MusPyExpress: Extending MusPy with Enhanced Expression Text Support

Phillip Long[1] p1long@ucsd.edu
Hao-Wen Dong[2] hwdong@umich.edu
Julian McAuley[1] jmcauley@ucsd.edu
Zachary Novack[1] znovack@ucsd.edu

[1]Computer Science & Engineering Department, University of California, San Diego
[2]Department of Performing Arts Technology, University of Michigan, Ann Arbor

Abstract

Current work in modeling symbolic music primarily relies on representations extracted from MIDI-like data. While such formats allow for modeling symbolic music as sequences of notes, they omit the large space of symbolic annotations common in western sheet music broadly known as *expression text*, such as tempo or dynamics, which specify time- and velocity-dependent controls on the musical composition and performance. To alleviate this gap, we present **MusPyExpress**, an extension to the popular symbolic music processing library MusPy [6] that enables the extraction of expression text along with symbolic music for downstream modeling. Utilizing this extension, we parse the PDMX dataset [14, 24] to illustrate the wealth of expression text available in MusicXML datasets. Additionally, we introduce multiple generative tasks, including joint expression-note generation, expression-conditioned music generation, and expression tagging, that take advantage of this additional notational information.

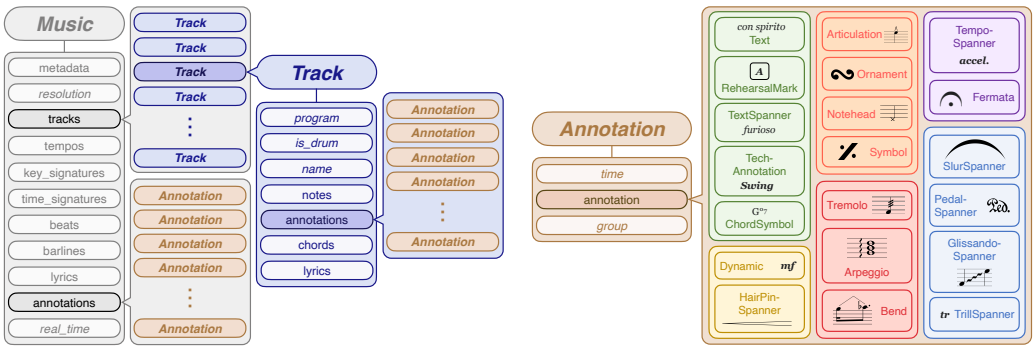
1 Introduction

Over the past decade, progress in modeling symbolic music, including tasks like symbolic music generation [7, 23, 5], transcription [10], and understanding [12, 1], has been predominantly driven by the popular MIDI [19] file representation. While MIDI is useful for modeling symbolic music as sequences of musical notes, there are a number of limitations to the encoding itself as well as the symbolic music processing libraries used to read it. Importantly, although MIDI is able to encode expression-adjacent features like time signature and tempo (in BPM), it omits most **expression text** present in western sheet music, such as tempo text (e.g., *adagio*), dynamic crescendos, section markings, and note articulations. Expression text is a key part of how music is communicated through sheet music, acting as a roadmap to the larger piece on *how* the sequence of notes should be played. In contrast to MIDI, the MusicXML format is designed with a focus on rendering sheet music for *reading* rather than listening, so it often contains an abundance of this expression text. Despite a recent surge in the number of MusicXML datasets [9, 16, 14, 24], MIDI remains the dominant standard for symbolic music data [11, 17, 22, 4, 2, 25, 8]. As a result, existing symbolic music processing libraries [6, 3, 13, 15] are generally optimized for the MIDI format and unequipped to harness the wealth of expression text available in MusicXML.

To remedy this gap, we introduce **MusPyExpress**, an extension of the popular symbolic music processing library MusPy [6] that processes expression text into discrete tensor data for downstream expression-aware sequence modeling tasks [21]. As a test bed, we utilize MusPyExpress to analyze the Public Domain MusicXML (PDMX) dataset [14, 24], the largest available MusicXML dataset. Additionally, we show how MusPyExpress enables a number of expression-aware symbolic music tasks, including joint expression-note generation, expression-conditioned music generation, and note-to-expression sequence tagging. MusPyExpress is available on GitHub as a branch of MusPy [1].

[1] https://github.com/salu133445/muspy/tree/expressive

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: AI for Music.



(a) *Music* Object Overview (b) Annotations in MusPyExpress

Figure 1: **Overview of MusPyExpress**. Although the format of the base *Music* object is the same as MusPy's [6], we have significantly expanded upon the annotations feature (which exists at both the score- and track-level), adding 28 new annotation types, the most of common of which are shown here. The boolean `real_time` attribute dictates whether temporal features such as onset or duration are stored in units of time steps (metrical time) or seconds (real time).

2 Related Works

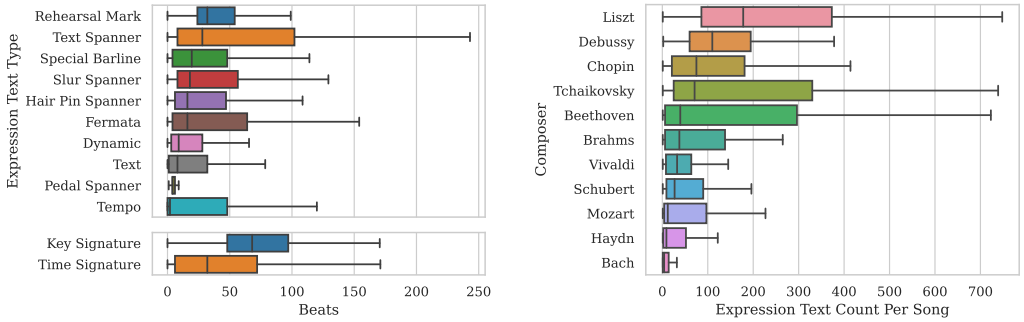
Several libraries support MusicXML for symbolic music processing, but they differ in focus and capabilities. music21 [3] is one of the most well-known toolkits, and despite extensive support for parsing expression text, these annotations are often embedded deep within its object hierarchy, making them cumbersome to extract for downstream modeling tasks. symusic [13] also supports MusicXML, but its primary design goal is efficient MIDI parsing rather than broad symbolic music representation, and the library was not developed with an emphasis on capturing expression text. jSymbolic [15] focuses on extracting statistical features from symbolic music, making it valuable for classification tasks but less suited for flexible input-output handling across formats. Despite these viable alternatives, our MusPyExpress library not only supports MusicXML but also emphasizes exposing expression text in a Python-based, lightweight, and accessible manner to facilitate downstream modeling.

3 MusPyExpress

We design an extension of the Python library MusPy [6], a package built to handle various forms of symbolic music with a universal *Music* object, which we term **MusPyExpress**. Given a piece of symbolic music, the resulting Music object can contain information on song layout, metadata, individual parts, time signatures, key signatures, tempo markings, notes, and bar boundaries. However, MusPy fails to parse most expression text, like dynamics, slurs, and articulations, and we design MusPyExpress to address this shortcoming. A Music object has the capacity to store "annotations," a catchall term for any information unrelated to the notes in a song or track. MusPyExpress expands this functionality, and creates a multitude of different Python objects, representing different types of expression text summarized in Figure 1, meant for storage within an annotation. Annotations can be stored in two places: at the score-wide level, generally reserved for temporally-related expression text and rehearsal marks, or the track-specific level, where dynamics and individual text instructions reside. In our implementation, we use the term "spanner" to refer to expression text that "spans" an explicit duration. For instance, a *TempoSpanner* object, like an *accelerando*, differs from a *Tempo* object, such as a *vivace* marking. Because MusicXML does not constrain creators to a strict set of expression text options, we have implemented the collection of the most common types of expression text. We further discuss the contributions of MusPyExpress in Appendix A.

4 Dataset Analysis

We utilize the Public Domain MusicXML (PDMX) dataset [14, 24] as a test bed for MusPyExpress in order to illustrate the wealth of available expression text in MusicXML datasets. Of the original 222,820 MusicXML files in PDMX, 212,406 (95.33%) contain expression text, comprising >4M total expression text markings and averaging 20.1 markings per song. We include further analysis of PDMX's expression text distribution in Appendix B.



(a) Relative Density of Expression Text

(b) Expression Text Usage by Composer

Figure 2: **Expression Text in PDMX**. Figure 2a shows boxplots of relative expression text density. Throughout this work, "Special Barline" refers to special barline types like double and dotted barlines and excludes the ordinary single barline. We include the relative density of key and time signature changes for reference. Figure 2b shows boxplots of expression text usage (i.e. the number of different expression text markings) per song for a select group of composers.

The average *absolute* expression text density (i.e. the metrical distance between two expression text markings, regardless of type) in PDMX is 17.21 beats. Meanwhile, the *relative* density over time for each expression text type (i.e. the distance in beats between two expression markings of the same type) is described in Figure 2a. Songs tend to avoid overusing structural control expression types, as doing so diminishes their effect, while expression types like tempos, dynamics, and text features can be quite concentrated, even more so for note-level annotations like pedal markings. Rehearsal marks exhibit a sparser distribution than their normal text counterparts, as these markings are reserved for demarcating different sections of a song. We also make use of PDMX's composer metadata to analyze expression text usage by composer in Figure 2b, finding that Romantic composers include expression text more frequently than their Baroque and Classical counterparts. This aligns with historical practice, since Romantic composers often explicitly encoded expression markings to emphasize individual expressivity, while Baroque and Classical composers often relied more on performance conventions of their time, resulting in fewer expression markings.

5 Experiments

Model	**PCE**	**SC** (%)	**GC** (%)	**Perplexity** (↓)
Ground Truth	2.68 ± 0.00	97.40 ± 0.01	93.67 ± 0.01	-
Baseline (*M*)	2.82 ± 0.02	93.31 ± 0.45	$\mathbf{93.87 \pm 0.21}$	2.80
Joint, Prefix (*M*)	1.75 ± 0.07	95.57 ± 0.45	97.41 ± 0.19	**2.64**
Joint, Anticipation (*M*)	$\mathbf{2.72 \pm 0.03}$	92.77 ± 0.46	93.93 ± 0.26	3.58
Conditional, Prefix (*M*)	1.83 ± 0.06	94.91 ± 0.44	94.43 ± 0.32	3.07
Conditional, Anticipation (*M*)	1.58 ± 0.05	$\mathbf{96.72 \pm 0.34}$	97.80 ± 0.14	3.02
Baseline (*R*)	$\mathbf{2.56 \pm 0.03}$	95.87 ± 0.36	92.19 ± 0.37	4.38
Joint, Prefix (*R*)	1.74 ± 0.07	$\mathbf{98.61 \pm 0.17}$	95.50 ± 0.36	3.79
Joint, Anticipation (*R*)	2.22 ± 0.05	95.16 ± 0.49	95.47 ± 0.28	**3.52**
Conditional, Prefix (*R*)	2.37 ± 0.05	92.93 ± 0.44	92.17 ± 0.48	4.05
Conditional, Anticipation (*R*)	2.40 ± 0.05	94.08 ± 0.36	$\mathbf{94.48 \pm 0.31}$	4.43

Table 1: **Evaluation Results**. Objective evaluation results for joint (1) and conditional (2) generation tasks. We evaluate the same metrics proposed in Multitrack Music Transformer (MMT) [5] – pitch class entropy (PCE), scale consistency (SC) and groove consistency (GC) – where a closer value to that of the ground truth is considered better. We also evaluate each model's perplexity over notes (i.e. ignoring expression text), where lower is better.

MusPyExpress enables a number of symbolic music modeling tasks that take into account the sequence of expression text features. We focus on modeling specific *tracks* rather than full scores, which we leave for future work. At a high level, a track is comprised of a sequence of notes

Model	Beat	Position	Time	Value	Duration	Total
Baseline (M)	11.36	86.34	-	1.26	14.68	0.05
Prefix (M)	**62.80**	**93.55**	-	**55.60**	**54.15**	**29.41**
Anticipation (M)	59.78	92.73	-	50.43	50.11	24.41
Baseline (R)	-	-	4.17	13.86	59.02	1.63
Prefix (R)	-	-	**46.05**	**57.39**	**66.54**	**33.90**
Anticipation (R)	-	-	40.25	53.32	65.60	27.75

Table 2: **Expression Tagging Results**. Per-field accuracy of expression tagging (3) models on validation set. As a baseline, we examine a model that simply predicts the most common value of each field. Baseline differences between the metrical and real timing schemes may arise because the real time representation truncates songs at 60 seconds, excluding expression text in longer pieces and thereby altering the distribution of field values.

$\boldsymbol{n} = (\boldsymbol{n}_{1:N})$ and expression text tokens $\boldsymbol{e} = (\boldsymbol{e}_{1:M})$. Using the subset of PDMX *tracks* that contain annotations, we propose three different generation tasks within this framework:

1. **Joint Note-Expression Text Generation**: We model *both* sequences together $p_\theta(\boldsymbol{n}, \boldsymbol{e})$ and thus produce a model that can generate both notes and expression text. This task mirrors how composers actually write music, adding new notes and expression text simultaneously.
2. **Expression-Conditioned Note Generation**: We modify the standard symbolic music generation task with expression text annotations $p_\theta(\boldsymbol{n} \mid \boldsymbol{e})$, learning a model to produce notes *given* a sequence of expression text tokens. This task could prove useful in film music generation and sound design for games and audiobooks, where it is often useful to match music to the climax, tension, and rhythm of the inputs.
3. **Expression Tagging**: Given an existing note sequence $\boldsymbol{n}$, we learn a model $p_\theta(\boldsymbol{e} \mid \boldsymbol{n})$ that tags the sequence with expression text $\boldsymbol{e}$ (i.e. add expression text to a planned musical score). This task serves as an intermediate step in the MIDI to performance rendering of a musical score; given only a plain MIDI file, a trained model could suggest appropriate expression markings such as *legato*, *crescendo*, or *dolce*, enabling automatic annotation of extensive collections of unexpressive MIDI data widely available in large-scale MIDI datasets.

We refer to the second and third experiments as "conditional," and each experiment is performed in both the metrical (*M*) and real (*R*) time domains. We discuss different data representations used, interleaving strategies, and experimental setup for each task in Appendix C.

Joint/Expression-Conditioned Results Table 1 compares our models trained on different timing and conditioning schemes against a *note-only* MMT [5] baseline. No model configuration clearly dominates across all metrics (which may be an artifact of our small model size). However, we find our best overall perplexity (i.e. a model's certainty in its predictions) in the joint prefix-conditioned metrical-time model, providing evidence that expression tokens can provide useful local information for realistic music generation.

Expression Tagging Results In Table 2, we show accuracy results for expression tagging with both timing and interleaving schemes. Notably, we find that the prefix-conditioned models, especially those in real time, clearly dominate the anticipatory models, as the prefix setup is much simpler when predicting the sparse sequence of expression tokens (which anticipatory models can struggle on [20]).

6 Conclusion

We present **MusPyExpress**, an extension to the popular symbolic music processing library MusPy [6] that supports expression text processing for use in downstream modeling tasks. We use the PDMX [14, 24] dataset as a test bed for our extension to show the abundance of available expression text in MusicXML datasets. We then employ the subset of PDMX tracks containing these annotations for three generative tasks – joint note-expression text generation, expression-conditioned note generation, and expression tagging – illustrating how MusPyExpress enables expression-aware downstream modeling. In the future, we plan to further explore expression-tagging systems that would allow us to annotate large, unexpressive MIDI datasets with expression text recommendations, as well as expand expression-conditioned generation into full fine-grained text-to-music generation.

References

[1] Yi-Hui Chou, I Chen, Chin-Jui Chang, Joann Ching, Yi-Hsuan Yang, et al. manrt-piano: large-scale pre-training for symbolic music understanding. *arXiv preprint arXiv:2107.05223*, 2021.

[2] Léopold Crestel, Philippe Esling, Lena Heng, and Stephen McAdams. A database linking piano and orchestral midi scores with application to automatic projective orchestration. In *Proceedings of 18th International Conference on Music Information Retrieval, ISMIR*, 2017.

[3] Michael Scott Cuthbert and Christopher Ariza. music21: A toolkit for computer-aided musicology and symbolic music data. 2010.

[4] Chris Donahue, Huanru Henry Mao, Yiting Ethan Li, Garrison W Cottrell, and Julian McAuley. Lakhnes: Improving multi-instrumental music generation with cross-domain pre-training. *arXiv preprint arXiv:1907.04868*, 2019.

[5] Hao-Wen Dong, Ke Chen, Shlomo Dubnov, Julian McAuley, and Taylor Berg-Kirkpatrick. Multitrack music transformer. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

[6] Hao-Wen Dong, Ke Chen, Julian McAuley, and Taylor Berg-Kirkpatrick. Muspy: A toolkit for symbolic music generation. *Proceedings of the 21st International Society for Music Information Retrieval Conference, ISMIR*, 2020.

[7] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *AAAI Conference on Artificial Intelligence*, number 1, 2018.

[8] Jeffrey Ens and Philippe Pasquier. Building the metamidi dataset: Linking symbolic and audio musical data. In *ISMIR*, pages 182–188, 2021.

[9] Francesco Foscarin, Andrew Mcleod, Philippe Rigaux, Florent Jacquemard, and Masahiko Sakai. Asap: a dataset of aligned scores and performances for piano transcription. In *ISMIR 2020-21st International Society for Music Information Retrieval*, 2020.

[10] Josh Gardner, Ian Simon, Ethan Manilow, Curtis Hawthorne, and Jesse Engel. Mt3: Multi-task multitrack music transcription. *arXiv preprint arXiv:2111.03017*, 2021.

[11] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, et al. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations (ICLR)*, 2019.

[12] Junyan Jiang, Ke Chen, Wei Li, and Gus Xia. Large-vocabulary chord transcription via chord structure decomposition. In *ISMIR*, pages 644–651, 2019.

[13] Yikai Liao and Zhongqi Luo. symusic: A swift and unified toolkit for symbolic music processing. In *Extended Abstracts for the Late-Breaking Demo Session of the 25th International Society for Music Information Retrieval Conference*, 2024.

[14] Phillip Long, Zachary Novack, Taylor Berg-Kirkpatrick, and Julian McAuley. PDMX: A large-scale public domain musicxml dataset for symbolic music processing. In *ICASSP 2025 - 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, 2025.

[15] Cory McKay and Ichiro Fujinaga. jsymbolic: A feature extractor for midi files. In *ICMC*, 2006.

[16] Silvan David Peter, Carlos Eduardo Cancino-Chacón, Francesco Foscarin, Andrew Philip McLeod, Florian Henkel, Emmanouil Karystinaios, and Gerhard Widmer. Automatic note-level score-to-performance alignments in the asap dataset. *Transactions of the International Society for Music Information Retrieval*, Jun 2023.

[17] Colin Raffel. *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. Columbia University, 2016.

[18] Yi Ren, Jinzheng He, Xu Tan, Tao Qin, Zhou Zhao, and Tie-Yan Liu. Popmag: Pop music accompaniment generation. In *Proceedings of the 28th ACM international conference on multimedia*, pages 1198–1206, 2020.

[19] Joseph Rothstein. *MIDI: A comprehensive introduction*. AR Editions, Inc., 1992.

[20] John Thickstun, David Hall, Chris Donahue, and Percy Liang. Anticipatory music transformer. *arXiv preprint arXiv:2306.08620*, 2023.

[21] 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*, 2017.

[22] Ziyu Wang, Ke Chen, Junyan Jiang, Yiyi Zhang, Maoran Xu, Shuqi Dai, Xianbin Gu, and Gus Xia. Pop909: A pop-song dataset for music arrangement generation. *Proceedings of 21st International Conference on Music Information Retrieval, ISMIR*, 2020.

[23] Shih-Lun Wu and Yi-Hsuan Yang. The jazz transformer on the front line: Exploring the shortcomings of ai-composed music through quantitative measures. *arXiv preprint arXiv:2008.01307*, 2020.

[24] Weihan Xu, Julian McAuley, Taylor Berg-Kirkpatrick, Shlomo Dubnov, and Hao-Wen Dong. Generating symbolic music from natural language prompts using an llm-enhanced dataset. *arXiv preprint arXiv:2410.02084*, 2024.

[25] Mingliang Zeng, Xu Tan, Rui Wang, Zeqian Ju, Tao Qin, and Tie-Yan Liu. Musicbert: Symbolic music understanding with large-scale pre-training. *arXiv preprint arXiv:2106.05630*, 2021.

[26] Hongyuan Zhu, Qi Liu, Nicholas Jing Yuan, Chuan Qin, Jiawei Li, Kun Zhang, Guang Zhou, Furu Wei, Yuanchun Xu, and Enhong Chen. Xiaoice band: A melody and arrangement generation framework for pop music. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2837–2846, 2018.

A Contributions of MusPyExpress

A.1 Type of Expression Text

MusPyExpress introduces 28 Python classes, representing different expression text types commonly found in western symbolic music. These classes are meant for storage as the `annotation` attribute inside of MusPy's more general *Annotation* object. Broadly, we can group these expression text types into four categories: text, structural controls, note-level annotations, and general-purpose abstractions.

Text-based expressions include rehearsal marks and natural language annotations such as *con spirito*.

- *ChordSymbol*: chord labels such as "Cmaj7."
- *RehearsalMark*: rehearsal indicators like "A" or "Verse."
- *Text*: general text annotations.
- *TextSpanner*: extended text annotations spanning multiple notes.

Structural controls encompass the broadest range, including tempo and dynamic markings, time and key signature changes, barlines, and *fermatas*.

- *ChordLine*: horizontal line indicating repeated chord structure.
- *Dynamic*: dynamic markings such as *mf* or *ff*.
- *Fermata*: sustained pause on a note or rest.
- *HairPinSpanner*: crescendo or diminuendo markings.
- *OttavaSpanner*: notation for playing passages one or more octaves higher or lower.
- *Symbol*: general score symbol or other graphical markers.
- *TempoSpanner*: tempo markings spanning multiple notes such as *ritardando* and *accelerando*.

Note-level annotations include articulations, slurs, and pedal markings, which can indicate unique stylistic passages when found in clusters.

- *Arpeggio*: broken chord indication.
- *Articulation*: markings such as *staccato* or accent.
- *Bend*: pitch bend for fretted instruments.
- *GlissandoSpanner*: sliding motion between two notes.
- *Notehead*: alternate notehead shapes.
- *Ornament*: decorative figures like mordents or turns.
- *PedalSpanner*: sustain pedal or similar keyboard pedal markings.
- *SlurSpanner*: slurs connecting multiple notes.
- *TechAnnotation*: instrument-specific technique annotations such as *mute* for a trumpet.
- *Tremolo*: rapid repetition of a note or alternation between notes.
- *TremoloBar*: whammy-bar notations for guitar.
- *TrillSpanner*: rapid alternation between two adjacent notes.
- *VibratoSpanner*: vibrato or wavy-line indications.

General-purpose abstractions are more for internal organization and storing miscellaneous expression text that do not fall neatly into the above categories.

- *Point*: structural indicator for positioning in a score.
- *Spanner*: general-purpose spanning annotation.
- *Subtype*: subtype label for higher-level categories.
- *SubtypeSpanner*: subtype annotation spanning multiple measures.

Beyond annotations, MusPyExpress also extends several of MusPy's base classes to better capture score-level details. In particular, it adds support for tempo marking text (in addition to raw BPM values), barline types for tracking special double or dotted barlines, and a grace-note attribute for notes to indicate *appoggiaturas* and *acciaccaturas*.

A.2 Timing

In MusPy, time is stored with a metrical time unit known as time steps. Each time step represents a fraction of a beat, and is therefore tempo agnostic. However, use often necessitates seconds-based encoding, so MusPyExpress includes functionality to convert to and from metrical and real time in a way that accounts for any temporally-related expression text in the music. This is valuable not only for realizing a score into MIDI or audio, but also for downstream modeling tasks where real time encoding makes it easier to align and interpret expression markings that are difficult to capture in tempo-agnostic time steps, such as tempo changes and *fermatas*. By supporting both metrical and real time representations, MusPyExpress provides more flexible handling of expression-aware symbolic music modeling.

A.3 File I/O

MusPyExpress also improves on MusPy's audio- and symbolic-output capabilities. MusPy was built to process a variety of symbolic music types, and has the ability to render these various file types as MIDI (symbolic domain) and WAV (audio domain). Native MusPy renderings are limited to notes with velocities inherited from the symbolic music source and, importantly, do not reflect any expression text dynamic markings. While MusPy renderings realize global tempo changes (e.g., BPM specifications), they do not account for local tempo variations such as *ritardando* or *accelerando*.

In MusPyExpress, audio and symbolic domain outputs now reflect any expression text markings present in a given Music object. For instance, note durations for a slurred musical section extend to the next note, and accented notes are markedly louder than their non-accented peers. Tempo-related expression text, like *ritardandos*, and volume-related features, like *crescendos*, are realized through changes in tempo and velocity, respectively. To translate such markings into concrete values, we use a set of configurable constants (e.g., slowing tempo $3\times$ for fermatas or increasing velocity $1.5\times$ for accents), which default to our manually curated values, in order to consistently map expression text into numerical adjustments.

Similar to MusPy, a MusPyExpress object can both import-from and export-to JSON format, which allows storage of these objects without any information loss.

B Analyzing PDMX with MusPyExpress

B.1 Parsing MusicXML-Adjacent Formats

MusPyExpress includes support for parsing not only standard MusicXML files but also MusicXML-adjacent formats that share similar structure and semantics. One such format is MSCZ (used by MuseScore), which differs only slightly in terminology from MusicXML but is otherwise structurally compatible. To demonstrate this, in addition to PDMX's MusicXML files, we also parse the original MSCZ files from which PDMX was derived to illustrate that MusPyExpress can read other symbolic music formats containing expression text.

B.2 Expression Text Usage in PDMX

PDMX contains >3.5M total expression text annotations. In Table 3, we dissect this total into counts for each available expression text type. Tempo and dynamic markings are the most prevalent categories, since every song requires at least one of each, whereas pedal and text spanners occur comparatively infrequently.

While some expression text types in PDMX have an explicit duration attribute, many, such as time and key signatures, do not, and so to analyze their distribution, we examine their implied duration, defined as the metrical distance to the next expression text of the same type, or to the end of the song if none follow. Figure 3 shows boxplots of of these explicit and implied durations (in beats) for different

Expression Text Type	Count
Tempo	1,082,640
Dynamic	1,069,492
Text	467,479
Barline	336,029
Rehearsal Mark	176,579
Hair Pin Spanner	164,014
Slur Spanner	86,764
Fermata	78,572
Pedal Spanner	48,421
Text Spanner	3,651
Total	3,513,641

Table 3: **Expression Text Frequencies**. Frequencies of different expression text types in PDMX. Tempo and dynamic markings are the most common, as every song requires at least one of each, while pedal and text spanners are the least frequent.

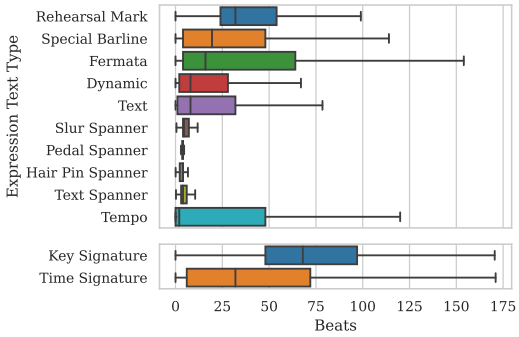


Figure 3: **Expression Text Durations**. Boxplots of durations (in beats) for different categories of expression text. Key signatures, rehearsal marks, and time signatures tend to span entire sections of songs, and thus have some of the longest durations. Pedal and text spanners, meanwhile, usually only apply to short sequences of notes and so are often quite brief. Tempo markings exhibit a long tail of extended durations, likely because many songs contain frequent tempo changes resulting in short local spans, while a few songs may have an initial tempo that persists across the entire piece.

expression text types. Expression types with implied durations, including dynamics, *fermatas*, and rehearsal marks, generally span entire sections and thus have longer durations, whereas explicitly-delimited markings like text spanners and pedal markings exhibit shorter lengths. For expression text types with implied duration, relative density (the metrical distance from one expression text to the next of the same type, examined in Figure 2a) is identical, and only in the case of explicit duration are relative density and implied duration different.

C Experiment Details

C.1 Data Representation

In this work, we use the tokenization scheme from Multitrack Music Transformer (MMT) [5]. Namely, we represent a song as a sequence of events $\boldsymbol{x}_i$ (either a note $\boldsymbol{n}_i$ or expression $\boldsymbol{e}_i$), where each event is represented by a tuple of variables. MMT uses a metrical timing system, where $\boldsymbol{x}_i$ has six fields: type, beat, position, pitch, duration, instrument. Songs in MMT encoding have a structural "preamble" starting each sequence, encoding the list of instruments and the start of notes.

To adapt the MMT framework for expression-aware modeling, we make a number of modifications. Most saliently, we encode expression text in the MMT framework by including an additional value in the type field to denote expression tokens, and convert MMT's pitch field into a *value* field, which

combines the standard vocabulary of 128 MIDI-pitches with nearly 700 expression text values (i.e. an expression token is a note token where its "pitch" denotes the type of expression text). Additionally, we provide an alternative *real* time encoding, where the beat and position fields are replaced by a single real time field, and temporally-related fields, namely time and duration, are encoded in seconds. We do this as many expression text types are temporal in nature, which would not be captured at the note-level in a metric system. Furthermore, we add a velocity field to support the learning of dynamic-related expressions.

C.2 Interleaving Methods

As we seek to model the sequence of notes and expression text together with an *autoregressive* model, how we interleave the notes and expression text into a single sequence $\boldsymbol{x} = \texttt{interleave}(\boldsymbol{n}, \boldsymbol{e})$ can make a large difference in performance [20]. Under each conditional experiment, "events" are conditioned on "controls." For example, expression text *controls* dictate the note *events* generated by an expression-conditioned model. In this work we test two different interleaving methods: prefix and anticipatory conditioning.

Prefix conditioning, used in most encoder-decoder models [26, 18], sets the control sequence as a *prefix* to the main event sequence. While simple, this forces the model to attend to long-term dependencies in order to model the relationship between events and controls that are close in time. Anticipation [20] seeks to remedy this fact and place controls close in the sequence to the events they affect. Specifically, under anticipatory conditioning, a control with onset time t is placed after the first event i with onset time t_i such that $t_i \geq t - \delta$ for some offset δ, which effectively interleaves the controls *within* the event sequence rather than separate from it (see [20] for an in depth description). For all experiments, we set $\delta = 8$, which is interpreted under metrical and real time schemes as beats and seconds, respectively.

C.3 Experimental Setup and Metrics

For all experiments, we use an MMT-style decoder-only transformer, with 6 layers, 8 attention heads and a hidden dimension of 512, totaling at about 20M parameters. We use an absolute positional embedding and set the maximum sequence length to 1024. We train each model for 80K steps with a batch size of 8, learning rate of 5e-4, with all experiments performed on a single NVIDIA GeForce RTX 3090 GPU, and use the Adam optimizer with default hyperparameters. For the joint and expression-conditioned experiments, we set the expression tokens as the controls for both interleaving methods. For expression tagging, we flip the roles and set the *notes* as the controls for interleaving. Note that all experiments are trained with the same next token prediction loss, the only difference being that the loss values of "control" tokens are zero-d out for the expression-conditioned (expression text) and expression tagging (notes) experiments. We use an 80%-10%-10% train-val-test split on the subset of 357,749 PDMX tracks that include at least one annotation.

For the joint and expression-conditioned generation tasks, we follow past work [5, 23, 20] and report the pitch class entropy, scale consistency, and groove consistency across 256 generations per model configuration, which measure how well the model captures the underlying musical patterns of the data, as well as the perplexity over notes (i.e. ignoring expression text) on the held out test set. Expression sequences for the expression-conditioned generations are pulled randomly from the test set. For the tagging task, we report per-field accuracy on predicted expression text, ignoring the type, instrument, and velocity fields, which generally achieve 100% accuracy by nature since our data representation encodes these fields as constant values in expression text events within single-track sequences.