretrained token embeddings.

4.1 Encoding Geometry

We first return to the $K = 27$ Text8 setting. Section 3 shows that scalar encoding performs substantially worse than one-hot encoding for CDM. We hypothesize that one reason is its encoding geometry. In a scalar codebook, different tokens occupy geometrically asymmetric positions on a line segment, whereas in a one-hot codebook all tokens are placed symmetrically. To test whether reducing this geometric asymmetry can improve continuous modeling without moving to a high-dimensional one-hot space, we construct a circular codebook.

In the circular codebook, denoted CIRCULAR-ALPHABET in Table 2, the 27 tokens are placed uniformly on a unit circle in alphabetical order. This changes the encoding geometry from a line segment to a closed curve while preserving a simple token order. As shown in Table 2, this modification leads to a significant improvement over scalar encoding. In contrast, the two control encodings SCALAR-ZERO-PAD and SCALAR-COPY do not yield comparable gains. SCALAR-ZERO-PAD maps each scalar value s to $(s, 0)$, while SCALAR-COPY maps it to (s, s). Both variants increase the dimension but preserve the intrinsic line geometry. These controls indicate that the improvement is not simply due to moving from one to two dimensions; rather, it comes from changing the encoding geometry.

We also test the effect of semantic ordering. We reorder tokens using frequency or statistical similarity estimated from the corpus, including the SCALAR-STAT, SCALAR-FREQ, and CIRCULAR-STAT variants reported in Table 2. These variants yield little improvement over the corresponding alphabetical orders, suggesting that in this small-vocabulary character setting, token-level semantic structure is relatively weak and changing the token-to-position assignment has limited effect.

To understand why circular geometry helps, we visualize the direct denoising behavior of the trained model. We take validation examples, add noise to a fixed level, let the model predict the clean tokens in one step, and then count how often each ground-truth token is predicted as each output token. Figure 3 compares the resulting confusion matrices for the scalar and circular codebooks.

The scalar codebook in Figure 3(a) exhibits a characteristic failure pattern: the two endpoint to-

Encoding	Dimension	SA (↑)
Scalar	1	56.15
Scalar-zero-pad	2	58.54
Scalar-copy	2	58.68
Circular-alphabet	2	64.03
Scalar-freq	1	56.16
Scalar-stat	1	57.29
Circular-stat	2	63.36
One-hot	27	71.73

Table 2: Small-vocabulary geometry experiments on Text8. SCALAR-ZERO-PAD and SCALAR-COPY embed the scalar line into two dimensions without changing its intrinsic line geometry. CIRCULAR-ALPHABET closes the line into a circle and substantially improves SA. Reordering tokens by simple statistics provides only marginal gains in this small-vocabulary setting.

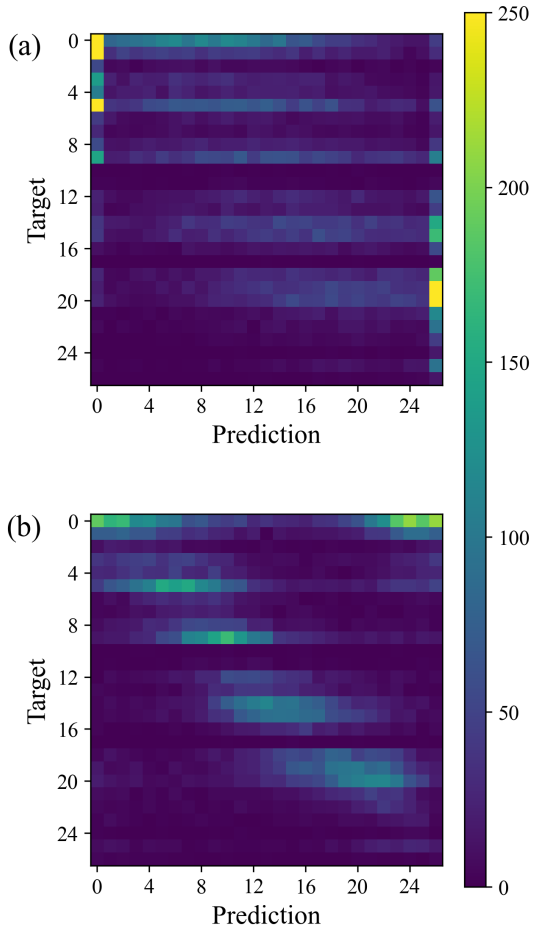

Figure 3: Direct denoising confusion matrices on Text8. Rows correspond to ground-truth tokens and columns correspond to predicted tokens. (a) Scalar codebook. The two endpoint tokens attract predictions, forming bright vertical stripes. (b) Circular codebook. Closing the line into a circle removes the endpoint asymmetry and yields a more diagonal denoising pattern.

kens produce bright vertical stripes, indicating that many different tokens are predicted as these endpoints. We observe the same phenomenon under other scalar orderings: whichever tokens are placed at the two ends of the line tend to attract predictions. The circular codebook in Figure 3(b) largely removes this asymmetry. Since the codebook is closed and every point has a similar local neighborhood, no token is geometrically privileged, and the confusion matrix becomes much closer to diagonal.

This experiment suggests that continuous codebooks should avoid strong geometric imbalance. One-hot encoding also has this advantage: it is highly symmetric and contains no line endpoints. However, symmetry alone is not sufficient. As we show next, one-hot becomes increasingly unsuitable as the vocabulary size grows due to its high encoding dimension.

4.2 Encoding Dimension

The circular experiment shows that balanced encoding geometry is important. However, high symmetry alone does not guarantee a good continuous codebook. One-hot encodings are highly symmetric and treat all tokens equally. Yet, as we show below, their performance degrades rapidly as the vocabulary size increases.

We study this issue using vocabularies of different sizes. For each vocabulary size K, we construct sphere codebooks of different dimensions D. Specifically, we randomly place token vectors on a D-dimensional sphere and use them as the codebook. This construction gives a family of geometrically balanced dense codebooks, allowing us to vary the dimension while avoiding the degenerate encoding geometry of scalar encodings.

Figure 4 shows the performance as a function of the relative dimension D/K. Across vocabulary sizes, we observe a clear inverted-U trend. Very low-dimensional codebooks lack sufficient geometric capacity, while overly high-dimensional codebooks become difficult for continuous models to learn. Importantly, the best relative dimension D/K decreases as K grows. For example, when $K = 1024$, the 1024-dimensional one-hot codebook performs poorly, while a 32-dimensional sphere codebook achieves strong performance. Thus, the best-performing dimension can be much smaller than the vocabulary size.

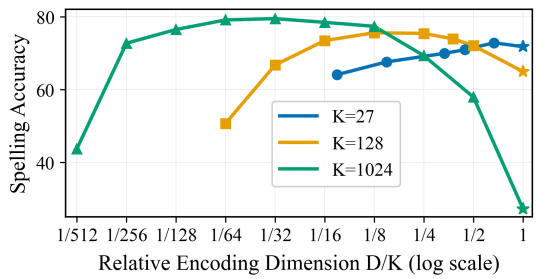

Figure 4: Spelling accuracy of sphere codebooks with different relative dimensions D/K. Each curve corresponds to a vocabulary size K, and one-hot is shown as the high-dimensional endpoint $D = K$. Raw data of the experiments are reported in Appendix D.1.

This dimension trend highlights a key difference between continuous and discrete models. In an autoregressive language model, a one-hot input is simply a way to index a learnable embedding table. The first linear layer can transform it into any internal representation useful for next-token prediction. In a continuous model, however, the codebook is the space where noise is injected and final nearest-codebook decoding is performed. The model cannot freely reinterpret this space without changing the probabilistic problem itself.

From this perspective, one-hot is not merely a high-dimensional representation; it defines a high-dimensional sparse codebook. Although symmetric, this codebook becomes difficult to model when K is large. Lower-dimensional sphere codebooks provide denser continuous modeling spaces while maintaining balanced geometry, explaining why the best dimension is an intermediate value rather than either extreme. Further discussions on the encoding dimension and additional experiments are provided in Appendix C.2.

We also train autoregressive models with the same families of encodings. Unlike continuous models, AR models do not show comparable gains from sphere codebooks over one-hot inputs. This again supports our main claim: the encoding problem is different in continuous models because the encoding space is part of the probabilistic model, rather than only an input representation.

The fact that the optimal relative dimension D/K decreases as the vocabulary size grows motivates the real-vocabulary experiments in Section 4.3. If continuous models prefer much lower-dimensional codebooks, then directly using high-dimensional pretrained autoregressive embeddings may be suboptimal.

4.3 Real-Vocabulary Experiments

Experiments in Section 4.1 and 4.2 suggest that, in large-vocabulary settings, reducing the encoding dimension and improving encoding geometry can substantially benefit continuous diffusion models. This motivates us to modify real pretrained token embedding tables through dimensionality reduction and geometry-aware regularization.

We conduct real-vocabulary experiments on OpenWebText (Gokaslan and Cohen, 2019) using the T5 tokenizer (Raffel et al., 2020), whose vocabulary size is $K = 32128$. We use Bayesian Flow Networks (BFN) (Graves et al., 2023) as the continuous diffusion model because it achieves substantially better generation quality than DDPM in this real-vocabulary setting. The model has approximately 170M parameters, matching the scale of the original BFN architecture. The original BFN formulation is defined for one-hot categorical states, and the BFN one-hot row in Table 3 corresponds to this original setting. To instantiate BFN with arbitrary codebooks, we use the SDE interpretation of BFN dynamics after a small modification to the sampling rule (Xue et al., 2024). Additional training details are provided in Appendix B.2.

Our baseline codebook is the T5-small input embedding table, which has dimension $D = 512$. We compare this baseline with PCA-reduced codebooks, whose results are reported in Table 3. Starting from the normalized T5 embedding table, we project the token vectors to $D \in \{64, 128\}$ principal components. These PCA codebooks retain the semantics of the pretrained token embeddings while reducing the encoding dimension.

PCA reduction can also change the distribution of vector norms. To control extreme radius variation, we apply a simple radius shrinkage operation. Let $p_i^{(D)}$ be the PCA-reduced vector for token i, $r_i = \|p_i^{(D)}\|$, and $\bar{r} = \frac{1}{K}\sum_{i=1}^{K} r_i$. We define

$$r_i^{(\alpha)} = \alpha r_i + (1-\alpha)\bar{r}. \tag{5}$$

The shrinkage codebook is then

$$c_i^{(\alpha)} = r_i^{(\alpha)} \frac{p_i^{(D)}}{\|p_i^{(D)}\|}. \tag{6}$$

Here $\alpha = 1$ recovers the raw PCA codebook, while $\alpha = 0$ projects all tokens to a sphere of radius $\bar{r}$. Intermediate values partially preserve radial information while reducing extreme norm variation. Since pretrained embedding norms may

contain useful frequency or semantic information, overly strong shrinkage can also remove useful structure. Thus, shrinkage introduces a trade-off between improving encoding geometry and preserving pretrained embedding information.

Table 3 reports the results. We evaluate generated text using generated perplexity (GenPPL), computed by a GPT2-Large (Radford et al., 2019) evaluator on samples of length 256. PCA reduction is highly effective: both 128-dimensional and 64-dimensional PCA codebooks achieve substantially lower GenPPL than the original 512-dimensional T5 embedding under the same BFN setup. This supports the conclusion from Section 4.2 that reducing the dimension of the continuous modeling space can benefit continuous language models. Additional samples from PCA-reduced codebooks are provided in Appendix D.2.

The effect of radius shrinkage is more nuanced. For the 64-dimensional PCA codebook, shrinkage improves GenPPL, suggesting that radius variation becomes a more important geometric issue after stronger dimensionality reduction. In contrast, shrinkage does not improve the 128-dimensional or 512-dimensional codebooks in our current runs. One possible explanation is that these higher-dimensional codebooks already retain a reasonably well-conditioned geometry, so shrinking the radii mainly removes useful pretrained radial information rather than providing additional geometric benefit. This pattern suggests that shrinkage should not be viewed as uniformly beneficial; it is most useful when dimensionality reduction creates norm imbalance that harms the continuous modeling space.

These results extend the controlled experiments to a real-tokenizer setting. They show that high-dimensional token embeddings inherited from autoregressive models are not necessarily good codebooks for continuous models. Lower-dimensional PCA codebooks provide more suitable continuous modeling spaces, confirming the importance of encoding dimension observed in Section 4.2. Radius shrinkage further improves the 64-dimensional PCA codebook, but it does not consistently help at higher dimensions, indicating that radius regularization is useful only when it corrects a genuine geometric imbalance. Overall, these findings show how the distinction between the representation role and the probability-modeling role of encoding leads to concrete codebook improvements for continuous language modeling.

Model	Encoding	GenPPL (↓)
AR	One-hot	184.39
BFN	One-hot	2825.75
	T5-512-raw	524.95
	T5-512-α0.9	559.17
	T5-512-α0.5	527.24
	T5-PCA128	323.43
	T5-PCA128-α0.9	455.74
	T5-PCA128-α0.5	442.61
	T5-PCA64	471.05
	T5-PCA64-α0.9	459.86
	T5-PCA64-α0.5	415.11

Table 3: Real-vocabulary experiments on OpenWebText with the T5 tokenizer. GenPPL is computed using a GPT2-Large (Radford et al., 2019) evaluator on generated text of length 256. The AR baseline uses one-hot encoding and the same Transformer architecture and parameter scale as BFN experiments. For BFN, PCA reduction substantially improves over the 512-dimensional T5 embedding and the original one-hot encoding, while radius shrinkage is helpful mainly for the 64-dimensional PCA codebook.

5 Conclusion

This paper studies the role of encoding in continuous language models. We show that, unlike in discrete generative models where token encodings mainly serve as neural representations, encodings in continuous generative models also participate in defining the probability modeling target. This additional probability-modeling role makes continuous models respond to encoding design in a fundamentally different way. Through controlled experiments, we demonstrate that encoding changes that have relatively small effects on discrete models can substantially affect continuous models. We further show that good continuous codebooks depend on properties that are not emphasized in standard discrete representation learning, including encoding geometry and encoding dimension. Finally, we use these insights to improve real-vocabulary continuous language modeling by modifying pretrained token embeddings through dimensionality reduction and mild geometry-aware regularization. Our results suggest that continuous language modeling requires a distinct theory of encoding-space design, rather than a direct reuse of representations developed for discrete language models.

Limitations

This work provides a systematic analysis of encoding spaces for continuous language models and shows that modifying pretrained autoregressive token embeddings can substantially improve continuous probabilistic modeling. However, the encoding properties studied in this paper, such as encoding geometry and encoding dimension, are still characterized mostly through empirical diagnostics rather than a complete mathematical theory. Future work may develop more rigorous tools for analyzing encoding-induced probability modeling targets, which could lead to more systematic principles for constructing or learning codebooks. Another important direction is to study whether different families of continuous generative models, such as DDPM-style diffusion models, BFN-style models, and other flow- or SDE-based models, require different encoding properties.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. 2021. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993.

Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948.

Prafulla Dhariwal and Alexander Nichol. 2021. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794.

Philip Gage. 1994. A new algorithm for data compression. *The C Users Journal archive*, 12:23–38.

Aaron Gokaslan and Vanya Cohen. 2019. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus.

Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and LingPeng Kong. 2022. Diffuseq: Sequence to sequence text generation with diffusion models. *arXiv preprint arXiv:2210.08933*.

Alex Graves, Rupesh Kumar Srivastava, Timothy Atkinson, and Faustino J. Gomez. 2023. Bayesian flow networks. *CoRR*, abs/2308.07037.

Hongcan Guo, Qinyu Zhao, Yian Zhao, Shen Nie, Rui Zhu, Qiushan Guo, Feng Wang, Tao Yang, Hengshuang Zhao, Guoqiang Wei, and Yan Zeng. 2026. Continuous latent diffusion language model. *Preprint*, arXiv:2605.06548.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6626–6637.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851.

Keya Hu, Linlu Qiu, Yiyang Lu, Hanhong Zhao, Tianhong Li, Yoon Kim, Jacob Andreas, and Kaiming He. 2026. Elf: Embedded language flows. *Preprint*, arXiv:2605.10938.

Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images.

Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. 2022. Diffusion-lm improves controllable text generation. *Advances in Neural Information Processing Systems*, 35:4328–4343.

Aaron Lou, Chenlin Meng, and Stefano Ermon. 2024. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

M Mahoney. 2011. Large text compression benchmark.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025. Large language diffusion models. *Preprint*, arXiv:2502.09992.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. 2025. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. In *ICLR*.

William Peebles and Saining Xie. 2023. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and 1 others. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Pierre H. Richemond, Sander Dieleman, and Arnaud Doucet. 2022. Categorical sdes with simplex diffusion. *Preprint*, arXiv:2210.14784.

Michael Robinson, Sourya Dey, and Tony Chiang. 2025. Token embeddings violate the manifold hypothesis. *Preprint*, arXiv:2504.01002.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 10674–10685. IEEE.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. 2015. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2020. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*.

Tong Wu, Zhihao Fan, Xiao Liu, Yeyun Gong, Yelong Shen, Jian Jiao, Hai-Tao Zheng, Juntao Li, Zhongyu Wei, Jian Guo, Nan Duan, and Weizhu Chen. 2023. Ar-diffusion: Auto-regressive diffusion model for text generation. *Preprint*, arXiv:2305.09515.

Tongda Xu, Mingwei He, Shady Abu-Hussein, Jose Miguel Hernandez-Lobato, Chunhang Zheng, Kai Zhao, Chao Zhou, Ya-Qin Zhang, and Yan Wang. 2026. Making reconstruction fid predictive of diffusion generation fid. *Preprint*, arXiv:2603.05630.

Kaiwen Xue, Yuhao Zhou, Shen Nie, Xu Min, Xiaolu Zhang, Jun Zhou, and Chongxuan Li. 2024. Unifying bayesian flow networks and diffusion models through stochastic differential equations. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Zekai Zhang, Xiao Li, Xiang Li, Lianghe Shi, Meng Wu, Molei Tao, and Qing Qu. 2025. Generalization of diffusion models arises with a balanced representation space. *CoRR*, abs/2512.20963.

Cai Zhou, Chenxiao Yang, Yi Hu, Chenyu Wang, Chubin Zhang, Muhan Zhang, Lester Mackey, Tommi Jaakkola, Stephen Bates, and Dinghuai Zhang. 2025. Coevolutionary continuous discrete diffusion: Make your diffusion language model a latent reasoner. *Preprint*, arXiv:2510.03206.

A Distinguishing Three Notions of One-Hot

In this section, we further clarify our theoretical framework by distinguishing three different uses of one-hot vectors. Although these objects have the same vector form, they play different roles in probabilistic modeling. The three notions are: (1) one-hot vectors as inputs to an autoregressive model; (2) one-hot vectors as the codebook of a continuous diffusion model; and (3) one-hot vectors as ground-truth probability targets in the output loss of an autoregressive model.

The distinction between (1) and (2) is discussed throughout the main text. In an autoregressive model, the input one-hot vector serves as a representation of token identity. It is passed through the first linear layer, which is equivalent to a learned embedding lookup table. The probability measure being modeled is still defined directly on the discrete token space. In contrast, when a continuous diffusion model uses one-hot vectors as its codebook, the one-hot space becomes the continuous space in which noising, denoising, and sampling are performed. Thus, the same vector representation now participates in defining the continuous probability-modeling target.

It is also important to distinguish (1) from (3). In an autoregressive model, the one-hot ground truth used in the cross-entropy loss has a direct probabilistic meaning: it represents the empirical target distribution over the vocabulary for a next-token prediction problem. The cross-entropy loss is directly tied to maximum likelihood estimation. By contrast, the one-hot input to the model does not itself represent a probability distribution. This explains why changing the input encoding can have limited effect on an autoregressive model within a broad range, whereas the one-hot target in the output loss is an essential part of the model's probabilistic formulation.

Finally, (2) and (3) are also different. The one-hot target in cross-entropy is itself a probability vector. The one-hot codebook used by a continuous diffusion model is related to probability modeling, but it is not generally a probability vector. Its role is to provide a continuous coordinate system in which a density or stochastic process is defined. Therefore, the output of a one-hot continuous diffusion model should not, in general, be interpreted as logits in the same way as the output of an autoregressive model. For standard continuous diffusion models, nearest-codebook decoding is the appropriate decoding rule.

Model	Decoding Rule	SA (↑)
DDPM	Nearest Neighbor	71.43
	Softmax Sampling	8.23
BFN	Nearest Neighbor	80.08
	Softmax Sampling	79.35

Table 4: Comparison of two decoding strategies under one-hot encoding. DDPM and BFN use the same one-hot codebook, but softmax-based probabilistic sampling is only effective for BFN because of its categorical belief interpretation. Experimental settings follow Appendix B.1.

There are special cases where the distinction becomes less sharp. For example, in BFNs (Graves et al., 2023), the original formulation admits a prior-posterior interpretation over categorical beliefs, while the same process can also be viewed as an SDE in a continuous state space (Xue et al., 2024). In this case, one-hot-like vectors can simultaneously serve as a codebook and as a belief representation. This explains why probability-style sampling can still work well for BFN under one-hot encoding. Table 4 illustrates the difference between DDPM and BFN under two decoding rules.

Overall, the same one-hot vector form can play conceptually different roles. It can be an input representation, a probabilistic target, or a continuous codebook. Our main argument concerns the second case: when one-hot vectors are used as a continuous codebook, they define the modeling space, but they should not automatically be interpreted as probability vectors.

B Experimental Details

In this section, we provide additional details for the experimental settings used in the main text.

B.1 Experimental Settings for Section 3, Section 4.1, and Section 4.2

For the small-scale text experiments, we use the Text8 (Mahoney, 2011) dataset with character-level tokenization. The vocabulary consists of the 26 English letters plus the space character, so $K = 27$. We split the data into sequences of length $n = 256$ and use $N = 5000$ training sequences unless otherwise specified.

For the image experiments, we use CIFAR-10 (Krizhevsky, 2009). To align the image and text

settings as closely as possible, we first convert images to grayscale, downsample them to 16×16, and then discretize each grayscale pixel into 27 bins, indexed from 0 to 26. This again gives $K = 27$ and sequence length $n = 256$. We also use $N = 5000$ training examples for the aligned image experiments. Under this preprocessing, the image and text datasets have the same formal structure: a length-256 sequence over a 27-token vocabulary. All codebooks are normalized to $[-1, 1]$ before noising. The normalized codebook is the actual space on which noising, denoising, and nearest-codebook decoding are performed.

For the larger-vocabulary Text8 experiments in Section 4.2, we train byte-pair encoding (BPE) (Gage, 1994) tokenizers of different vocabulary sizes on the full Text8 corpus. We then retokenize the data and split it into sequences of 256 tokens. This setup allows us to study how codebook dimension affects continuous modeling as the vocabulary size grows.

All AR, DDPM, and MDM models use the same main Transformer backbone unless otherwise stated. The backbone has hidden dimension 768, 24 layers, and 12 attention heads, with approximately 170M parameters. The DDPM implementation follows a DiT-style architecture (Peebles and Xie, 2023), while the MDM implementation follows RADD (Ou et al., 2025). All results are reported after sufficient training, using the best validation or generation metric reached during training. The small-scale experiments can be completed on two NVIDIA H20 GPUs.

B.2 Experimental Settings for Section 4.3

For the real-vocabulary experiments in Section 4.3, we keep the same model scale as in Appendix B.1. We use the T5-small tokenizer, whose vocabulary size is $K = 32128$ (Raffel et al., 2020). The baseline codebook is the input embedding table of T5-small, with dimension $D = 512$. We emphasize that this is the static token embedding table used for lookup, not a contextual embedding produced by the T5 encoder.

We use OpenWebText (Gokaslan and Cohen, 2019) as the training corpus. After tokenization with the T5 tokenizer, we split the token stream into sequences of length $n = 256$. For these experiments, we use BFN rather than the DDPM model used in the smaller experiments, because we found BFN to be substantially more stable and effective in this real-vocabulary setting. Although the original BFN formulation is defined for one-hot categorical states, we use its SDE-equivalent interpretation to instantiate BFN over arbitrary codebooks, as discussed in the main text. All BFN experiments in Section 4.3 use batch size 512 and are trained for 2 epochs on OpenWebText. We use the AdamW optimizer with learning rate 5×10^{-4}.

C More Observations and Discussions

In this section, we provide additional observations that support the main claims but are not discussed in detail in the main text.

C.1 Shuffle Experiment

Although our main focus is the text modality, the aligned image experiments reveal informative phenomena about continuous codebooks. In Section 3, the scalar-shuffle experiment on grayscale CIFAR-10 shows that changing the token-to-coordinate mapping can severely affect continuous models while leaving discrete models much less affected.

Recall that CIFAR-10 images are converted to grayscale, downsampled to 16×16, and discretized into 27 bins. The natural scalar encoding maps grayscale bins in increasing intensity order to scalar coordinates $0, \ldots, 26$. In the scalar-shuffle experiment, we instead arrange the original grayscale bins in the following order:

$$13, 26, 0, 25, 1, 24, 2, 23, 3, 22, 4, 21, 5, 20, \\ 6, 19, 7, 18, 8, 17, 9, 16, 10, 15, 11, 14, 12. \quad (7)$$

We then map this sequence to scalar coordinates $0, \ldots, 26$. For example, grayscale bin 13 is encoded as 0, bin 26 as 1, bin 0 as 2, and so on. This permutation is designed to make neighboring scalar coordinates correspond to grayscale values that are visually far apart, thereby strongly disrupting the semantic locality of the continuous code space.

This perturbation dramatically harms CDM generation, while producing much weaker effects on AR and MDM. We also test global cyclic shifts of the grayscale order, such as cyclically shifting by five positions. These transformations change the absolute assignment of grayscale bins to scalar coordinates but largely preserve the local order relation. As shown in Table 5, such cyclic shifts have only mild effects compared with the scalar-shuffle construction.

These results suggest that local semantic relationships in the code space are particularly important for continuous models. A continuous diffusion model ultimately predicts vectors that must be

Model	Encoding	FID (↓)
CDM	Identical	21.64
	Reverse	22.45
	Cyclic-1	23.13
	Cyclic-5	25.41
	Cyclic-14	22.96

Table 5: Global cyclic shifts of the grayscale scalar order have only mild effects on CDM generation. This contrasts with the scalar-shuffle encoding in Section 3, which destroys local semantic order and severely degrades FID.

decoded back to discrete tokens, often through a nearest-codebook rule. When nearby code points correspond to semantically distant tokens, small continuous denoising errors can lead to large semantic errors after decoding. By contrast, global relabeling that preserves local order has a much smaller effect. This observation is also consistent with prior analyses suggesting that diffusion models often generalize through local interpolation in the data or latent space (Zhang et al., 2025; Xu et al., 2026): in our setting, meaningful interpolation requires local neighborhoods in the code space to correspond to local changes in the underlying data.

C.2 Hierarchical Encoding

In Section 4.2, we study how the dimension of sphere codebooks affects continuous language modeling. Scalar and one-hot encodings can be viewed as two extreme cases: scalar codebooks are very low-dimensional, while one-hot codebooks are maximally high-dimensional. We also evaluate a different family of intermediate-dimensional codebooks, which we call hierarchical encodings.

For a vocabulary of size K, a hierarchical codebook with dimension D partitions tokens into $D-1$ groups. The first $D-1$ coordinates represent the group identity using a one-hot or simplex-like group code, while the final coordinate is a scalar value that distinguishes tokens within the same group. This construction combines a coarse group structure with a local scalar coordinate.

The results are shown in Table 6. Similar to the sphere codebooks in Section 4.2, hierarchical encodings also show an inverted-U-like trend. Very low-dimensional hierarchical codebooks perform poorly, while increasing the number of groups substantially improves performance. However, their best dimensions are generally larger than those of sphere codebooks. This difference is consistent with their different geometries: sphere codebooks spread tokens across many directions more uniformly, whereas hierarchical codebooks concentrate variation along group identities and within-group scalar coordinates. As a result, hierarchical codebooks often need more nominal dimensions to reach strong performance.

This comparison further supports the main conclusion of Section 4.2. Dimension matters, but dimension alone does not determine the quality of a continuous codebook. The geometry induced by specific encodings is also important. Sphere codebooks tend to use the available dimensions more uniformly, while hierarchical codebooks impose a more structured and less isotropic geometry. Both families improve over scalar and one-hot extremes in certain regimes, but their optimal dimensions differ because their encoding geometries differ.

D Raw Data and Additional Samples

D.1 Raw Data of Section 4.2

Table 7 reports the raw results for the dimension sweep experiments in Section 4.2. These results are used to produce the dimension-performance curves in the main text.

D.2 Additional Samples of Section 4.3

Figure 5 shows randomly selected samples generated by the main real-vocabulary BFN model discussed in Section 4.3. The samples are generated using the same decoding and evaluation settings as in the main experiments. The displayed samples are from the T5-PCA128 codebook run.

E Potential Risks

This work studies encoding spaces for continuous language models and is primarily methodological. Nevertheless, improved language generation models may contribute to general risks associated with text generation, including the production of misleading or low-quality synthetic text, amplification of biases present in training corpora, or misuse in automated content generation.

F Use Of Scientific Artifacts

F.1 Licenses and Terms of Use

We use existing public datasets and pretrained models only for research purposes. The T5-small tokenizer and input embedding table are obtained from

Encoding	SA (↑)	Encoding	SA (↑)	Encoding	SA (↑)
Scalar-K27	56.15	Scalar-K60	39.96	Scalar-K128	35.73
One-hot-K27	71.73	One-hot-K60	69.73	One-hot-K128	64.89
Hier-K27-D3	30.81	Hier-K60-D12	60.15	Hier-K128-D16	57.80
Hier-K27-D5	33.59	Hier-K60-D15	65.82	Hier-K128-D32	67.75
Hier-K27-D9	69.91	Hier-K60-D20	63.73	Hier-K128-D48	71.36
Hier-K27-D12	73.91	Hier-K60-D30	68.05	Hier-K128-D64	71.81
Hier-K27-D18	73.65	Hier-K60-D45	73.47	Hier-K128-D80	76.35

Table 6: Hierarchical encoding results on Text8. Hierarchical codebooks also show an inverted-U-like trend, but their best dimensions are typically larger than those of sphere codebooks.

Encoding	SA (↑)	Encoding	SA (↑)	Encoding	SA (↑)
Scalar-K27	56.15	Scalar-K128	35.73	Scalar-K1024	39.72
One-hot-K27	71.73	One-hot-K128	64.89	One-hot-K1024	27.18
Sphere-K27-D2	64.03	Sphere-K128-D2	50.64	Sphere-K1024-D2	43.64
Sphere-K27-D4	67.55	Sphere-K128-D4	66.71	Sphere-K1024-D4	72.67
Sphere-K27-D9	69.90	Sphere-K128-D8	73.40	Sphere-K1024-D8	76.45
Sphere-K27-D12	70.91	Sphere-K128-D16	75.56	Sphere-K1024-D16	79.08
Sphere-K27-D18	72.80	Sphere-K128-D32	75.37	Sphere-K1024-D32	79.44
–	–	Sphere-K128-D48	73.90	Sphere-K1024-D64	78.42
–	–	Sphere-K128-D64	72.02	Sphere-K1024-D128	77.33
–	–	–	–	Sphere-K1024-D256	69.26
–	–	–	–	Sphere-K1024-D512	57.87

Table 7: Raw results for the sphere codebook dimension sweep. The table reports spelling accuracy (SA) for different vocabulary sizes K and codebook dimensions D.

the public T5-small checkpoint. CIFAR-10, Text8, and OpenWebText are used as standard research datasets. We do not redistribute these datasets or claim new ownership over them.

F.2 Consistency with Intended Use

The datasets and pretrained models used in this work are standard research artifacts, and our use is consistent with research-oriented language modeling and generative modeling experiments. We do not use the data to infer protected attributes, identify individuals, or make decisions about people.

F.3 Privacy, Identifiers, and Offensive Content

OpenWebText and Text8 are web-scale or web-derived text corpora and may contain personal names, offensive language, or other undesirable content that naturally occurs in web text. CIFAR-10 contains low-resolution images and class labels. We do not perform additional collection of personal data, and we do not attempt to identify or profile individuals. Our experiments operate on tokenized or discretized data and report aggregate metrics. We do not redistribute raw OpenWebText documents, Text8 text, or CIFAR-10 images. Since the goal of this paper is methodological analysis of encoding spaces rather than dataset construction or deployment, we rely on the standard research use of these benchmarks.

F.4 Artifact Documentation

The language data used in this paper is English. Text8 is a character-level benchmark derived from English Wikipedia text. OpenWebText is a web-text corpus intended to approximate the data distribution used for GPT-style language modeling. CIFAR-10 is an image dataset of small natural images from 10 object classes. The T5-small tokenizer and embedding table are used as pretrained token-level artifacts for the real-vocabulary experiments.

G Software and Implementation Details

Our models are implemented in PyTorch. Tokenization and pretrained T5 artifacts are obtained through the Hugging Face Transformers ecosystem. Dataset loading and preprocessing use standard Python data-processing utilities and public dataset interfaces. Generated perplexity is computed using a GPT2-Large evaluator. Image FID is computed using a standard FID implementation. All major implementation details that affect results, including sequence length, tokenizer, codebook dimension, model size, and evaluation metric, are reported in the relevant experimental sections.

If code is released, we will include package versions, command-line configurations, and codebook construction scripts. Any modification to existing model implementations, such as adapting BFN to arbitrary codebooks through its SDE interpretation, will be documented in the released code.

H Use of AI Assistants

AI assistants were used to help with language polishing, organization, and editing suggestions during manuscript preparation. They were not used as authors and did not replace the authors' responsibility for the scientific claims, experimental design, analysis, or final writing. All technical content, experiments, conclusions, and manuscript decisions were reviewed and approved by the authors.

income in Latin America’s quality of jobs. That is left, the expensive budget eliminated these 800 million spending programs. In giving rich American donors latto, with low labor unemployment they are far more far operating is so more subjective. It’s in offspring of environments and families, the more to listen. Trueest to unemployment is that a poll is: Contest. It limits its low amount of income tax to those where the budget wouldn’t fund its conservative investments. For 280.4 percent It comes to a redesign 45% programs. The program made conservative do niche spending. The new Alternative Fed Economic Reforms, a group of coalition programs which establishs programs that make investors be free of that. But it funds struggling to stallize every 12.1 million investment and raise its range of programs, increase in value missing industry prices, according to studies — by 20 percent (fe. 30 percent) I of tax reform by 58 percent I. Reiding programs are part of $5.3 million in companies participating, create up a percentage’s new credit. Capital Tax is the economy for a834-50 percent tax savings -- one of that’s where 10 percent make spending decisions. Price Tax

put onto this. Street so far as I see whether I still want to do anything. I don't know this then so that no matter what happened most about the men. "You go to do for myself, and carefully" sat cautiously. And this was the way I said what that quitting shocked. This I was thought to be again... someone, while they know me, excuses me. Not to say anything important. "I never really expects me that something you wrong, it is to do, it will be while you do anything. It went wrong and disastrous." Course Wilson's in particular factor is stark or uncertain. The change will happen, and what do you - actually frightening in a child of 20 people - there are all discussions with alsofrom the men people will need to get to do it.) 2.8: So you you know to be equally defended in their own, shall anyone whose office is our own, low level urban shore which is naturally on has the best. It is exceptional to people, not taught them, or as a person that has been acting hesitantly attacked across of the local community. The council were planning to build up on their high shore in the more buildings

Figure 5: Generated samples from the T5-PCA128 codebook experiment in Section 4.3. The samples are randomly selected from the generation set used for GenPPL evaluation.