Unsupervised Discrete Representations of American Sign Language

Artem Abzaliev and **Rada Mihalcea**
University of Michigan
{abzaliev, mihalcea}@umich.edu

Abstract

Many modalities are naturally represented as continuous signals, making it difficult to use them with models that expect discrete units, such as LLMs. In this paper, we explore the use of audio compression techniques for the discrete representation of the gestures used in sign language. We train a tokenizer for American Sign Language (ASL) fingerspelling, which discretizes sequences of fingerspelling signs into tokens. We also propose a loss function to improve the interpretability of these tokens such that they preserve both the semantic and the visual information of the signal. We show that the proposed method improves the performance of the discretized sequence on downstream tasks.

1 Introduction

Modern NLP models operate on a set of discrete tokens from a fixed vocabulary, which is the most common way to represent textual data. However, integrating modalities that are continuous into LLMs, such as videos, audio, or gestures, remains an open research question. In recent years a new paradigm has emerged, where an underlying sequence is first discretized ("tokenized"), and then provided to the LLM decoder (Borsos et al., 2023; Copet et al., 2024; Wang et al., 2023; Zhan et al., 2024; Jiang et al., 2023). While this is an intuitive way to incorporate more modalities into LLMs, it is not well understood how comparable these tokens are with respect to the ones used in text. In this work, with a specific focus on American Sign Language (ASL) fingerspelling, we investigate what kind of information is contained in these tokens, how this information is represented, and whether their intuitive meaning is similar to the subword tokens used for textual data.

We focus on fingerspelling ASL for several reasons. First, each visual sign in a fingerspelling video corresponds to a single text character and

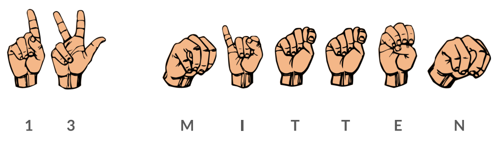

Figure 1: A fingerspelling sign sequence together with the corresponding phrase. Some characters (e.g., "T" and "E" here) can be visually similar, which makes the task challenging.

there is a many-to-one correspondence between the ground truth text and the sign sequence (e.g., the first N frames in the video represent character "x"). This is unlike audios or co-speech gestures, which usually do not have this property: for instance, a textual description of the audio ('Berlin techno') can refer to various time intervals in a clip, and many various descriptions can be generated for similar audio (many-to-many). This property allows us to get the correspondence between the discretized gestures and ground-truth characters. Additionally, fingerspelling has a relatively small character vocabulary size (63 different characters in our case), whereas full word vocabularies are much larger and harder to interpret. This allows us to visually inspect the similarities and differences in the tokens because each character has a corresponding sequence of frames.

"Sign language models that use have a history of underperforming those that operate on learned video embeddings; it is unclear to what extent this is due to the information bottleneck in the (imperfectly predicted) pose representation, vs. availability of higher quality pretrained video encoders than pretrained pose encoders. Pose inputs offer some benefits like computational efficiency and privacy."

We focus on pose-based inputs as a representation of ASL, as opposed to learned video embeddings, for two main reasons. First, it is a computationally faster method than processing the entire video, and because of this it can also be effectively run on a CPU. Second, flattening pose-based (x,y) coordinates allows us to apply techniques from the audio domain, such as neural compression, to the ASL domain.

The paper makes the following main contributions. First, we develop and train a Residual Vector Quantized (RVQ) hand fingerspelling tokenizer. Tokenization applied to this particular modality has not been explored before. We show that such tokenization opens possibilities for a range of applications, such as using a plain seq2seq transformer model for machine translation between sign language and written language, without convolutional architecture. Second, given our finding that these RVQ tokens are hard to interpret with information spread across several tokens, we propose an additional training objective that stabilizes the training and improves the interpretability of the resulting tokens. Specifically, we add during training a Connectionist Temporal Classification (CTC) loss between the predicted characters for each frame and ground truth fingerspelled character phrase, and show that this loss function improves the quality of the tokenizer on downstream tasks. Finally, we provide an in-depth analysis of the learned tokens and show that the residual quantizers encode information hierarchically from coarser to finer details.

2 Related Work

Audio Tokenization. A recent paradigm in audio processing is to map continuous audio signals into discrete tokens via self-supervised learning. This is typically done by imposing constraints on a latent space being discrete (van den Oord et al., 2018) and pre-training on a large amount of unlabeled data. For example, Baevski et al. (2020); Chung et al. (2021); Hsu et al. (2021) formulate self-supervised tasks as Masked Language Modelling similar to BERT (Devlin et al., 2019). Zeghidour et al. (2021) and Défossez et al. (2022) utilize adversarial training and RVQ to learn the discrete codes.

Language Modeling on Discrete Tokens. Discretizing audio allows us to apply the Next Token Prediction objective to the audio domain. Previous work has explored this direction, for instance AudioLM (Borsos et al., 2023), Vall-E (Wang et al., 2023; Chen et al., 2024), AudioGen (Kreuk et al., 2023), MusicGen (Copet et al., 2024). They show that language modeling is a meaningful task on audio tokens and, combined with powerful vocoders, can lead to state-of-the-art audio generation.

Latent Space Discretization for Other Modalities. In computer vision, Stable Diffusion (Rombach et al., 2022) and DALLE-E (Ramesh et al., 2021) both discretize and downsample the image space with discrete VAE (Kingma and Welling, 2022), and run the generation tasks in this latent space. Jiang et al. (2023) trains a tokenizer of the human poses and adds those tokens to the vocabulary of a language model. Zhao et al. (2023) uses BERT-like training on the word-level sign language. Different from those works, we operate on a character-level sign language.

3 Dataset

We perform all of the experiments using the American Sign Language Fingerspelling dataset (Ashley Chow, 2023). This is the same dataset used in a Kaggle competition for ASL fingerspelling. The dataset contains signs for randomly generated addresses, phone numbers, and URLs, in total more than more than three million fingerspelled characters produced by over 100 deaf signers. The keypoints were extracted from raw videos with the MediaPipe holistic model, 543 different landmarks with (x, y) coordinates. There are total of 63 unique characters in the dataset. We follow the official competition vocabulary, which includes many more characters that do not have phonologically distinct handshapes, for example "!", "&", or "@". The motivation to include those characters is to include URLs in the list of fingerspelled phrases.

We only use hand landmarks, 21 for each hand, so 42 (x, y) points for each video frame in total. We only use (x, y) coordinates since the z coordinates are noisy according to the dataset description webpage. We discard frames that do not have any hand coordinates on them, normalized the coordinates per video, and filled nans with zeros. We split the dataset into training, validation, and test sets, a total of 38,000 sign sequences for training and 13,000 for validation and test.

# characters	3M+
# signers	100+
# keypoints	543
# videos total	64,000

Table 1: Dataset statistics

4 Discrete Tokenization of ASL Fingerspelling

To create a discrete representation of the continuous signal of ASL fingerspelling, we first start by discretizing the individual video frames in the input sequence using an autoencoder with the discrete latent space induced by a Residual Vector Quantizer (RVQ) (Zeghidour et al., 2021). Our method builds upon Défossez et al. (2022), but without adversarial training and without time-domain compression. We do not compress along the time dimension since the sign language domain is different from the audio signal. For instance,Encodec compresses 16kHz audio into 50 integers, resulting in 320x compression. However, mono audios include only one channel, while in the ASL fingerspelling dataset, we use 42*2 channels, but relatively short duration (97% of videos are less than 262 frames in the video). Additionally, omitting compression along the time domain allows for easier visual inspection of individual tokens. Therefore we opt out for a 1:1 time mapping between continuous and discrete input and only compress along the channel dimension, so 42 (x, y) float coordinates are encoded into a single integer.

Autoencoder architecture. We use the modified Squeezeformer (Kim et al., 2022) adapted for gesture recognition from the first place of the ASL Kaggle competition [1] as an encoder. Squeezeformer is a hybrid architecture that combines convolutional and self-attention blocks. It is a small-sized model with a total of 1.329M parameters. A decoder is a mirrored version of the encoder with the same number of parameters.

RVQ utilizes several codebooks N_q to compress the signal hierarchically. The first codebook receives the embeddings from the encoder and compresses embedding for each frame into a single integer. Each subsequent codebook operates on the difference between the actual and quantized embeddings from the previous codebook. Additionally, Défossez et al. (2022) uses quantizer dropout (QDrop), where during the training all codebooks after random value $1 < k \leq N_q$ are not used, so the model learns to compress the signal hierarchically. We present all our results with 1024 discrete codes and 4 codebooks, with quantizer dropout. We ran preliminary experiments with fewer discrete codes/codebooks, but we found that the reconstruction quality was worse. This is in line with previous findings on video and audio tokenization (Yu et al., 2024; Ji et al., 2024), where researchers found that a large vocabulary size is beneficial for reconstruction.

Training. We train for 100 epochs in total and select the model with the lowest validation loss. We flip between right-hand and left-hand with 25% probability for each video. We use an AdamW schedule-free optimizer with a learning rate of 1e-3. [2]. We use 3 different loss functions: 1) Smooth L1 Loss between the original and reconstructed sign gestures similar to Jiang et al. (2023) 2) commitment loss from the quantizer to ensure that the codebook is utilized efficiently 3) CTC loss between the predicted character for each video/sign frame and the ground truth characters.

[1]https://github.com/ChristofHenkel/kaggle-asl-fingerspelling-1st-place-solution

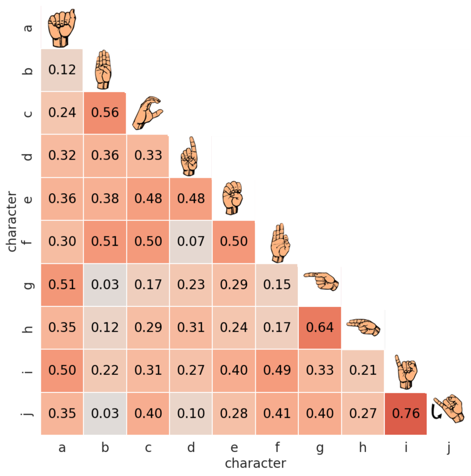

Figure 2: Cosine similarities between sign token embeddings that represent the characters. We define a sign language token to be representative for specific character by calculating the mutual information between the character and all sign tokens, and selecting the one with the highest mutual information. The tokenization model learns to group similar gestures together in the embeddings space, for example sign token embeddings "i" and "j", or "h" and "g".

4.1 CTC Loss on the Discrete Representations

A plain autoencoder using RVQ does not utilize any additional information from the input sequence, the training objective is simply to reconstruct the original signal as well as possible. However, such information is often available and can be used to boost the quality of the model. We propose to utilize signed phrases during the training: we add a small LM-like head on top of the quantized embeddings to classify each discretized frame into one of 63 characters. The motivation for this strategy is that we force the model to predict the same token for the same sign like "a" for a token "123", even if the visual information is different, i.e. different angle or different camera centering. During the training, after the RVQ but before the decoder, we add a small linear layer that for each video frame predicts a character it corresponds to. Since we know a ground truth phrase, we have all the necessary components for the CTC loss - it handles the many-to-one correspondence between the video phrases

[2]https://github.com/facebookresearch/schedule_free

and the characters, nudging the model to predict similar tokens for visually different sign gestures. Quantized video sequences are much longer than character sequences (262 frames vs 18 characters on average), but CTC loss is specifically designed for this scenario.

5 Interpreting Discrete Representations

5.1 Visualizing the Representations

After we train the tokenizer, we fully tokenize our training, validation, and test data, i.e. transform it from the set of (x, y) continuous coordinates into sequences of discrete values. We visualize learned tokens in Appendix A. Additionally, we investigate the sign token embedding space for the first quantizer. For each character "a"-"z", we can calculate the mutual information between the presence of a character in a fingerspelled phrase and a presence of a token in a discrete sign sequence. We select the tokens with the highest mutual information, and take the corresponding vector from the RVQ first quantizer, 128-dimensional in our case. We then can calculate cosine similarity between all "character-specific" sign token embedding vectors. The results of this analysis are presented in Figure 2. We show a heatmap with the cosine similarity values, and visualize the fingerspelled signs on the main diagonal. Similarly looking signs, like "i" and "j", or "h" and "g" are closer together in the embeddings space.

5.2 Information Preserved by the Tokenizer

Representation	CTC loss	Qdrop	Levenshtein distance
Discrete	✗	✗	0.648
Discrete	✗	✓	0.652
Discrete	✓	✗	0.665
Discrete	✓	✓	**0.690**
Continuous	n/a	n/a	0.684

Table 2: Levenshtein distance for seq2seq translation from sign language tokens to characters

RVQ discretization compresses (42*2) float coordinates into four integers for each time step t, where four is the number of quantizers and each quantizer has a cardinality of 1024. We want to evaluate how much information the tokenizer preserves. We run a series of downstream tasks, where we probe discrete gesture sequences for the ground truth text phrase. The first experiment we considered is a seq2seq translation from the sign language tokens to the text tokens. Both the encoder and the decoder operate on the set of discrete tokens, so we can apply standard seq2seq architecture from the machine translation where the decoder attends to the hidden states from the encoder. The only difference is that we have four integers instead of one, so we sum up all the embeddings from them, similar to Parallel Pattern in Copet et al. (2024).

We compare our seq2seq results with the convolutional approach where the encoder operates on the set of (x,y) coordinates. We use simpler architecture from the first-place solution from the Kaggle competition, Squeezeformer, but with only 42 keypoints.[3] We tokenize the text by a character with a total vocabulary size of 63 characters. We use normalized total Levenshtein distance as a metric for our comparisons since this is the metric that was used in the competition[4]. It is defined as $(N - D)/N$, where N is the total number of characters in all of the labels and D is the total Levenshtein distance for all the examples.

The results of this comparison are presented in the Table 2. During the experiments we noticed that quantizer dropout plays an important role, so we added quantizer dropout as an additional ablation variable. Without CTC loss and Quantizer dropout, tokenization results in small information loss - total Levenshtein distance is worse than the baseline continuous model, suggesting that tokenization deletes some details. Adding CTC loss mitigates the loss of information for this particular task and provides a small increase in the Levenshtein distance.

CTC loss	Qdrop	Avg. MI score
✗	✗	0.090
✗	✓	0.089
✓	✗	0.185
✓	✓	**0.455**

Table 3: Comparison of average mutual information scores for RVQ and our proposed approach with integrated CTC loss. The mutual information scores are calculated between the presence of a token/bigram token in a discretized sign sequence and the presence of a character in a fingerspelled phrase. The scores are averaged across all characters "a"-"z".

5.3 Token-based Predictions

Another question we investigate is how exactly sign language tokens interact with the final prediction: is a single token triggering a prediction (e.g., a token 42 is in the input, so the output must contain the letter "a"), or is it a more complex interaction, e.g.,

[3]First place winners used a two-step approach and a lot of data augmentations, while we only use a one-step approach and the only data augmentation we use is flipping the hand

[4]The best performing public model has a score of 0.836. This is not directly comparable to our method, since we trying to keep things simple to study discretization and not how to get the best possible performance

several tokens "1, 2, ..., 3" indicate the presence of the letter "a"? To check the single token hypothesis, we analyze the mutual information between two binary variables: the presence of a token from the first quantizer in a sequence and the presence of a character in a phrase, both just a binary label. Table 3 shows the results. For each character "a"-"z" we select a sign language token with the highest adjusted mutual information averaged, and average across all characters. Incorporating CTC loss (our method) significantly enhances the mutual information scores for individual tokens compared to those without CTC loss, demonstrating that a single sign language token largely influences a model's prediction of specific characters.

6 Conclusion

In this work, we developed a tokenizer for sign language, and used several strategies to probe the sign tokens for various properties. We showed that an RVQ quantizer can result in minor loss of information, and proposed a method to improve learned representations. Our results indicate that these tokens can be used as a drop-in replacement for their continuous counterpart, and they can be useful for architectures that assume discrete representations on continuous multimodal signals. Our code is available at `https://github.com/MichiganNLP/discrete_asl`.

7 Limitations

We focus on American Sign Language, although there are many other sign languages, such as the Australian Aboriginal Sign Language. Further exploration is needed to model and understand these sign languages.

Our approach relies on a hand landmarks detection model, and the results of our experiment may be influenced by the underlying keypoint extraction system. However, we anticipate any impact (after re-training) to be minimal. Additionally, CTC loss between the predicted characters and the ground truth characters requires additional annotations for the video, which may not always be available.

8 Ethics Statement

While our approach is trained on a large sample of signers, left-handed users represent a smaller fraction (10% of all the videos). Although we attempted to offset this issue with data augmentations, the smaller amount of data available for this group may still result in lower performance. A similar issue may be observed for users with physical impairments.

References

Manfred Georg Mark Sherwood Phil Culliton Sam Sepah Sohier Dane Thad Starner Ashley Chow, Glenn Cameron. 2023. Google - american sign language fingerspelling recognition.

Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations.

Zalán Borsos, Raphaël Marinier, Damien Vincent, Eugene Kharitonov, Olivier Pietquin, Matt Sharifi, Dominik Roblek, Olivier Teboul, David Grangier, Marco Tagliasacchi, and Neil Zeghidour. 2023. Audiolm: a language modeling approach to audio generation.

Sanyuan Chen, Shujie Liu, Long Zhou, Yanqing Liu, Xu Tan, Jinyu Li, Sheng Zhao, Yao Qian, and Furu Wei. 2024. Vall-e 2: Neural codec language models are human parity zero-shot text to speech synthesizers.

Yu-An Chung, Yu Zhang, Wei Han, Chung-Cheng Chiu, James Qin, Ruoming Pang, and Yonghui Wu. 2021. W2v-bert: Combining contrastive learning and masked language modeling for self-supervised speech pre-training.

Jade Copet, Felix Kreuk, Itai Gat, Tal Remez, David Kant, Gabriel Synnaeve, Yossi Adi, and Alexandre Défossez. 2024. Simple and controllable music generation.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Alexandre Défossez, Jade Copet, Gabriel Synnaeve, and Yossi Adi. 2022. High fidelity neural audio compression.

Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. 2021. Hubert: Self-supervised speech representation learning by masked prediction of hidden units.

Shengpeng Ji, Ziyue Jiang, Xize Cheng, Yifu Chen, Minghui Fang, Jialong Zuo, Qian Yang, Ruiqi Li, Ziang Zhang, Xiaoda Yang, Rongjie Huang, Yidi Jiang, Qian Chen, Siqi Zheng, Wen Wang, and Zhou Zhao. 2024. Wavtokenizer: an efficient acoustic discrete codec tokenizer for audio language modeling.

Biao Jiang, Xin Chen, Wen Liu, Jingyi Yu, Gang Yu, and Tao Chen. 2023. Motiongpt: Human motion as a foreign language.

Sehoon Kim, Amir Gholami, Albert Shaw, Nicholas Lee, Karttikeya Mangalam, Jitendra Malik, Michael W. Mahoney, and Kurt Keutzer. 2022. Squeezeformer: An efficient transformer for automatic speech recognition.

Diederik P Kingma and Max Welling. 2022. Auto-encoding variational bayes.

Felix Kreuk, Gabriel Synnaeve, Adam Polyak, Uriel Singer, Alexandre Défossez, Jade Copet, Devi Parikh, Yaniv Taigman, and Yossi Adi. 2023. Audiogen: Textually guided audio generation.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. 2018. Neural discrete representation learning.

Chengyi Wang, Sanyuan Chen, Yu Wu, Ziqiang Zhang, Long Zhou, Shujie Liu, Zhuo Chen, Yanqing Liu, Huaming Wang, Jinyu Li, Lei He, Sheng Zhao, and Furu Wei. 2023. Neural codec language models are zero-shot text to speech synthesizers.

Lijun Yu, José Lezama, Nitesh B. Gundavarapu, Luca Versari, Kihyuk Sohn, David Minnen, Yong Cheng, Vighnesh Birodkar, Agrim Gupta, Xiuye Gu, Alexander G. Hauptmann, Boqing Gong, Ming-Hsuan Yang, Irfan Essa, David A. Ross, and Lu Jiang. 2024. Language model beats diffusion – tokenizer is key to visual generation.

Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. 2021. Soundstream: An end-to-end neural audio codec.

Jun Zhan, Junqi Dai, Jiasheng Ye, Yunhua Zhou, Dong Zhang, Zhigeng Liu, Xin Zhang, Ruibin Yuan, Ge Zhang, Linyang Li, Hang Yan, Jie Fu, Tao Gui, Tianxiang Sun, Yugang Jiang, and Xipeng Qiu. 2024. Anygpt: Unified multimodal llm with discrete sequence modeling.

Weichao Zhao, Hezhen Hu, Wengang Zhou, Jiaxin Shi, and Houqiang Li. 2023. Best: Bert pre-training for sign language recognition with coupling tokenization.

A Visualizing learned tokens

Given the discrete representations, we can visually inspect what individual tokens look like. We show an example of a learned token in Figure 4. It can be seen that the model learned to group visually similar gestures together; in this case, this sign represents the number ’8’ in ASL. More examples and an illustration of how the hierarchical quantization affects the codes are included in Appendix A.

We visually investigate how several quantizers affect learned representation. We represent the hierarchical token structure as a tree in Figure 2. For this figure, we sample random signs from different videos for a specific token, for instance starting with (98, ...). We observed that the tokens from the first quantizer can be easily visually distinguished from each other (i.e. token 218 from Figure 4 vs token 98 from Figure 2), while it is much harder to distinguish other quantizers (i.e. see token (98, 264...) and token (98, 777...) on the Figure 2 This is expected and this result is an effect of quantizer dropout since we force the model to put as much information as possible into the first quantizer. We hypothesize that each subsequent quantizer after the first one grasps more and more fine-grained details. As an example, we calculate the average angle of the ring finger for the 4 bottom leaf tokens in Figure 2. The results in Table 4 confirm the hypothesis from the visual inspection: cluster (98, 624, 608, ...) indicates a very narrow-angle of the ring finger. Other tokens might specialize in similar fine-grained details, but this hypothesis requires further investigation.

Token	Average angle
(98, 624, 235)	71.4°
(98, 624, 608)	34.6°
(98, 777, 865)	68.7°
(98, 777, 888)	64.1°

Table 4: The angle between two joints for the ring finger for different tokens.

B Probing the Tokenizer

We also construct a simple probing task to predict whether a single character occurs in the text, for instance, whether we can predict an occurrence of character "a" in the phrase "vanilla" from the discrete sequence of gesture tokens. We train a small transformer encoder (6 layers, hidden dimension 128, 8 heads) with the classification head for 2 classes from scratch for this experiment. The vocabulary size of the model is 1027 (1024 discrete

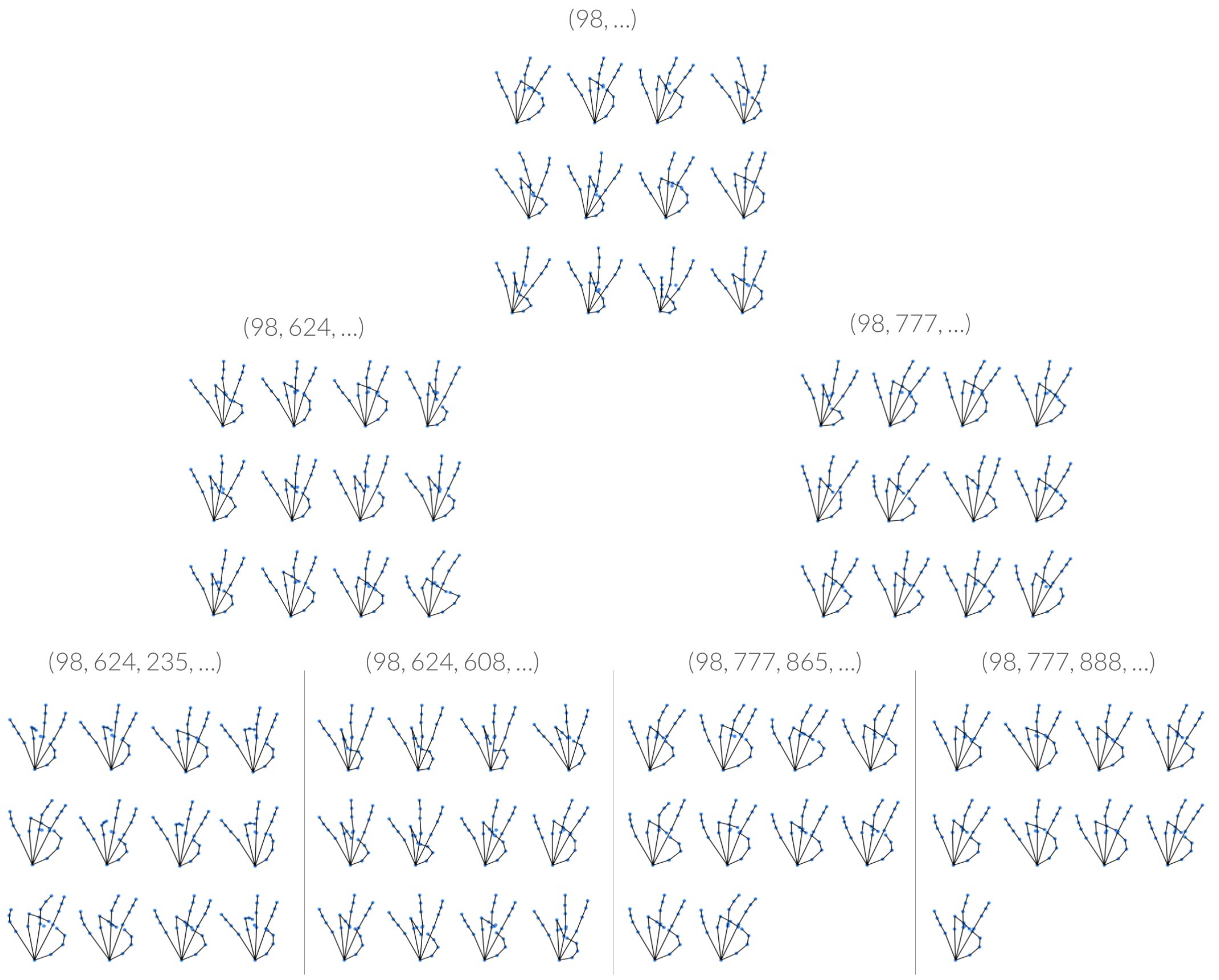

Figure 3: Hierarchical representation of token 98. Each sign token consists of 4 integers which we can represent as a tree. The root of the tree (98, ...) image shows randomly selected video frames that had 98 as a first integer. The second row is a subset of video frames that had 98 as a first integer, but split into 2 subgroups: (98, 624...) and (98, 777...). The third row is even further split by the results from the third codebook. We do not show the results from the 4th codebook, because it it hard to fit on the page and the differences are barely visible.

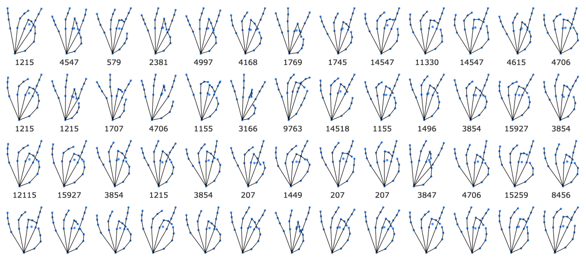

Figure 4: Example of a token (218, 89, ...) selected from random frames from our validation data. The integer value below the gesture indicates a unique video id, showing that the model learned to group similar gestures both within a single video and across the videos.

tokens from the tokenizer plus <BOS>, <EOS>, and <PAD> tokens). Since the quantizer returns 4 different integer sequences, we have 4 different embedding for each codebook and we sum them before passing to the transformer. We initialize each embedding layer in the transformer from the quantizer since it helps the model to converge faster. For the continuous architecture, we use a Squeezeformer encoder with the classification head for the two classes.

We run all the probing experiments on the test data, so the model does not see phrases during the training either directly or indirectly. The plain RVQ models without any additional modification perform slightly worse than a continuous model, with less than a 1% difference (0.11% on average across all alphabetic characters "a"-"z"). Our proposed RVQ model with CTC loss and Quantizer Dropout performs 4% better than the continuous model. Table 5 shows the results for each individual letter.

	Continuous	Discrete, no CTC	Discrete, with CTC
a	0.852	0.867	**0.908**
b	0.937	0.923	**0.948**
c	0.857	0.848	**0.896**
d	0.861	0.842	**0.888**
e	0.862	0.866	**0.919**
f	0.936	0.919	**0.962**
g	0.901	0.905	**0.925**
h	0.896	0.912	**0.926**
i	0.871	0.883	**0.915**
j	0.962	0.955	**0.978**
k	0.916	0.923	**0.932**
l	0.856	0.860	**0.881**
m	0.895	0.867	**0.913**
n	0.840	0.852	**0.898**
o	0.783	0.762	**0.887**
p	0.858	0.855	**0.905**
q	0.983	0.987	**0.990**
r	0.830	0.835	**0.907**
s	0.858	0.860	**0.915**
t	0.789	0.821	**0.887**
u	0.838	0.818	**0.886**
v	0.910	0.897	**0.944**
w	0.912	0.895	**0.944**
x	0.979	0.985	**0.989**
y	0.901	0.919	**0.924**
z	0.955	0.953	**0.974**

Table 5: Accuracies for a probing task "predict whether individual character is in a sequence". We compare continuous method, discrete RVQ without CTC loss and our proposed method with CTC loss.

C Language Modeling on Discrete Sign Representations

As a final downstream application, we train a phrase-conditional LLM on discrete sign representations, identical to MusicGen (Copet et al., 2024). The model accepts four sequences of integers from a quantizer together with the phrases to be signed and generates four parallel sequences of tokens. Those tokens can be decoded into the signs by using a decoder from the discretization model.

While MusicGen conditions audio generation on the description in natural language, we condition on the sequence of characters, where each character should represent several frames in the generated video. We use an embedding layer with a vocabulary size of 63 (63 characters in total in our dataset, although some appear very rare) and a projection layer on top. We train a decoder-only transformer with causal attention that attends to embedded characters and autoregressively predicts the next sign token given all previous sign tokens and all the characters in the phrase.

We train the model for 100 epochs, with AdamW optimizer and learning rate 3e-4. The accuracy of the next token prediction is 43% for the first quantizer, 26% for the second, 15% for the third, and 7% for the fourth. Lower accuracy for each subsequent quantizer is expected, given that we train the discretization model with the quantizer dropout, where we randomly drop different codebooks except the first one.

We use the model trained on continuous gestures to predict the phrase from the LLM-based generation. We generate 100 different samples for the random phrases in the test dataset. The resulting Normalized Levenshtein Distance is 0.278. Although some signs can be recognizable, this is a relatively low performance, which indicates that our method is promising but also leaves room for improvement in future work.