CMSC838B Project: Differentiable Symbolic Music for Personalized Violin Fingering Generation

Minsi Hu
Department of Computer Science
University of Maryland
College Park, MD 20742
mhu0315@umd.edu
https://github.com/Mimsqueeze/Violin-Fingering-Generation

Abstract

In this work, we present a novel method for generating personalized violin fingerings for symbolic sheet music, given user preferences. Recent work has explored this problem via statistical methods or BLSTM networks, but none have explored the powerful transformer architecture — renowned for excelling at processing sequential data. In this work we discuss a method for converting symbolic sheet music (in MusicXML format) into a differentiable format compatible with transformer architecture. Then, we explore and iteratively refine different transformer architectures to generate personalized fingerings, utilize LoRA + prefix tuning for finetuning on preferences, and present qualitative results of our method working reasonably well in practice. However, we find that biggest limitation for any method in solving this problem is the lack of a comprehensive violin fingering dataset, so future directions should develop a pipeline to obtain that data, or use self-supervision to take advantage of unlabeled data (as some related works do).

1 Introduction

The Neanderthals created bone flutes 50,000 years ago. Symbolic music has been around more than 1,200 years. Modern string instruments arose 500 years ago. The point is, music has been around for a very long time — and as long as music has been around, people have asked "what is the best way to perform a passage of music?" Of course there is no correct answer: it depends on the skill level of the musician, desired style of performance, and many other factors. Additionally, interpretation of music is purely subjective.

Another question we can ask is, given a passage of music, is there a way we can generate a method to perform a passage, curated to human preferences? Restricting the domain to violin music and symbolic sheet music, this project aims to address the following problem statement: "Given a passage of violin sheet music, generate an optimal set of finger positions conditioned on user preferences." Ideally, the generated fingerings should take the skill level of the music into account, be comfortable/viable to play, and fit the style of playing.

1.1 Motivation

We will provide some motivation on why this problem is meaningful. Figure 1 demonstrates some examples with violin sheet music with markings. The example in the top left demonstrates the difference between fingers are the considered "good" and "bad" depending on ease of playing. The good fingering minimizes the movement of both the left and right hand while playing the passage. The bottom left demonstrates a fingering pattern that allows for more expressiveness in playing, due to playing the passage on the lower, more resonant, rich "G string" on the violin. Finally, the example

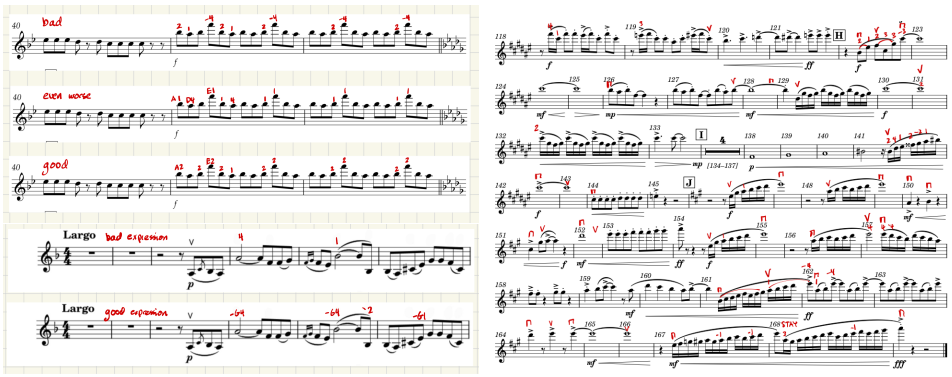

Figure 1: Examples of violin sheet music with markings.

on the right shows the amount of time and effort musicians need to put into creating comprehensive fingerings/markings for their music — oftentimes, being the biggest time sink when learning a piece of music. Therefore, such a model that can predict the optimal fingerings based on preferences is both non-trivial and practical!

2 Related Work

Extensive research has been done on music generation tasks, and translation tasks between different music modalities (text, symbolic music, audio). However, there hasn't actually been that much research done specifically on the task of generating violin fingerings. Older methods attempted to address the task of generating violin fingerings primarily via statistical methods, such as Optimal Path Paradigm [7] and Hidden Markov Model [6]. More recently, approaches have shifted to learning based methods, particularly using BLSTM (Bi-directional long short-term memory) networks [2, 5] to sequentially generate violin fingerings for a passage of music. However, no work has yet utilized transformer architecture [8], which we will be exploring in this work.

3 Methodology

For this methodology section, we will first explain the challenge of converting symbolic sheet music into a differentiable/continuous format, and then sequentially cover the approaches we tried in tackling this problem.

3.1 Differentiable Symbolic Sheet Music

One of the core issues in symbolic music processing is how we can represent sheet music in a way that a model can understand. Traditionally, symbolic sheet music is represented with MusicXML [3], which is a markdown language, similar to HTML, for representing symbolic sheet music. Typically, it encodes each note with a single tag, including note attributes such as position in the measure, duration, pitch, etc.

The first thing we need to consider is, how do we actually want to encode a note — which attributes do we want to use? It turns out, we only need three attributes to uniquely define a note inside of a piece of music: start (when the note starts), duration (how long the note lasts), and pitch (what is the frequency of the note). With these three attributes, which can be extracted from a MusicXML file, we can encode every note in a piece, listing the attributes for each note in a CSV file. We can also encode the beat type (eigth, quarter, half, etc.) of the note — we can be flexible with what attributes we want to encode. For our specific problem of violin fingering, we add three additional attributes: string, hand position, and finger to play the note with. This is the file format that is given to us in the TNUA Dataset [1]: they encode the start, duration, pitch, beat type, string, position, and finger for each note.

With these attributes, we can take inspiration from NLP tokenization and encode each attribute with an embedding matrix and concatenate them together: Refer to Figure 2.

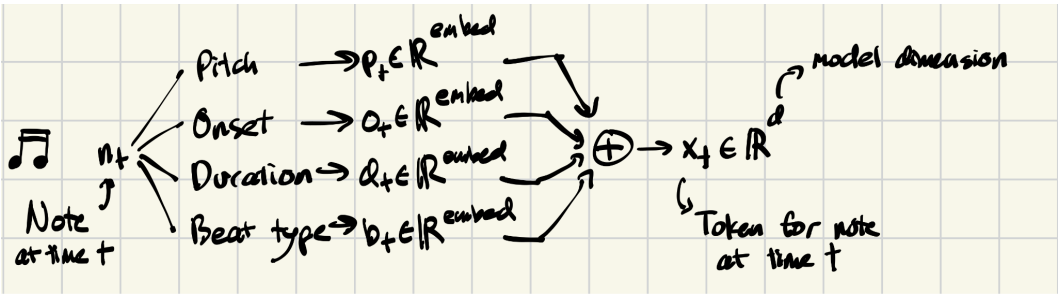

Figure 2: Diagram of note encoding into token

Note, these embedding matrices for each attribute will be learned in our proposed transformer architecture. Also, there is freedom in choosing which attributes to encode.

3.2 TNUA Dataset

The dataset we will use in this paper will be the TNUA dataset [1], containing 10 labeled violin scores each performed by 10 professional musicians.

3.3 Model Architecture

Now that we know a way to encode each note into a token, we can naturally use a transformer architecture to process note "tokens" and produce rich, output embeddings. We decided to explore transformers due to them not being previously explored in literature for this specific task, as well as the fact transformers are good at dealing with sequential data. Intuitively, the fingering of a note depends on the notes around it — making transformer architecture a clear choice due to its self-attention mechanism. Refer to Figure 3 for a diagram of our base model architecture, featuring a single encoder. All of our implementations build on this base models and stack layers of these encoders to produce rich note embeddings.

3.3.1 Initial Implementation — Unconditional predictions

Our initial implementation encoded each note with four attributes: pitch, duration, onset (start), beat type. We would feed the note embeddings into the encoder stack, which would produce rich embeddings. Then, we used a separate MLP to predict each fingering attribute: string (G, D, A, E), position (1-12), and finger (0, 1, 2, 3, 4). The loss function we used was simple the cross entropy between predicted attributes and ground truth attributes from the TNUA dataset. This initial approach worked poorly, due to the fact that string, position, and finger predictions were done with separate MLPs. As a result, they did not actually depend on each other, so the model "blended together" different fingerings. For example, in order to play an E5 (MIDI value 76) on the violin, a violinist may commonly play the open E string (String E, position 1, finger 0) or play fourth finger on the A string (String A, position 1, finger 4). Since both fingerings are likely and common, the model had a high probability of blending the two fingerings: predicting fourth finger on the E string or playing the open A string, which are invalid fingerings for the E5 pitch. Therefore, we needed a method to address this — we need to somehow tie together the prediction of string, position, and finger with each other. Diagram of this architecture found in Supplementary Materials Figure 6.

3.3.2 Improved Implementation — Conditional predictions

This, in the second iteration of the model, we decided to tie the predictions of fingering attributes together: use the rich note embedding to predict the string first. Then, use the string prediction AND note embedding to predict position. Then, use both position and string, as well as the embedding to predict the finger. This approach worked better! However, there were still cases of "blending" fingerings resulting in invalid configurations. This was most likely due to the loss function we were using: we were computing individual cross entropy losses over each fingering attribute (string, position, finger), and then summing them together.

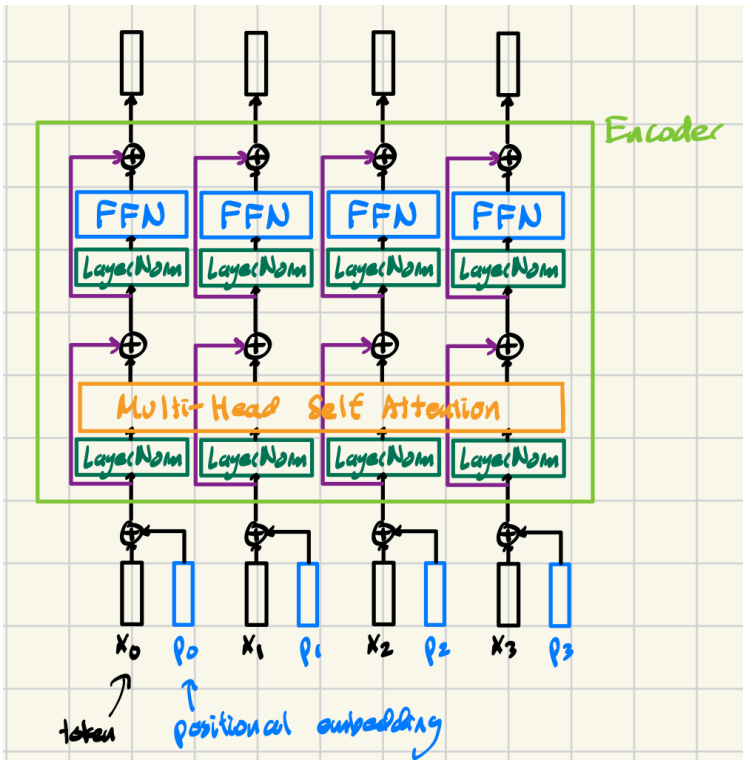

Figure 3: Diagram of our base transformer architecture.

$$L(s, p, f) = CSE(s, \hat{s}) + CSE(p, \hat{p}) + CSE(f, \hat{f}) \tag{1}$$

Therefore, we don't actually penalize for invalid fingerings: we penalize for each individual attribute being off. We needed to rethink the approach: specifically the fact our loss function in Eq 1 was inadequate. Diagram of this architecture found in Supplementary Materials Figure 7.

3.3.3 Joint predictions

We need a method of jointly predicting string, position, and finger at once. Something I wanted to avoid was explicitly recording which fingering orientations were possible for each pitch (because I thought it was going to be annoying), but I ended up doing this. I dropped the position attribute of fingering due to it not being completely necessary: I focused on the string and finger attributes (you can uniquely encode a fingering of a note with just these). Since there are 4 strings (G, D, A, E) and 5 fingerings (0, 1, 2, 3, 4), there are at most 20 possible fingerings for each pitch. However, some fingerings are not possible, for example you cannot play a note on a string lower than the base pitch of that string. Therefore, there is only one possible fingering for G3 (open G string). Additionally, there are ergonomic constraints: you cannot play a note with a finger that is greater than the number of semitones the note is above its base string. For example, to play a Ab4, which is a semitone above the A string, you can only use first finger on the A string. You cannot use any other fingers, since they would be too uncomfortable. Therefore, I manually encoded a mask of possible fingerings for each pitch, and instead of having MLPs predict each fingering attribute (string, position, finger), I have a single MLP take an output embedding and produce a probability distribution over possible fingerings, with this validity mask. Our new loss function is just a simple cross entropy loss (Eq 2).

$$L(x) = CSE(x, \hat{x}) \tag{2}$$

Turns out this implementation works MUCH better: we've solved the issue of invalid fingerings. However, we notice that the fingerings produced are still not very good. Our average F1 score is only around 0.53. With our current architecture, within a context window, we use our prediction MLP

to predict all fingerings of all notes within the context window at once! The issue with this is that the fingering of each note is not actually conditioned on the fingering of other notes: they related by cross-attention but the fingering of each note does not actually affect the fingering generating of other notes. We can solve this issue by making the model autoregressive — similar to how text generation works in NLP. Diagram of this architecture found in Supplementary Materials Figure 8.

3.3.4 Autoregressive generation

In order to make our model autoregressive, we employ the following scheme: within a context window of size N, we allow the model to "see" the fingering of the first N-1 notes. Then, conditioned on the fingering information of the first N-1 notes, the model must predict the fingering of the last (Nth) note, only the last note. For the start of the piece, where there aren't any notes to condition on, we can just use MASK tokens to fill the first N-1 slots. For inference, we can use a sliding window approach to autoregressively generate fingerings for all of the notes in the piece.

This approach works MUCH better! Now, each fingering prediction is conditioned on the previous N-1 fingerings predicted for previous notes. Our F1 score goes up to around 0.6 with this. However, we still have some issues: actual musicians write fingerings of a note depending on future notes to be played. For example, if there is a hard passage upcoming that is more comfortable to be played in a higher position, the musician should pre-emptively shift up to a higher position to prepare. Currently, with this autoregressive generation, the transformer does not have the context of future notes to determine fingerings. Diagram of this architecture found in Supplementary Materials Figure 9.

3.3.5 Autoregressive generation with future context

We can further improve on the previous model by having the MLP predict the fingering of the middle note in the context window, and mask out the fingerings of notes after the middle note. Namely, within the context window of size N, the model will have access to notes and fingerings of the first N//2 notes, it will try to predict the fingering of the note at position N//2 + 1 note, and it will not have access to the fingerings of the notes at position N//2 + 1 and onward. This way, the model can be autoregressive, but it doesn't purely have a casual mask. This distinguishes this task from pure text generation — we can do this because we already know the full sheet music. We are using the sheet music sequence to generate a sequence of fingerings. With this approach, we can autoregressively generate fingerings, conditioned on past generated fingerings and future notes. This is the final implementation that we explore for this project, and it performs well with a test F1 score of around 0.64 (note low F1 score due to no actual "ground truth", only preferences from dataset). Diagram of this architecture found in Supplementary Materials Figure 10.

3.3.6 Injecting Personalization

The last problem we need to solve is how to inject personalization (user preferences) into the generation task. To do this, the approach we opted for is training a LoRA [4] of our base model with the architecture described above, with an extra prefix token. Since encoding user preferences in abstract text preference is not in the scope of this project (we don't have the data for this as well), we will deal with a set of personalization options. The options we went for are performance level (measured in beginner, medium, advanced) and shifting level (less, normal, more). These preference attributes were derived through an analysis of the TNUA dataset: for each piece in the dataset, we computed the proportion of the piece that was spent in each hand position (1st to 12th position), and the shifting/position change rate (given a note, probability to change hand position for the next note). The intuitive for these statistics are that, if the piece spends a large proportion played in 1st (basic) position, it is more likely a fingering used by a beginner violinist. On the other hand, if less of the piece is played in 1st position, it is likely played by a more advanced violinist opting for higher expression. For position change rate, high values indicate a high frequency of shifting, while low values indicate a low frequency of shifting. Thus, with these two attributes, we can allow users to specify/customize what type of fingerings they want the model to generate.

We subsequently partition the dataset into three sets for each attribute: the lower 10% of pieces that spent the greatest proportion in first position are labeled as "beginner". The middle 80% we label as "medium". Then, the upper 10% of pieces spending the least amount of time in first position we label as "advanced" fingerings. For shifting, we label the lower 10% of pieces that shifted the least as "less

shifting", middle 80% as "normal shifting", and upper 10% as "more shifting". Then, we can use these labels to finetune the model with low-rank adaptation matrices.

The one caveat of this approach is that, because we partitioned the dataset 0.1, 0.8, 0.1, we have a disproportionate amount of data in each label class. Therefore, for training we balance the training by randomly sampling the same amount of pieces from each class, and we do something similar for testing/evaluation. With the finetuning approach, our model is able to reach up to 0.95 F1 score, predicting fingerings across different preference classes! This is our final model architecture, and can be found in Supplementary Materials Figure 11.

4 Experiment

Our final base model (Figure 10) took around 30 minutes to train on a single NVIDIA RTX A4000 GPU, on around 3300 32-note long sequences derived from the TNUA dataset. The initial model reached F1 score around 0.64, and the dataset did not have personalization labels. The finetuning process was done with LoRA + prefix tuning (Figure 11) on the data sequences labeled with personalization attributes: each label had 330 32-note long sequences. The finetuning model reached a testing F1 score of 0.95 on labeled personalization data, indicating a high degree of personalization capability in generation. The training and testing sets were generated by extracting 32-note long sequences from all pieces of music, scrambling then, and partitioning with a 90/10 testing/training split. We train each model over 100 epochs and save the model that achieves the highest F1 score on the testing set. The hyperparameters used to train both the base and LoRA models can be found in the code in our Github Repository. Inference with the model is very fast — the model can autoregressively predict fingerings for an entire violin concerto in less than 20 seconds.

We present some qualitative generation results of our final base model compared to our LoRA model conditioned on user preferences in Figures 4, 5.

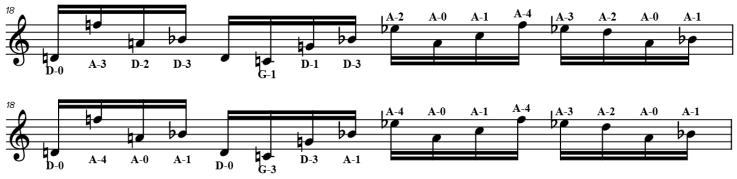

Figure 4: Examples of music generation from our model, excerpt from Bach: Sonatas and Partitas for Solo Violin, Partita No. 2 in D minor, BWV 1004. Top generated with base model, bottom generated with LoRA with (beginner, less shifting) user preferences.

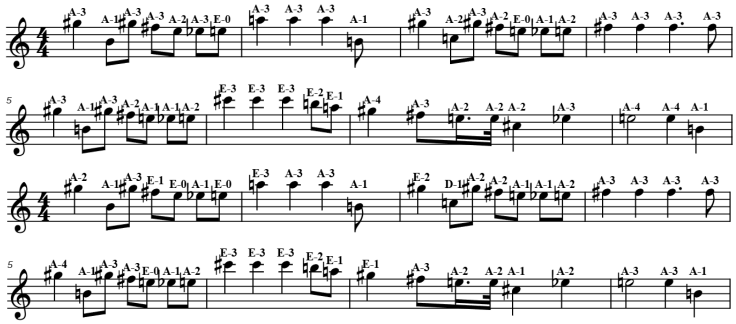

Figure 5: Examples of music generation from our model, excerpt from Elgar: Salut d'Amour, Op. 12. Top generated with base model, bottom generated with LoRA with (advanced, more shifting) user preferences.

5 Discussion

5.1 Qualitative Results

Analyzing the qualitative results seen in Figures 4 and 5, we can see that our LoRA model does well to augment generation of violin fingerings. In Figure 4, the base model (top) generates a reasonable fingering that shifts up into third position, and then stays there, before shifting into second position. For the LoRA model (bottom) with beginner and less shifting preferences, the model generates fingerings that allow the musician to stay in first position in the first half of the bar with an A-4 stretch to hit the F natural, then has the musician go into second position (like the base model) to avoid string crossings.

In Figure 5, we notice some interesting results. Both models generate mediocre fingerings — however, the LoRA model conditioned on advanced and more shifting preferences generates fingerings that tend to be more sporadic — more shifting and jumping up and down the string, and more string crossings. As a result, part of the generated fingerings for the LoRA model, albeit consistent with the conditions, are much less playable for a musician. This brings us into the limitations section.

5.2 Limitations

Unfortunately, when doing rigorous testing of the model it doesn't actually seem to perform very well. In many cases, the LoRA model produces the same/similar fingerings for contrasting preferences: beginner/advanced, less/more shifting. Additionally, when our model does produce different results for different conditioning, oftentimes it sacrifices playability. Our model does perform reasonably well for unseen data — but not great by the standards of a professional musician.

The reason for this is likely the lack of labeled, personalized data. For this project, we just used the original dataset (TNUA Dataset) and partitioned it based on heuristics. Additionally, the method we used to partition the dataset into finetuning datasets was by labeling renditions of entire pieces by artists. Therefore, many pieces did not have contrasting conditions: beginner/advanced, and instead only had beginner/medium. The same is the case for shifting. As a result, our model probably learned to memorize predictions based on the notes in the piece, rather than the condition provided. A better method of doing this would be partitioning on a per-piece basis.

Also, the TNUA Dataset by itself is already pretty small — meaning we had an even smaller finetuning dataset. The number one bottleneck for a super effective violin finger generation model is simply the lack of good data. Training only takes around 20 minutes, and the piece diversity is not great (only 10 violin pieces).

Additionally, we lack a good quantitative measure of model performance (this is why we don't have a quantitative results section). We can compute the loss and F1 score of the model for basic generation tasks, but we can't go much beyond that. As a result, many related works only include qualitative results and user case studies.

The training/testing split for this project was done by combining all sequences from all pieces from the TNUA dataset. In hindsight, a better method of evaluation would be to set aside one piece from the dataset, and train using sequences from the other nine pieces — therefore the model is actually evaluated with pieces that it hasn't seen before. However, this approach may not be practical with only a few pieces spanning limited styles of music and interpretations.

Other than the apparent lack of data for solving this task, some model improvements may include a method for the model to "go back" and revise previous fingerings (like masked LLM diffusion models), a better loss function that considers ergonomics and transition costs between fingerings, and better methods of preference conditioning.

6 Conclusion

In this work, we presented a novel method of generating personalized violin fingerings based on user preferences. We covered methods of converting symbolic sheet music into a continuous, differentiable format via token embeddings, and trained a transformer architecture to jointly and autoregressively predict violin fingerings. Personalization was injected via prefix tuning and LoRA, and we showed that our model was able to successfully produce reasonable fingerings for playing violin passages

— as well as being able to condition on user input. However, there are definitely many areas of improvement: particularly there is a need for a large, comprehensive violin fingering dataset with personalization captions, or at least a pipeline to obtain that data. If you would like to play around and explore the model, the code is available in this Github Repository!

References

[1] Vincent KM Cheung, Tsung-Ping Chen, and Li Su. An interactive automatic violin fingering recommendation interface. In *2021 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, pages 306–312. IEEE, 2021.

[2] Vincent KM Cheung, Hsuan-Kai Kao, Li Su, et al. Semi-supervised violin fingering generation using variational autoencoders. In *ISMIR*, pages 113–120, 2021.

[3] Michael Good et al. Musicxml: An internet-friendly format for sheet music. In *Xml conference and expo*, pages 03–04, 2001.

[4] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

[5] Wei-Yang Lin, Yu-Chiang Frank Wang, and Li Su. Enhancing violin fingering generation through audio-symbolic fusion. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 811–815. IEEE, 2024.

[6] Wakana Nagata, Shinji Sako, and Tadashi Kitamura. Violin fingering estimation according to skill level based on hidden markov model. In *ICMC*, 2014.

[7] Samir I Sayegh. Fingering for string instruments with the optimum path paradigm. *Computer Music Journal*, 13(3):76–84, 1989.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

7 Supplementary Materials

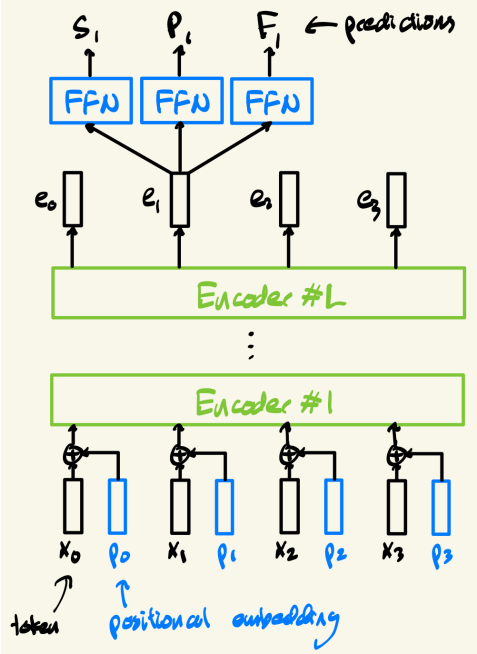

Figure 6: Initial Implementation architecture.

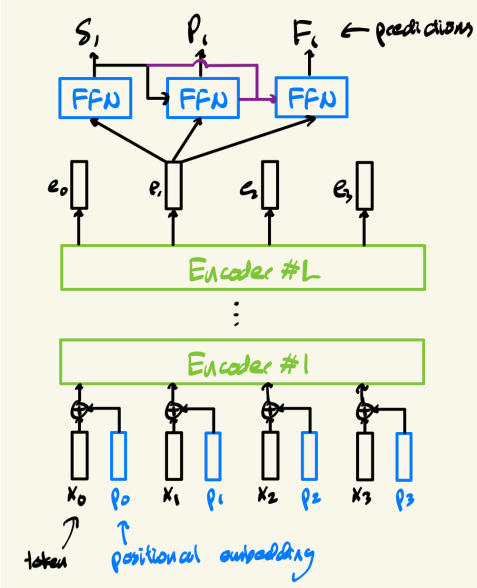

Figure 7: Conditional predictions architecture.

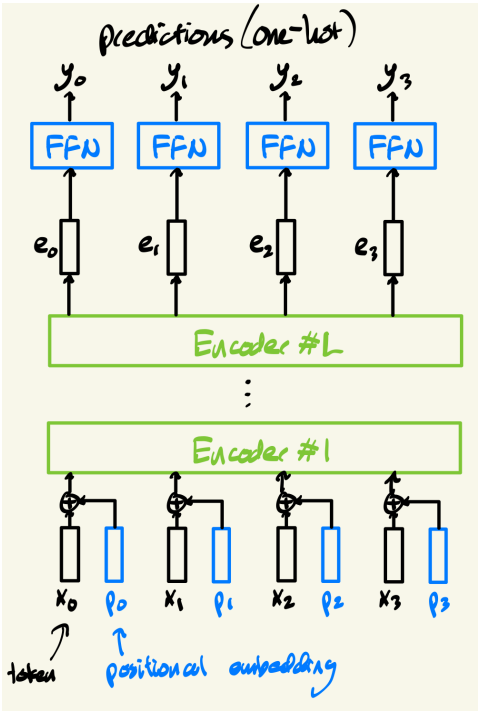

Figure 8: Joint predictions.

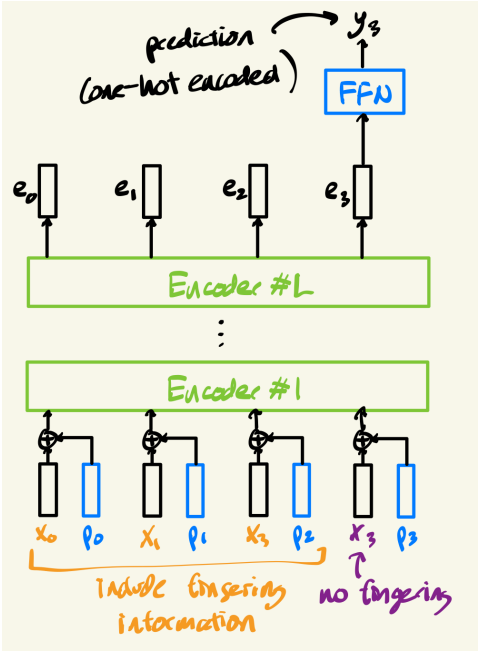

Figure 9: Autoregressive generation architecture.

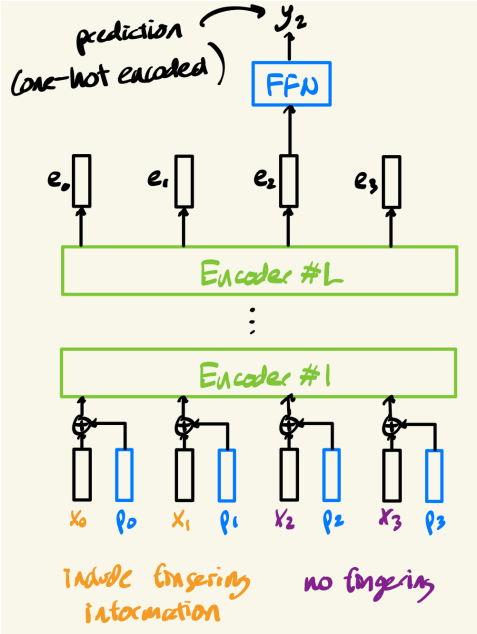

Figure 10: Future context architecture.

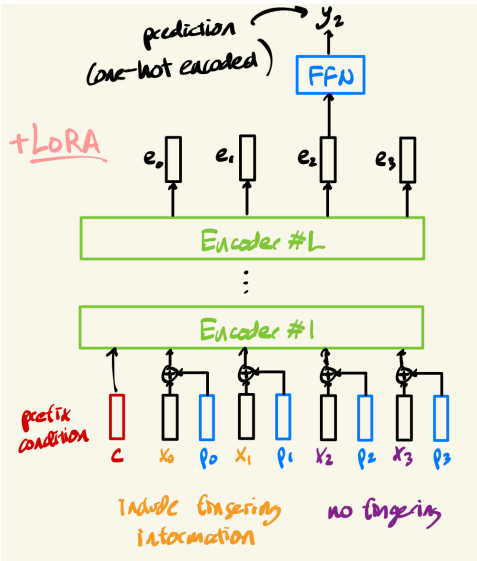

Figure 11: Finetuning architecture.