GenéLive! Generating Rhythm Actions in Love Live!

Atsushi Takada
Daichi Yamazaki
Yudai Yoshida
Nyamkhuu Ganbat
Takayuki Shimotomai
Naoki Hamada
KLab Inc.
Tokyo, Japan
{takada-at,yamazaki-d,yoshida-yud,ganbat-n,shimotomai-t,hamada-n}@klab.com

Likun Liu
Taiga Yamamoto
Daisuke Sakurai
Kyushu University
Fukuoka, Japan
{liu.likun.654,yamamoto.taiga.160}@s.kyushu-u.ac.jp
d.sakurai@ieee.org

ABSTRACT

A *rhythm action game* is a music-based video game in which the player is challenged to issue commands at the right timings during a music session. The timings are rendered in the *chart*, which consists of visual symbols, called *notes*, flying through the screen. KLab Inc., a Japan-based video game developer, has operated rhythm action games including a title for the "Love Live!" franchise, which became a hit across Asia and beyond. Before this work, the company generated the charts manually, which resulted in a costly business operation. This paper presents how KLab applied a deep generative model for synthesizing charts, and shows how it has improved the chart production process, reducing the business cost by half. Existing generative models generated poor quality charts for easier difficulty modes [3]. We report how we overcame this challenge through a multi-scaling model dedicated to rhythm actions, by considering beats among other things. Our model, named *GenéLive!*, is evaluated using production datasets at KLab as well as open datasets.

CCS CONCEPTS

• **Applied computing → Sound and music computing**; • **Computing methodologies → Artificial intelligence**; **Neural networks**; **Supervised learning**; **Batch learning**.

KEYWORDS

Generative model, music, video game, mobile application

1 INTRODUCTION

The success of deep generative models is rapidly spreading over the entire fields of industry and academia. In today's game developments, generative models are starting to help us to create various assets including graphics, sounds, character motions, conversations, landscapes, and level designs. For instance, the Game Developers Conference (GDC) 2021[1] held a special session named "Machine Learning Summit" to present various generative models in production, such as for generating character motions to match conversations [2] and for generating 3D face models of characters from human face pictures [14].

[1]https://gdconf.com/.

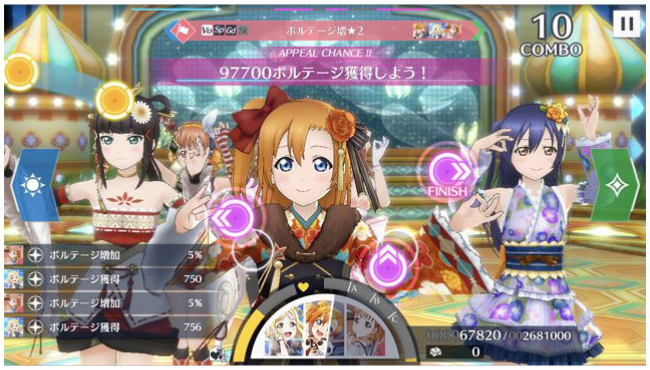


Figure 1: "Love Live! School Idle Festival All Stars".[2]

This article presents an application of a generative model for the rhythm action game business at KLab Inc, a Japan-based video game developer for smartphones and other mobile devices. The company has by today operated three rhythm action game titles online. A particularly successful title, "Love Live! School Idol Festival All Stars" or simply "Love Live! All Stars" (fig. 1), has been released in 6 languages and has been played worldwide while under KLab's operation, acquiring more than 10 million users. There has been a range of similar games with comparable impacts, which makes this work relevant to a large audience.

We target the generation of *charts*, which instruct the player to tap or flick buttons at specified moments, the defining challenge of rhythm action games. These buttons are known as *notes* as they fly through the screen forming a spatial pattern resembling a musical score. The audio record playing in the background is commonly referred to as a *song*. A song will be assigned charts of various difficulty modes ranging from *Beginner*, *Intermediate*, *Advanced* and *Expert* to *Challenge*, in the increasing order.

We focus on semi-automated chart generation. The company's workflow (fig. 2), indeed, does *not* demand a *fully*-automated chart generation, since KLab's artists need to experiment with different variations of candidate products, employing their professional skills

[2]©2013 PROJECT Lovelive! ©2017 PROJECT Lovelive! Sunshine!! ©2020 PROJECT Lovelive! Nijigasaki High School Idol Club ©Bushiroad International ©SUNRISE ©bushiroad.

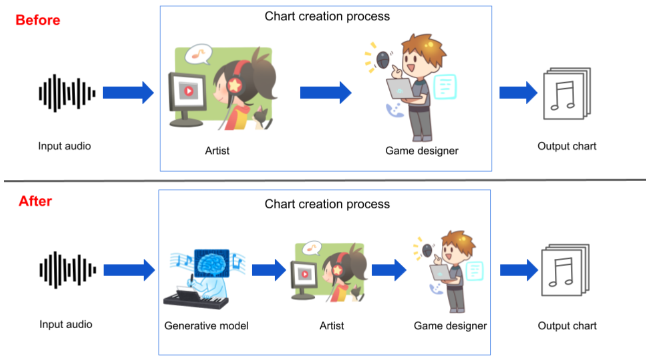


Figure 2: KLab's chart generation workflow before and after this work.

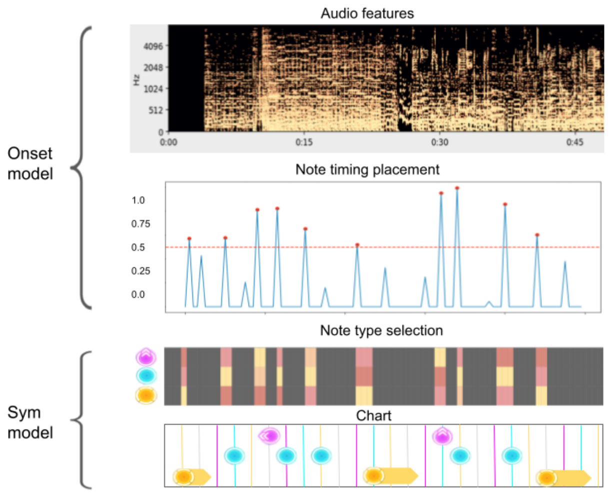


Figure 3: Data processing in the generative model. The present article focuses on the onset model.

– this is a high-level decision critical for the success of the franchise. We thus focus on semi-automation, which was to generate the *first drafts* of the charts (see fig. 3) so the artists can be freed from this low-skill labor. Another requirement was to prioritize the generation for easier difficulty modes, since generating harder ones tended to demand artistic decisions and game design ones (as explained in section 2).

Our model, named *GenéLive!*, successfully reduced the business cost by half (see section 7 for more feedback). The model has been deployed in business operation and will stay so for the foreseeable future.

Reducing the cost of chart generation is indeed a key business challenge for rhythm action developers since it is the bottleneck of the routine business operation. Instead of *altered* an input chart [24], KLab needed to generate a chart *from the audio alone*. Section 2 provides more details on the problem definition.

In our preliminary experiments, the Dance Dance Convolution (DDC) [3] generated charts for higher difficulty game modes at a human-competitive quality. (Find more related work in section 8.) However, the generation of low-difficulty charts had room for improvement (as the authors themselves pointed out [3]). As our primary target was easier modes, this was a significant challenge. As the problem replicated in KLab's datasets, too, we needed to improve the model. To this end, we have developed, and present, new techniques to overcome this problem: (i) the *multi-scale conv-stack* architecture that can capture the temporal dependency between quarter notes in the chart as well as 8th ones and (ii) the *beat guide* that hints at beat timing, which is musically natural positions to place notes in the chart (see section 5).

The DDC consists of two submodels: the *onset*, which generates the timing of a note, and the *sym*, which decides note types (like tap or flick). As deciding the timing is the bottleneck of KLab's workflow (detailed in section 2.3), this article focuses on presenting our onset model.

The present work has the following contributions:

- A deep generative model for the chart creation workflow of KLab's rhythm action titles, which halved the chart delivery time.
- The workflow is usable to rhythm actions in general – the results verified the versatility also for non-commercial use, which likely extends to other developers' business use.
- Our model achieved its business-quality performance by improving the state-of-the-art chart generator DDC [3] with two novel techniques, the multi-scale conv-stack and beat guide.
- Each of our improvements enhances the performance for all difficulty modes. The improvements were effective particularly for easier difficulty modes, overcoming a commonly known weak point of the DDC.
- The present model exhibited a generic performance in other game titles, too. These include *Stepmania*, whose dataset is publicly available, and another title at KLab.
- We share our best practices for utilizing state-of-the-art, computationally demanding generative models in bi-weekly release cycles within a moderate budget. These include utilizing a supercomputer, ResDevOps, and a web-based user interface. The practices seem to be applicable even outside the mobile game industry.

2 PROBLEM DEFINITION

Leading rhythm action titles today tend to take the form of one piece of a large entertainment franchise, like KLab's "Love Live! All Stars" does of "Love Live!". The company's role is to operate the mobile app, while songs are delivered by other participants of the franchise. After the first release of the app, KLab continued to contribute by offering new playable songs. This is why a significant cost for the company's business is posed by chart generation.

2.1 Priority on Easier Difficulty Modes

When we tried out the DDC's web-based demo [3], we concluded that the generative model exhibited reasonable performance for using it as the base model in our study. As stated before, one challenge was to overcome the well-known characteristic of the model, which was the suboptimal quality of chart generation for easier difficulty modes [3].

The easier modes were indeed the primary targets for our project as there was a significant amount of low-skill labor involved in these modes replaceable by automation (see section 2.3). Also, harder modes tended to be easier modes with alterations (as verified in section 4.2.2). These alterations, meant for core players, rely on artistic decisions that connect the gameplay to emotions – e.g. entertaining the player with novel action patterns and associating the actions to the dance animation being rendered in the background.

This led to our neural network architectures as introduced in section 5, which found improved performance for the easier modes.

2.2 Speed in Deliveries and Assistance

For each game title from KLab, four to six songs are made available to the players each month. Before this work, creating charts for one song took 40 hours on average. 12–18 records were processed for all difficulty modes, amounting to as many as 100 charts.

Releasing songs at a frequent pace has been found to be crucial for keeping active game players – the pace has been increased from 2 audio titles per month to 6 as the business expanded. This makes it all the more important to reduce the chart generation cost.

In fact, this project was launched in order to keep pace with the tripling increase in the frequency of chart generation (as well as that in the number of difficulty modes). Due to the fast speed of the mobile gaming business, where a business goal must be achieved in a short period, and priorities may change likewise, we were consulted to start reducing their workload within a few months.

The task thus was to establish aiding tools, including a novice user interface, rather quickly and to do so while collaborating with the artists and game designers from early on.

2.3 Chart Generation Workflow

To create a chart, artists repeatedly listened to the whole of a song to understand its musical structure as set by business partners. During this process, they pondered how to place hundreds of notes to be tapped by the player, to eventually craft the chart through trial and error. This first draft did not require too much expert skill, although it was causing as much half the cost in the workflow.

The charts were then modified so that the actions connected with emotions, like imitating the dance motion rendered in the background or flicking to a specific direction relating to the lyrics. It may be revised further to enhance the gameplay experience in more focused consideration on the overall game design.

In essence, the first draft of the chart generation was crafted only from the input audio, while the enhancements were applied from information harder to compile into numerical data. We thus targeted the auto-generation of the first drafts.

3 SOLUTION OVERVIEW

As it happened, KLab's chart generation workflow was formed without regard to automation, hardly reaching clear rules or a mathematical optimization target. Due to this, we chose to apply supervised machine learning. Fortunately, by the time of the start of collaboration (December 2019), KLab has released hundreds of audio sequences used in songs and corresponding human-generated charts.

On one hand, this project had the requirement of speedy deliveries and assistance. On the other hand, our project goal includes challenging research to improve the state-of-the-art deep generative model. As usual, it required a great amount of trial and error for studying novel neural network architectures, often taking six months or more.

To deal with the timescale gap, we organized a model development team and a model serving team and made them coherent by keeping in touch with the artist team to get feedback, reflecting it to model development and serving, and delivering an updated model as soon as possible.

3.1 Model Development

Our model development was conducted as a collaborative research project between KLab and Kyushu University. Since the team spreads over different organizations and locations, we need a web-based collaborative platform to share source code, datasets, models, experiments, and discussions across the team. Our system architecture for model development is shown in fig. 4. We will describe key elements of the system in the subsequent sections.

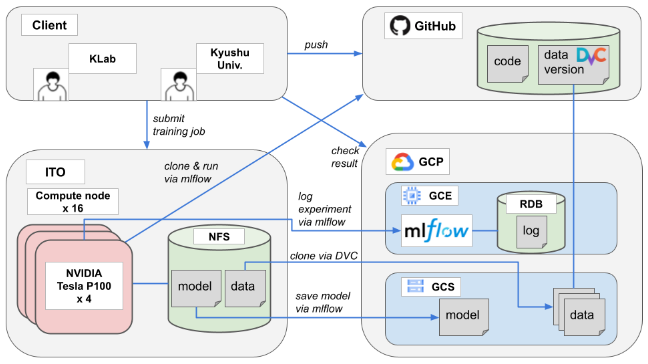


Figure 4: The system architecture for model development.

3.1.1 Version Control. Providing the single source of truth is the absolute foundation of our inter-organizational collaboration. KLab and Kyushu University shared source code on GitHub and worked together based on GitHub Flow. Each individual tried out their research idea on a separate feature branch. If the idea is found successful, then its feature branch is merged to the main branch.

As new songs and their charts are frequently provided by the artist team, data versions should also be managed. We stored every version of datasets in Google Cloud Storage (GCS) and tracked their hashes in GitHub by using Data Version Control (DVC).

3.1.2 Training on Supercomputer. To improve the DDC model, we would like to conduct thousands of training runs of a deep generative model, which requires a high-end GPU cluster. In our preliminary experiments, training a DDC model requires approximately four hours running in parallel on four NVIDIA Tesla P100 GPUs.

We used the supercomputer ITO[3] at Kyushu University for training and hyperparameter search. Our runtime environments were unified via MLflow Projects and Miniconda. Training jobs utilized

[3]https://www.cc.kyushu-u.ac.jp/scp/eng/

64 NVIDIA Tesla P100 GPUs, which are distributed to 4 units each in the 16 nodes. Overall, more than 80,000 GPU hours have been consumed for this project.

3.1.3 Collaboration on Cloud. The supercomputer shuts down several days every month due to scheduled maintenance. To keep experiments and models always accessible, our training jobs on the supercomputer transfer experimental settings initially, models and performance metrics periodically to our MLflow Tracking server hosted on a Google Compute Engine (GCE) instance. This enables us to share trial and error with colleagues in a real-time, completely reproducible manner and to discuss together on the fly.

3.2 Model Serving

The model serving was in charge of game operation engineers in KLab. To make our chart generation program available for artists on-demand, it should be executed by artists themselves without the help of engineers. Since the program requires high-end GPUs, installing it on artists' local machines is not a suitable option. Artists also wanted to preview a generated chart without integrating it to the game app. Our system architecture for model serving is shown in fig. 5. We will describe key elements of the system in the subsequent sections.

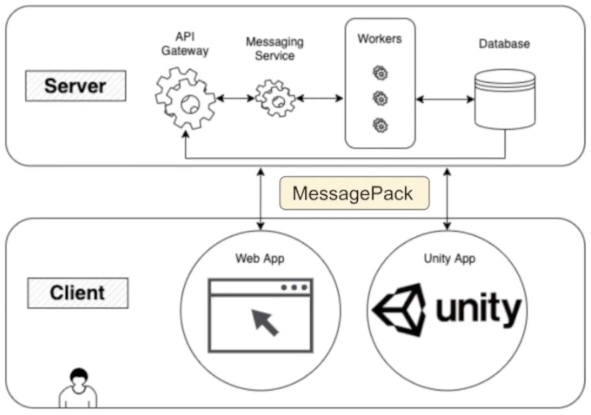


Figure 5: The system architecture for model serving.

3.2.1 Backend. To deliver new model versions continuously, we decided to provide the chart generation functionality as a web API. Since our model is implemented in PyTorch, we would like to use Python to write the chart generation API. The API was implemented by Fast API and hosted on Google Kubernetes Engine (GKE).

3.2.2 Web User Interface. Since our users were supposed to be non-engineers, we developed a novice user interface by Vue.js and Vuetify. The first step in generating a chart with this system is to upload an audio file of a song. Then, the user specifies the bit per minute (BPM) of the song, the difficulty mode, the minimum interval between notes, and the model version to be used. After a while, the generated chart becomes downloadable in MIDI or JSON format, which is notified on Slack. Artists will import the MIDI file into their digital audio workstation (DAW) software and edit it to better synchronize with the song's audio, lyrics, and dance movie.

3.2.3 Chart Previewer. Artists also would like to preview a generated chart on a playable screen to check its feeling as soon as possible. However, integrating the chart data into the actual game app takes a long time. We thus developed a simple Unity app for chart preview. This previewer loads the song and chart data from the chart generation server. With this app, the artists can play a song on their smartphones as soon as chart generation finishes.

4 DATASETS

4.1 Data Acquisition

We have acquired songs and charts used in "Love Live! School Idol Festival All Stars" (in short "Love Live! All Stars") and "Utano Princesama Shining Live" ("Utapri"). We additionally use openly accessible songs and charts from "Fraxtile" and "In the groove" of the game title "Stepmania", which were also in the prior work [3].

Table 1 summarizes the datasets. Charts of the same song with different difficulty modes are counted as different charts.

Table 1: Dataset Statistics

Datasets	# songs	# charts	# difficulty modes
Love Live!	163	501	4
Utapri	140	579	5
Stepmania Fraxtil	90	450	5
Stpmania ITG	133	652	5

4.2 Characteristics of Charts at KLab

To get ideas on how to improve the neural network architecture, we first analyzed the statistics of KLab's charts.

4.2.1 Note Timings. Figure 6 shows how frequently each note timing appears in KLab's charts. The 4th note accounts for 70–90% of a chart, and the 8th takes up 10–20%; the 12th and 16th are marginal. This fact gives us two ideas: (i) *beat guide*: the beat of a song will be a hint for placing 4th notes, (ii) *multi-scale conv-stack*: temporal max-pooling layers will be able to extract temporal dependencies of 4th and 8th note scale.

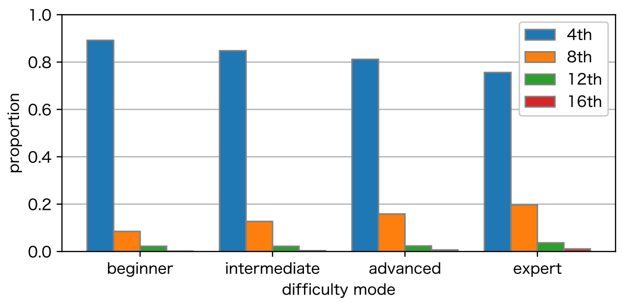


Figure 6: Note timings in "Love Live! All Stars".

4.2.2 Inclusion Between Difficulties. We view a chart as a bit string representing the existence of a tap for frames in a quantized time. We define the *inclusion rate* as

$$I(s,t) = \frac{\|s \otimes t\|}{\|s\|} \quad (1)$$

for strings s and t, where $\otimes$ is the bit-wise AND product and $\|\cdot\|$ is the number of 1s in a string. Inclusion rates (averaged over songs) in table 2 shows that different difficulty modes actually share a large amount of notes (similarly to other songs – see appendix A). This reassures our focus on easier difficulties and (re-)using the model for different modes (see section 6 for evaluation).

Table 2: Inclusion rates in "Love Live! All Starts".

s \ t	Beginner	Intermediate	Advanced	Expert
Beginner	100%	97.4%	97.4%	98.6%
Intermediate	64.7%	100%	98.8%	98.2%
Advanced	49.8%	76.2%	100%	98.6%
Expert	39.0%	59.4%	73.5%	100%

4.3 Data Augmentation

We augment the audio and note labels. The audio, and its mel spectrogram, are augmented using the following filters:

- Time mask – A time mask is a processing method that masks randomly selected frequency values in the time domain.
- Frequency mask – Similar to the time mask, the frequency mask masks a random frequency band for each iteration.
- Time warp – The time warp is a special technique we have introduced to help with our data augmentation. Contrary to the naive masking method, the time warp treats the mel spectrogram as an image and applies distortion horizontally to the spectrogram.

The training labels (tap or not), originally valued 1 or 0, are fluctuated [15].

5 GENÉLIVE!

To overcome the known weak point of the DDC, i.e., poor quality charts for easier difficulty modes, we designed a new generative model named "*GenéLive!*".

5.1 Audio Feature Extraction

Following Donahue et al. [3], our model uses the Short-Time Fourier Transform (STFT) and mel spectrogram of the audio. The STFT allows the model to capture features in the frequency domain. 32 ms were set for both the window length and stride.

The mel spectrogram cuts input features that are irrelevant to human ears. Following Hawthrone et al. [10], we use 229 logarithmically-spaced frequency bins. We set the lowest frequency to 0 kHz and the highest to 16 kHz (0 and 3575 in mel scale). Accordingly, 229 evenly distributed triangular filters in mel scale are applied.

5.2 Base Model

Our base model replicates the DDC [3] (our improvements are explained in sections 5.3 and 5.4). As shown in fig. 7, the base model consists of the CNN layers and LSTM layers. For signals in the frequency domain, we utilize the CNN layers to capture the frequency features, and for the time domain, we utilize LSTM layers for the task.

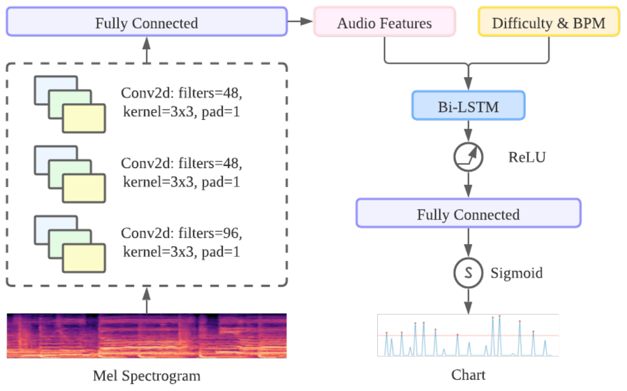

Fully Connected
Audio Features
Difficulty & BPM
Conv2d: filters=48, kernel=3x3, pad=1
Conv2d: filters=48, kernel=3x3, pad=1
Conv2d: filters=96, kernel=3x3, pad=1
Bi-LSTM
ReLU
Fully Connected
Sigmoid
Mel Spectrogram
Chart

Figure 7: Overall architecture of our network.

Convolution Stack. The main task for the convolution stack (or *conv-stack*) is to extract features from the mel spectrogram using the CNN layers. The conv-stack comprises a standard CNN layer with batch normalization, a max-pooling layer, and a dropout layer. The activation function is ReLU. Finally, to regularize the output, we use a fully connected layer.

BiLSTM. We propose to adopt bi-directional LSTM layers [6] to capture temporal features. Supplying as the input the output of the previous conv-stack.

Further, the chart generation needs to take into account the difficulty mode. To accomplish this, we have encoded the difficulty mode into a scalar (e.g., Beginner is 10, Intermediate is 20, and so on) and append this value as a new feature to the output of the conv-stack. An additional input to the BiLSTM layers is the beat guide we will see in section 5.3. Find details of the model architecture and the corresponding parameters in appendix C.

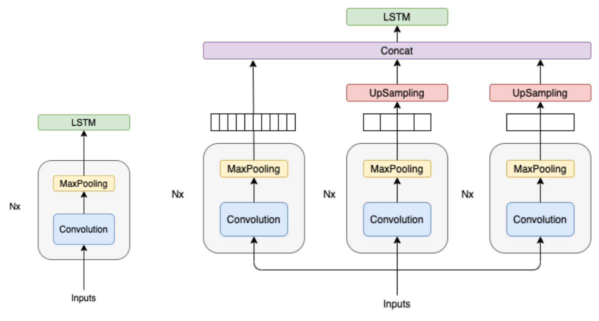
5.3 Beat Guide

Although it had been rare to consider the positions of beats in the model, the beat guide is indeed crucial to the generation of the charts, as beats correlate with the perceived emotion for a song.

Our BiLSTM layers thus process three pieces of information about beats: the starting time, tempo, and denominator. Since the tempo may shift during one song, the beat guide is encoded into a variable-length vector and is appended to the input feature. In each measure, the first beat is indicated by 2, the other beats are 1, and non-beat positions are 0.

5.4 Multi-Scale Conv-Stack

One key difference between the DDC and the present model is the structure of the conv-stack. In the model used in DDC, the convolution layers are applied repeatedly to the input of mel spectrogram, whereas the max-pooling reduces the matrix size only along the frequency axis and not time (fig. 8a).



(a) DDC [3] **(b) GenéLive!**

Figure 8: Conv-stack architectures.

The present model uses three conv-stacks with different temporal resolutions. The stack with the highest resolution (stack 1) does not perform max-pooling along the temporal dimension. The process is as same as the conv-stack of the DDC. In stack 2 and stack 3, max-pooling is also performed along the time dimension, and the length is reduced to one-eighth and one-sixteenth, respectively. Finally, up-sampling is applied to stack 2 and stack 3, and the three matrices, which have the same length in the temporal dimension, are concatenated (fig. 8b). By doing so, we expect to be able to extract not only the local features of a frame but also the global features of the surrounding frames by stacking three datasets with different temporal resolutions.

6 EXPERIMENTS

In order to measure the performance of each component of the present method, we take the ablation approach, in which the *GenéLive! model* with all the presented methods applied is compared against other models lacking some single component each.

Since the characteristics of the charts can differ between different game titles, we conducted experiments separately for each game title. In the following text, we deal with the results on the “Love Live! All Stars” dataset. For results on other datasets, see appendix B.

6.1 Set Up

6.1.1 Training Methodology. For an objective, we learned models to minimize *BCE loss* between model prediction and target label. Model parameters are updated by gradient-based method using an optimizer *Adam* [12]. Also, we adopted the cosine annealing scheduler [17] as a learning rate scheduler for better convergence. During training, dropout is employed in both the fully connected layer and LSTM layer to avoid overfitting.

6.1.2 Data Preprocessing.

Data split. The dataset was split into 8 : 1 : 1 for training, evaluation, and testing sets with holdout employed. The dataset was first split into a few subgroups with similar BPM (beat per minute). The stratified K-folds cross-validator was then used to re-split each dataset into three.

Chunking time length. Since we are using a model based on CNN and LSTM, there is no structural restriction on the time length of the input. In practice, however, it is impossible to input a whole song due to memory capacity constraints and the different lengths of each song. Thus, the audio and target are separated into short chunks and then fed into the model. Since the chunk length has room for exploration, we treated it as a hyperparameter and searched the optimal value in the experiment.

6.1.3 Hyperparameters. We conducted grid-search to find optimal hyperparameters. In the tuning process, we looked at the median of F-score^m metric (explained in section 6.1.4) for the validation dataset and chose the hyperparameter which gives the best performance. The hyperparameters consist of the following: the learning rate, $\eta_{\min}$ in the *cosine annealing scheduler* [17], the choice of conv-stack (conventional or improved one), the width and scale of *fuzzy label* [15], the dropout rate in the linear layer, the dropout rate in the RNN layer, the number of RNN layers, and the weighting factor in BCE loss.

6.1.4 Metrics. To supplement occasional evaluation with KLab’s artists (as shown in section 7), we calculated the F-score for the test datasets in two ways, following the manner in the previous work [3].

Firstly, the F-score is calculated by averaging the results for each chart. We denoted the metric as F-score^c. Charts with different difficulty modes for the same song are considered distinct. Secondly, the score, F-score^m, is calculated by micro averaging. We considered the predicted note to be true positive if the note is placed within ±50 ms around the ground truth.

6.1.5 Computing environment. We conducted the experiments using the supercomputer ITO’s NVIDIA Tesla P100 GPUs (HBM2 16GB VRAM (720GB/s), 1,328–1,480MHz, 56SM, 3,584 CUDA cores). The implementation is based on `pytorch`, and to pre-process audio data `librosa` was employed.

6.2 Results

We conducted 10 experiments for each condition. For the final evaluation of the F-score, we adopted the median of the 10 trials, instead of the mean, to ignore disturbance by outliers. The following procedure was taken for the evaluation per trial: the model parameters were stored for each of the 200 iterations in the model training process, and all evaluation metrics were calculated for each step. Then, the value with the highest performance for each evaluation metric was adopted as the performance of the model.

The model was trained using charts of all difficulty modes, and the prediction results were obtained using charts of each difficulty mode where the conditioning vector of the difficulty mode was added to the input. In the case of evaluating the performance regardless of the difficulty mode, all predictions for each difficulty mode were combined and calculated metrics based on the same average method of each evaluation index.

Table 3 shows the present model, GenéLive!, consistently outperformed the state-of-the-art model, DDC.

Table 3: Chart generation quality of the present model (GenéLive!) and the state-of-the-art (DDC).

Difficulty	F-score^m (std.) GenéLive!	F-score^m (std.) DDC	F-score^c (std.) GenéLive!	F-score^c (std.) DDC
Beginner	0.8643 (0.0018)	0.7823 (0.0081)	0.8627 (0.0018)	0.7832 (0.0080)
Intermediate	0.7953 (0.0024)	0.7452 (0.0039)	0.7949 (0.0020)	0.7465 (0.0045)
Advanced	0.7885 (0.0017)	0.7494 (0.0044)	0.7878 (0.0021)	0.7503 (0.0042)
Expert	0.7784 (0.0062)	0.7610 (0.0059)	0.7796 (0.0060)	0.7607 (0.0058)
all	0.8021 (0.0016)	0.7524 (0.0046)	0.8104 (0.0013)	0.7559 (0.0049)

The results of the experiment without the beat guide are shown in fig. 9. For comparison, the results of the GenéLive! model are also plotted side by side.

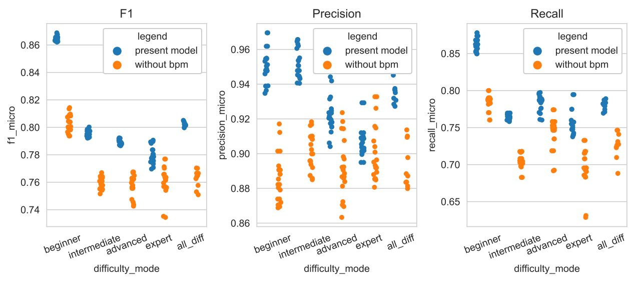


Figure 9: Performance boosted by the beat guide.

The results of training the model using the unmodified version of the conv-stack are shown in fig. 10. For comparison, the results of the GenéLive! model are also plotted side by side.

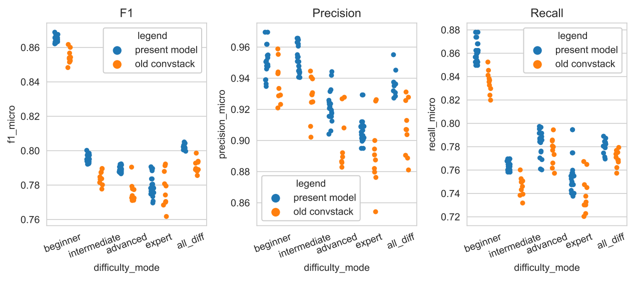


Figure 10: Performance boosted by the conv-stack.

The results of separate trainings for each difficulty mode are shown in fig. 11. In this case, we trained and evaluated the model using data of single difficulty mode. For comparison, the results of the GenéLive! model, where the model are trained using charts of all difficulty modes at once, are also plotted side by side.

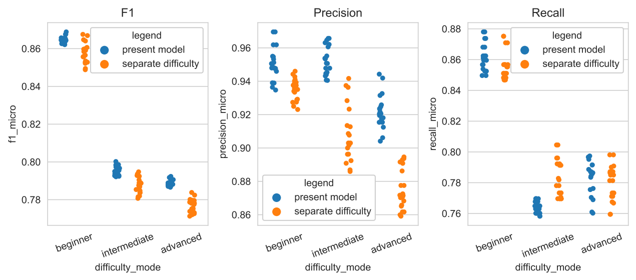


Figure 11: Performance boosted by difficulty-wise training.

6.3 Discussion

6.3.1 Beat Guide. Figure 9 shows that removing the beat guide significantly degraded the performance of the chart prediction task. Compared with the other two experiments, the beat guide is the most significant element in the present model.

The model's CNN seems to value locations where the sound is loud. If the beat guide is absent they tend to fail at other rhythm patterns like a repeating sequence with moderate volume.

Comparing the results for different difficulty modes, we can see that the effect of adding the beat guide to the input was larger for the easier difficulty modes. In general, charts of lower difficulty modes tended to consist of rhythms easier to capture. Keeping the beat of the song is one of the simplest rhythms, so in charts of lower difficulty modes, the note is often placed at the beat positions as shown in fig. 6. In turn, the easier difficulty mode of a chart, the more emphasis is placed on periodical rhythms, which CNN-based models are not good at, than the sound played in the music, and this may be the reason why the performance of the note generation task with easier difficulty modes is lower in previous studies such as [3]. Our experiments show that adding the beat guide to the input is a key to overcoming this weak point.

6.3.2 Multi-Scale Conv-Stack. Figure 10 shows that when we used the DDC's conv-stack, the performance degraded. This is the effect of the model being able to "look at" the time direction not only in the LSTM layer but also in the CNN layer by improving conv-stack. Specifically, compared to the original conv-stack, the stack 2 of our multi-scale conv-stack is able to take into account 8 times coarser information in the time direction (as discussed in section 5.4). Since the time resolution is 32 ms in our case (as explained in section 5.1), the second stack can consider 256 ms forward or backward, which corresponds to the length of a 8th note in a 120 BPM song. Similarly, our stack 3 sees 16 times coarser information, which amounts to a 4th note length. Experiments on other datasets also show the performance improvement, see appendix B.

6.3.3 Training With All Difficulties. Figure 11 shows that when all difficulty modes were trained with a single model, the performance improved. This result implies the effectiveness of the CNN for learning charts of varying difficulties. While it was understood that the DDC model as proposed [3] was poor at generating charts for easier difficulties, it had been unclear what kind of improvements should be effective. For example, is it better to let a *single* model instance consume charts having multiple difficulties or, instead, *multiple* instances consume them, where each instance is specialized for a certain difficulty mode? On one hand, the similarity of charts in different difficulties for the same song could work as a hint. On the other hand, however, the network might be confused by the same song resulting in different charts. Another question regarding the poor performance of the DDC on easier difficulties is whether the fundamentals of its design had a flaw in the first place.

This work reveals that learning different difficulties with a single model instance in fact outperforms the other, under the conditions stated in section 4.2.2. It also follows that the DDC's direction allows learning easy difficulties, although our model had received improvements and was intended for KLab's dataset.

7 BUSINESS FEEDBACK

Since the first deployment of this chart generation system to this day (July 2020–January 2022; 18 months), the artist team has used the system to create charts for all 110 songs released in "Love Live! All Stars" (82 songs had been released before the period).

The present collaboration cut down half the chart creation time for the artist team. About 20 hours of work are saved per song, whereas it used to take about 40 hours per song. For the Beginner and Intermediate difficulty modes, the charts generated by our model can be used with minor modifications. The artist team uses our model also for the more difficult game modes.

The artist team told us that they were able to start using our model quickly and to continue doing so thanks to the web user interface we provided for them (fig. 12). The parameters that the artists have to supply have clear musical meanings such as minimum note interval and BPM, hiding technical details of the ML model which are foreign to the artists. Using a web interface also allowed us to update the model without interfering with the artist team's computers.

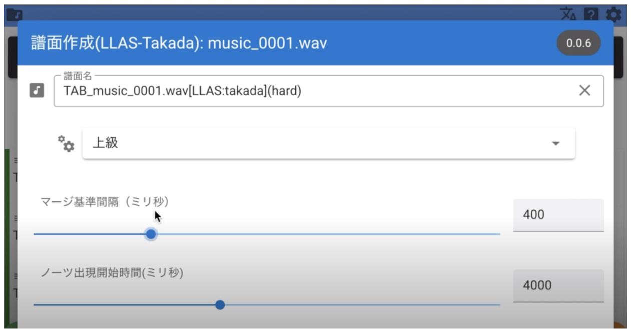

譜面作成(LLAS-Takada): music_0001.wav
0.0.6
譜面名
TAB_music_0001.wav[LLAS:takada](hard)
上級
マージ基準間隔（ミリ秒）
400
ノーツ出現開始時間(ミリ秒)
4000

Figure 12: The web user interface

In most cases, the timing for notes was determined by the model and artists used them as they were. Figure 13 compares the first 8 measures of an automatically generated chart for the Advanced difficulty mode of a typical song with the released version which was manually modified from the generated one. Of the 22 auto-generated notes, 21 were accepted as they were and the other two were not used, while three additional notes were added.

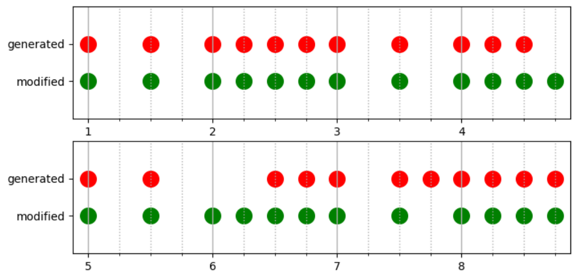


Figure 13: A generated chart for the Advanced difficulty mode and its manually-modified version (the first 8 measures of a typical song).

A tendency in the parts that have been manually corrected was that the parts failed to capture the musical structure. In the 8 measures of the example chart in fig. 13, the audio record for the measures 5–8 (lower row) was a repeat of the first 4 measures (upper row). These two portions were thus manually changed by the artists into a repeating pattern. Note that manually duplicating notes like this is actually an easy task for the artists. Overcoming this issue of the model was thus given low priority by the artists.

8 RELATED WORK

Early chart generation tended to use a rule-based algorithm [19] or genetic algorithm [18]. The DDC by Donahue et al. [3] improved the quality by employing the deep neural network. Lin et al. used a multilayer feed-forward network to generate charts, synchronizing the notes with instrumental sounds [16]. The model by Tsujino and Yamanashi [24] alters the difficulty of an input chart – this work instead generates the first draft chart (without an input chart).

Onset detection based on neural networks is studied for speech recognition, where BiLSTMs [5], CNN approch [21] and multi-resolution feature representation are established. Schluter & Böck [21] demonstrated a CNN-based approach. Likewise, music information retrieval has also seen benefits of neural networks [1, 11, 25].

In addition, various deep generative models are used to create game contents and assets. *Procedural content generation via machine learning*, a model for generating game content using trained machine learning models, can generate a variety of game content such as items, maps and rules [8, 22]. Hastings et al. proposed an automatic content generation that learns players' preferences based on their past play history and generates new graphical and game content during gameplay [9]. In addition, Green et al. proposed a framework generating tutorial text by discovering whether it is effective to win or lose a game [7]. Tilson et al. investigated generating image assets for games using unsupervised learning such as GAN and VAE [23].

Volz et al. have generated various levels of Super Mario Bros. using GAN [26]. Park et al. used GAN to generate levels for a computer science educational game [20].

There is a method to synthesize high-quality, realistic full-body animations of 3D characters using deep learning [2]. It is possible to express emotions and generate facial expressions and movements corresponding to the NPC's personality and profession. A model has been developed to create a face mesh from a photograph of a face and generate a face model of a 3D character [14].

In addition, there are examples of successful use of deep learning to reduce man-hours in the game development process, such as the development of AI for mini-games in MMORPGs by applying AlphaZero [13] and the improvement of game frame rates by applying deep super-resolution in the time direction [4].

9 CONCLUSIONS

We assisted chart creation at KLab Inc. by establishing a new deep generative model, while the model ended up being in fact versatile for more generic datasets. The model successfully generated charts for easier difficulty modes, filling the quality gap between easier and harder game modes, which was a challenge that had been admitted by the authors of the state-of-the-art model DDC. This was achieved by utilizing two techniques, (i) the multi-scale conv-stack and (ii) the beat guide. KLab's artists use the model via a web interface, cutting the business cost by half.

REFERENCES

[1] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. 2013. Audio Chord Recognition with Recurrent Neural Networks.. In *Proceedings of the 14th International Society for Music Information Retrieval Conference*. ISMIR, Curitiba, Brazil, 335–340. https://doi.org/10.5281/zenodo.1418319

[2] Yu Ding. 2021. Machine Learning Summit: Full-Body Animation Generation for Expressive NPCs. https://www.gdcvault.com/play/1027068/Machine-Learning-Summit-Full-Body Accessed on February 10, 2022.

[3] Chris Donahue, Zachary C. Lipton, and Julian McAuley. 2017. Dance Dance Convolution. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, Sydney, Australia, 1039–1048. https://proceedings.mlr.press/v70/donahue17a.html

[4] Andrew Edelsten. 2021. Machine Learning Summit: 3D Parametric Face Model and Its Applications in Games. https://www.nvidia.com/en-us/on-demand/session/gdc21-gdc2103/ Accessed on February 10, 2022.

[5] Florian Eyben, Sebastian Böck, Björn W. Schuller, and Alex Graves. 2010. Universal Onset Detection with Bidirectional Long Short-Term Memory Neural Networks. In *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR 2010, Utrecht, Netherlands, August 9-13, 2010*, J. Stephen Downie and Remco C. Veltkamp (Eds.). International Society for Music Information Retrieval, Utrecht, Netherlands, 589–594. http://ismir2010.ismir.net/proceedings/ismir2010-101.pdf

[6] Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks* 18, 5-6 (2005), 602–610.

[7] Michael Green, Ahmed Khalifa, Gabriella Barros, and Julian Togellius. 2021. "Press Space to Fire": Automatic Video Game Tutorial Generation. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 13, 2 (Jun. 2021), 75–80. https://ojs.aaai.org/index.php/AIIDE/article/view/12977

[8] Matthew Guzdial, Joshua Reno, Jonathan Chen, Gillian Smith, and Mark Riedl. 2018. Explainable PCGML via Game Design Patterns. In *Joint Proceedings of the AIIDE 2018 Workshops co-located with 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2018), Edmonton, Canada, November 13-14, 2018 (CEUR Workshop Proceedings, Vol. 2282)*, Jichen Zhu (Ed.). CEUR-WS.org, Edmonton, Canada. http://ceur-ws.org/Vol-2282/EXAG_107.pdf

[9] Erin Jonathan Hastings, Ratan K. Guha, and Kenneth O. Stanley. 2009. Automatic Content Generation in the <emphasis emphasistype="italic">Galactic Arms Race</emphasis> Video Game. *IEEE Transactions on Computational Intelligence and AI in Games* 1, 4 (2009), 245–263. https://doi.org/10.1109/TCIAIG.2009.2038365

[10] Curtis Hawthorne, Erich Elsen, Jialin Song, Adam Roberts, Ian Simon, Colin Raffel, Jesse H. Engel, Sageev Oore, and Douglas Eck. 2018. Onsets and Frames: Dual-Objective Piano Transcription. In *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, September 23-27, 2018*, Emilia Gómez, Xiao Hu, Eric Humphrey, and Emmanouil Benetos (Eds.). International Society for Music Information Retrieval, Paris, France, 50–57. http://ismir2018.ircam.fr/doc/pdfs/19_Paper.pdf

[11] Eric J. Humphrey and Juan Pablo Bello. 2012. Rethinking Automatic Chord Recognition with Convolutional Neural Networks. In *11th International Conference on Machine Learning and Applications, ICMLA, Boca Raton, FL, USA, December 12-15, 2012. Volume 2*. IEEE, Florida, USA, 357–362. https://doi.org/10.1109/ICMLA.2012.220

[12] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. http://arxiv.org/abs/1412.6980 cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[13] Zihan Lei. 2021. Machine Learning Summit: Applying 'AlphaZero' to Develop AI in Turn-Based Card Games. https://www.gdcvault.com/play/1027063/Machine-Learning-Summit-Applying-AlphaZero Accessed on February 10, 2022.

[14] Pei Li. 2021. Machine Learning Summit: 3D Parametric Face Model and Its Applications in Games. https://www.gdcvault.com/play/1027003/Machine-Learning-Summit-3D-Parametric Accessed on February 10, 2022.

[15] Yubin Liang, Wanxiang Li, and Kokolo Ikeda. 2019. Procedural Content Generation of Rhythm Games Using Deep Learning Methods. In *Entertainment Computing and Serious Games*, Erik van der Spek, Stefan Göbel, Ellen Yi-Luen Do, Esteban Clua, and Jannicke Baalsrud Hauge (Eds.). Springer International Publishing, Cham, 134–145.

[16] Zhiyu Lin, Kyle Xiao, and Mark Riedl. 2019. GenerationMania: Learning to Semantically Choreograph. arXiv:1806.11170 [cs.SD]

[17] Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. https://openreview.net/forum?id=Skq89Scxx

[18] Adam Finn Nogaj. 2005. *A genetic algorithm for determining optimal step patterns in Dance Dance Revolution*. Technical Report. Technical report, State University of New York at Fredonia.

[19] Karl O'Keeffe. 2003. *Dancing Monkeys (Automated creation of step files for Dance Dance Revolution)*. Technical Report. Imperial College London, London, United Kingdom.

[20] Kyungjin Park, Bradford W. Mott, Wookhee Min, Kristy Elizabeth Boyer, Eric N. Wiebe, and James C. Lester. 2019. Generating Educational Game Levels with Multistep Deep Convolutional Generative Adversarial Networks. In *2019 IEEE Conference on Games (CoG)*. IEEE, London, UK, 1–8. https://doi.org/10.1109/CIG.2019.8848085

[21] Jan Schlüter and Sebastian Böck. 2014. Improved musical onset detection with Convolutional Neural Networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Florence, Italy, 6979–6983. https://doi.org/10.1109/ICASSP.2014.6854953

[22] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2018. Procedural Content Generation via Machine Learning (PCGML). *IEEE Transactions on Games* 10, 3 (2018), 257–270. https://doi.org/10.1109/TG.2018.2846639

[23] Adam Tilson and Craig M Gelowitz. 2019. Towards Generating Image Assets Through Deep Learning for Game Development. In *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*. IEEE, Edmonton, AB, Canada, 1–4. https://doi.org/10.1109/CCECE.2019.8861965

[24] Yudai Tsujino and Ryosuke Yamanishi. 2018. Dance Dance Gradation: A Generation of Fine-Tuned Dance Charts. In *Entertainment Computing - ICEC 2018 - 17th IFIP TC 14 International Conference, Held at the 24th IFIP World Computer Congress, WCC 2018, Poznan, Poland, September 17-20, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11112)*, Esteban Clua, Licinio Roque, Artur Lugmayr, and Pauliina Tuomi (Eds.). Springer, Poznan, Poland, 175–187. https://doi.org/10.1007/978-3-319-99426-0_15

[25] Karen Ullrich, Jan Schlüter, and Thomas Grill. 2014. Boundary Detection in Music Structure Analysis using Convolutional Neural Networks. In *Proceedings of the 15th International Society for Music Information Retrieval Conference, ISMIR 2014, Taipei, Taiwan, October 27-31, 2014*, Hsin-Min Wang, Yi-Hsuan Yang, and Jin Ha Lee (Eds.). International Society for Music Information Retrieval, Taipei, Taiwan, 417–422. http://www.terasoft.com.tw/conf/ismir2014/proceedings/T075_271_Paper.pdf

[26] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M. Lucas, Adam M. Smith, and Sebastian Risi. 2018. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*, Hernán E. Aguirre and Keiki Takadama (Eds.). ACM, Kyoto, Japan, 221–228. https://doi.org/10.1145/3205455.3205517

A INCLUSION RATES OF ADDITIONAL DATASETS

Tables 4 and 5 shows inclusion rates between charts of different difficulty modes for each dataset. Looking at the upper-right triangle of the tables, we can see that charts created in our company have higher inclusion rates.

Table 4: Inclusion rates between difficulties of "Utapri".

s \ t	Beginner	Intermediate	Advanced	Expert	Challenge
Beginner	100%	91.0%	91.6%	92.4%	94.7%
Intermediate	60.2%	100%	93.9%	94.3%	96.6%
Advanced	37.8%	58.9%	100%	96.1%	96.5%
Expert	25.9%	40.4%	65.8%	100%	96.7%
Challenge	18.9%	30.2%	47.2%	69.7%	100%

Table 5: Inclusion rates between difficulties of "Stepmania".

s \ t	Beginner	Easy	Medium	Hard	Challenge
Beginner	100%	91.2%	90.2%	88.8%	86.5%
Easy	37.1%	100%	88.4%	86.3%	84.5%
Medium	23.8%	57.2%	100%	87.1%	84.7%
Hard	16.0%	37.9%	59.1%	100%	85.8%
Challenge	11.0%	27.4%	42.5%	63.8%	100%

B DETAILS OF THE EXPERIMENTAL RESULTS

In this chapter, we present the details of the experimental results. The experimental results include those for three more datasets than "Love Live! All Stars", which are introduced in section 4; "Utapri", "Stepmania Fraxtil" and "Stepmania ITG". Note that "Stepmania Fraxtil" and "Stepmania ITG" are combined and then trained and evaluated. They are denoted as "Stepmania" in tables below.

Table 6 shows the performance of the present model, GenéLive!, for each dataset.

Table 6: Performance of the present model (GenéLive!).

Dataset	Difficulty	F-score^m (std.)	F-score^c (std.)
Love Live! All Stars	Beginner	0.8643 (0.0018)	0.8627 (0.0018)
	Intermediate	0.7953 (0.0024)	0.7949 (0.0020)
	Advanced	0.7885 (0.0017)	0.7878 (0.0021)
	Expert	0.7784 (0.0062)	0.7796 (0.0060)
	all	0.8021 (0.0016)	0.8104 (0.0013)
Utapri	Beginner	0.6701 (0.0083)	0.6664 (0.0100)
	Intermediate	0.7799 (0.0055)	0.7732 (0.0058)
	Advanced	0.7634 (0.0024)	0.7630 (0.0031)
	Expert	0.7994 (0.0036)	0.7904 (0.0034)
	Challenge	0.8563 (0.0059)	0.8563 (0.0059)
	all	0.7656 (0.0042)	0.7400 (0.0043)
Stepmania	Beginner	0.7556 (0.0127)	0.7432 (0.0132)
	Intermediate	0.7271 (0.0029)	0.7198 (0.0027)
	Advanced	0.7442 (0.0024)	0.7442 (0.0025)
	Expert	0.7349 (0.0022)	0.7317 (0.0029)
	Challenge	0.7734 (0.0033)	0.7717 (0.0025)
	all	0.7451 (0.0019)	0.7354 (0.0037)

Table 7 shows the performance of models trained without the beat guide. Compared to the present model, the performance is significantly degraded in all datasets. Furthermore, the degradation is more significant in easier difficulty modes.

Table 7: Performance of GenéLive! without the beat guide.

Dataset	Difficulty	F-score^m (std.)	F-score^c (std.)
Love Live! All Stars	Beginner	0.8039 (0.0063)	0.8038 (0.0065)
	Intermediate	0.7597 (0.0043)	0.7612 (0.0043)
	Advanced	0.7586 (0.0077)	0.7603 (0.0074)
	Expert	0.7620 (0.0110)	0.7639 (0.0107)
	all	0.7645 (0.0067)	0.7699 (0.0052)
Utapri	Beginner	0.4506 (0.0243)	0.4227 (0.0252)
	Intermediate	0.6514 (0.0134)	0.6447 (0.0132)
	Advanced	0.6825 (0.0092)	0.6832 (0.0091)
	Expert	0.7339 (0.0099)	0.7279 (0.0106)
	Challenge	0.8213 (0.0121)	0.8213 (0.0121)
	all	0.6732 (0.0106)	0.6156 (0.0137)
Stepmania	Beginner	0.5959 (0.0165)	0.5926 (0.0165)
	Intermediate	0.6874 (0.0081)	0.6788 (0.0079)
	Advanced	0.7224 (0.0046)	0.7212 (0.0038)
	Expert	0.7309 (0.0046)	0.7288 (0.0046)
	Challenge	0.7755 (0.0039)	0.7744 (0.0040)
	all	0.7297 (0.0039)	0.6894 (0.0043)

Table 8 shows the performance of models which is trained and evaluated separately for each difficulty mode. In "Utapri" dataset, the performance is better when a single model is trained with charts of all difficulty modes. However, "Stepmania" dataset didn't reproduce the result. We are thinking that this is due to lower inclusion rate of charts in "Stepmania" than that of charts created in our company, comparing tables 2, 4 and 5.

Table 8: Performance of GenéLive! trained difficulty-wise.

Dataset	Difficulty	F-score^m (std.)	F-score^c (std.)
Love Live! All Stars	Beginner	0.8575 (0.0054)	0.8549 (0.0057)
	Intermediate	0.7873 (0.0042)	0.7867 (0.0043)
	Advanced	0.7766 (0.0036)	0.7765 (0.0037)
Utapri	Beginner	0.6405 (0.0108)	0.6368 (0.0146)
	Intermediate	0.7672 (0.0064)	0.7644 (0.0075)
	Advanced	0.7593 (0.0040)	0.7605 (0.0044)
	Expert	0.7902 (0.0031)	0.7814 (0.0049)
Stepmania	Beginner	0.7602 (0.0022)	0.7506 (0.0024)
	Intermediate	0.7288 (0.0039)	0.7228 (0.0033)
	Advanced	0.7474 (0.0028)	0.7470 (0.0026)
	Expert	0.7303 (0.0046)	0.7271 (0.0046)
	Challenge	0.7765 (0.0041)	0.7723 (0.0042)

Table 11 shows the performance of models with the DDC's conv-stack in place of our multi-scale conv-stack. We can see that there are certain performance improvements when the newer conv-stack is adopted, in all datasets.

C MODEL ARCHITECTURES

Tables 9 and 10 show the conv-stack architecture of the DDC and our GenéLive!

Table 9: Multi-scale conv-stack of GenéLive!

Block	Conv stack 1	Conv stack 2	Conv stack 3
Convolution 1	Kernel 3x3, Channels 48, Padding 1	Kernel 3x3, Channels 48, Padding 1	Kernel 3x3, Channels 48, Padding 1
	Batch Norm (48)	Batch Norm (48)	Batch Norm (48)
	ReLU	ReLU	ReLU
Convolution 2	Kernel 3x3, Channels 48, Padding 1	Kernel 3x3, Channels 48, Padding 1	Kernel 3x3, Channels 48, Padding 1
	Batch Norm (48)	Batch Norm (48)	Batch Norm (48)
	ReLU	ReLU	ReLU
	MaxPool (1, 4)	MaxPool (4, 4)	MaxPool (4, 4)
	Dropblock(0.25, 5, 0.25)	Dropblock(0.25, 5, 0.25)	Dropblock(0.25, 5, 0.25)
Convolution 3	Kernel 3x3, Channels 96, Padding 1	Kernel 3x3, Channels 96, Padding 1	Kernel 3x3, Channels 96, Padding 1
	Batch Norm (96)	Batch Norm (96)	Batch Norm (96)
	ReLU	ReLU	ReLU
	MaxPool (1, 4)	MaxPool (2, 4)	MaxPool (4, 4)
	Dropblock (0.25, 3, 1)	Dropblock (0.25, 3, 1)	Dropblock (0.25, 3, 1)
Convolution 4	Kernel 3x3, Channels 384, Padding 1	Kernel 3x3, Channels 192, Padding 1	Kernel 3x3, Channels 192, Padding 1
	Batch Norm (384)	Batch Norm (192)	Batch Norm (192)
	ReLU	ReLU	ReLU
	MaxPool (1, 4)	MaxPool (1, 4)	MaxPool (1, 4)
	AvgPool(1,3)	AvgPool(1,3)	AvgPool(1,3)
Up Sampling		Upsample (8, 1)	Upsample (16, 1)
Functional 1	Dropout(0.5)		
Bi-LSTM	770 Features, 2 Layers, Bidirectional		
	Dropout(0.5)		
Functional 2	Linear 768, 1		
	Sigmoid		

Table 11: Performance of GenéLive! with DDC's conv-stack.

Dataset	Difficulty	F-score^m (std.)	F-score^c (std.)
Love Live! All Stars	Beginner	0.8537 (0.0040)	0.8531 (0.0038)
	Intermediate	0.7842 (0.0040)	0.7844 (0.0037)
	Advanced	0.7733 (0.0059)	0.7733 (0.0064)
	Expert	0.7769 (0.0104)	0.7800 (0.0107)
	all	0.7895 (0.0037)	0.7992 (0.0034)
Utapri	Beginner	0.6771 (0.0113)	0.6755 (0.0126)
	Intermediate	0.7773 (0.0052)	0.7697 (0.0059)
	Advanced	0.7587 (0.0039)	0.7578 (0.0042)
	Expert	0.7925 (0.0024)	0.7844 (0.0028)
	Challenge	0.8576 (0.0040)	0.8576 (0.0040)
	all	0.7613 (0.0048)	0.7390 (0.0073)
Stepmania	Beginner	0.7516 (0.0337)	0.7415 (0.0344)
	Intermediate	0.7241 (0.0070)	0.7211 (0.0079)
	Advanced	0.7426 (0.0073)	0.7421 (0.0064)
	Expert	0.7318 (0.0032)	0.7292 (0.0051)
	Challenge	0.7711 (0.0036)	0.7679 (0.0028)
	all	0.7412 (0.0045)	0.7342 (0.0122)

Table 12: Performance of GenéLive! without the fuzzy label.

Dataset	Difficulty	F-score^m (std.)	F-score^c (std.)
Love Live! All Stars	Beginner	0.8541 (0.0044)	0.8527 (0.0041)
	Intermediate	0.7861 (0.0045)	0.7856 (0.0046)
	Advanced	0.7780 (0.0028)	0.7777 (0.0028)
	Expert	0.7780 (0.0050)	0.7796 (0.0049)
	all	0.7918 (0.0025)	0.8005 (0.0029)

Table 10: The DDC's conv-stack.

Block	Layers & Parameters
Convolution 1	Kernel 3x3, Channels 48, Padding 1
	Batch Norm (48)
	ReLU
Convolution 2	Kernel 3x3, Channels 48, Padding 1
	Batch Norm (48)
	ReLU
	MaxPool (1,2)
	Dropout (0.25)
Convolution 3	Kernel 3x3, Channels 96, Padding 1
	Batch Norm (96)
	ReLU
	MaxPool (1,2)
	Dropout(0.25)
Functional 1	Linear 96*input//4, 48x16
	Dropout(0.5)
Bi-LSTM	48x96+1+1, 48x8, 2 Layers
Functional 2	Linear 48x16, 1
	Dropout (0.5)
	Sigmoid

D EVALUATION ON FUZZY LABEL

Table 12 shows the evaluation result when the fuzzy label is not applied to label data during training. The experimental results reproduced the results of the previous study [15] that *fuzzy label* improves performance on imbalanced datasets.