A TRANSFORMER-BASED "SPELLCHECKER" FOR DETECTING ERRORS IN OMR OUTPUT

Timothy de Reuse **Ichiro Fujinaga**
Centre for Interdisciplinary Research in Music Media and Technology, McGill University
{timothy.dereuse, ichiro.fujinaga}@mcgill.ca

ABSTRACT

The outputs of Optical Music Recognition (OMR) systems require time-consuming human correction. Given that most of the errors induced by OMR processes appear "non-musical" to humans, we propose that the time to correct errors may be reduced by marking all symbols on a score that are musically unlikely, allowing the human to focus their attention accordingly. Using a dataset of Romantic string quartets, we train a variant of the Transformer network architecture on the task of classifying each symbol of an optically-recognized musical piece in symbolic format as correct or erroneous, based on whether a manual correction of the piece would require an insertion, deletion, or replacement of a symbol at that location. Since we have a limited amount of data with real OMR errors, we employ extensive data augmentation to add errors into training data in a way that mimics how OMR would modify the score. Our best-performing models achieve 99% recall and 50% precision on this error-detection task.

1. INTRODUCTION

An enormous amount of music scores exist only in the form of digital images, where its musical content is not machine-readable. Among the fields of research that that assist with large-scale processing of musical documents is Optical Music Recognition (OMR), which investigates how to transform images of music scores into symbolic music files (e.g., MusicXML files) that can be searched, played back, and edited. OMR has the potential to be a valuable tool for music archives and libraries, but it is not yet reliable enough; all methods require a human correction step after the OMR process, as the raw outputs of currently-available OMR algorithms contain too many errors to be acceptable for musicians or music researchers. This is an issue when using OMR on printed Western music notation, and is worse for other types of music notation, especially when training data is scarce. While custom interfaces exist for the purpose of facilitating human correction, such as the correction interface of MuRET [1], this is still a time-consuming task even for experts. While OMR speeds up symbolic encoding in many cases [2], the process of scanning documents, performing OMR, and correcting OMR is not always faster than that of transcribing pieces from scratch on complex, polyphonic scores [3].

We observe that OMR systems tend to introduce musically unlikely phenomena into scores. A typical example of the errors induced by the OMR process of PhotoScore [1] (a leading proprietary, commercial OMR software) is illustrated in Figure 1. In the OMR'ed score, consider the pattern of missing staccato articulations in measure 1, the eighth note misidentified as a sixteenth note in measure 2, the strangely placed grace note in measure 3, and the accidental misidentified as a grace note in measure 6. Also note how the slurs in each instrument of the OMR'ed score do not match up with each other, though the correct phrasing could probably be guessed from the melodic content alone. While not all of these phenomena are necessarily impossible in a string quartet, a human looking over the score would notice their presence.

Given the musical "incorrectness" of many of the errors introduced by OMR, we propose that a machine-learning algorithm could flag most OMR errors automatically given information on the conventions of the genre. This could reduce the correction time of a score; instead of checking every measure of OMR output against the corresponding measure in the score image, the user would only have to check those regions flagged by the algorithm as possibly erroneous. Even if the algorithm flags some non-erroneous regions of the score, it could potentially still exclude large portions of it from needing human review.

The reader may question why we aim for *detection* of errors as opposed to *correction* of errors outright; this could further reduce the time to correct OMR output even further, and there exists a large body of work on error correction for natural language that we could draw on. We posit, however, that any amount of correction that is not guaranteed to address nearly *all* errors in an OMR output would not significantly save time on the part of the human corrector, as they would still be required to manually review the result anyway. Current studies on correcting errors in polyphonic music (discussed in Section 2) do not attain high enough performance to meet this bar.

[1] /www.neuratron.com/photoscore.htm

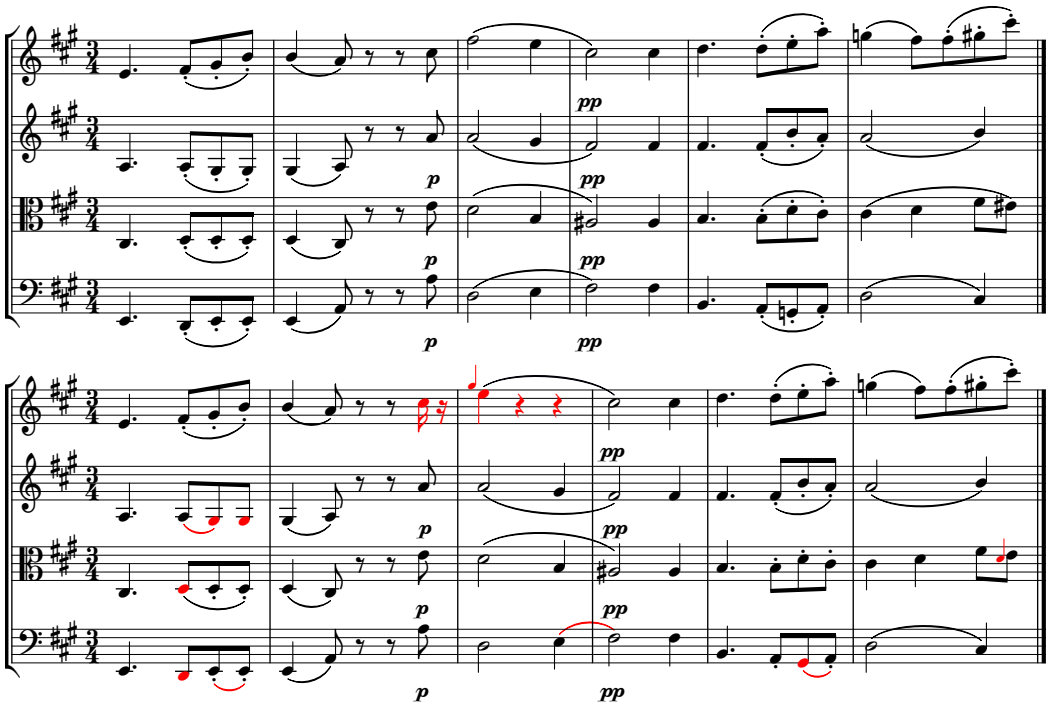


Figure 1: Two versions of Felix Mendelssohn's Quartet No. 2 in A Major, Op. 13, mm. 4-9. The upper system is an engraving of the correct score. The lower system is an output from PhotoScore's OMR Process. Some prominent errors are highlighted in red.

2. PREVIOUS WORK

There is little other research that seeks to detect or correct errors in symbolic music in the context of OMR. Most studies that discuss errors made by OMR systems seek to characterize them as a part of a method of evaluating their performance [4, 5]. As part of an end-to-end OMR system, Rossant and Blach [6] explicitly encode some musical rules (such as ensuring the number of beats in a measure matches the most recent time signature) that let them highlight areas that are likely incorrect, and also use confidence measures of detection steps from earlier in their OMR process to highlight areas that may have been detected incorrectly. More research focuses on error correction in music for other applications. McLeod et al. [7] focus on the task of analyzing errors in the output of Automatic Music Transcription (AMT) methods, introducing rule-based and machine leaning-based models to solve error detection and correction tasks, though these are intended as proof-of-concept baselines to define the tasks. Ycart et al. [8] compare a number of machine-learning models on the closely related task of Symbolic Music transduction, which involves processing the raw per-note probabilities output by an AMT model and producing a valid symbolic music file as output. Sidorov et al. [9] analyse music using formal grammars, taking the assumption that "correct" music tends to be highly compressible, and so any note whose presence alone significantly increases the amount of information in a piece is likely to be an error.

Significantly more literature exists in the fields of Grammar Error Detection (GED) and Grammar Error Correction (GEC) in natural language. The fields are far too broad to cover here; we direct readers to reviews on these subjects by Wang et al. [10], covering GEC, and Madi and Al-Khalifa [11], covering GED. Early models in both of these fields were rule-based, wherein every type of error had to be manually codified [12]. The current state of the art in these areas is in sequence-to-sequence deep-learning models similar to those used for machine translation, reaching human-level performance on common English-language GEC benchmarks [13]. To achieve this level of performance, however, a large amount of training data is required; there is much less research on GEC and GED in low-resource languages, the vast majority being on either English or Chinese [11]. Our task faces a similar issue; there exists orders of magnitude fewer training data for any one genre of music than there exists for the English language.

Data augmentation is commonly used in GEC and GED where data availability is low. Errors are added to correct examples and the models are then directed to correct those errors; we refer to errors created in this way as *synthetic errors* and errors created by the process of interest (e.g. errors induced by an OMR process) as *natural errors*. Early work on this process used simple statistics about distributions and types of errors to add errors semi-randomly [14, 15]. Later work, inspired by back-translation techniques from machine translation, trained models that *add* synthetic errors simultaneously with ones that *remove* them. Htut and Tetreault [16] evaluate a number of these models, finding that they provide significant improvements to training on natural errors when the dataset containing natural errors is small. Notably, they

find these gains in performance even when the synthetic errors are qualitatively different than the natural errors; that is, a GEC method can still perform well even when it is trained on grammatical errors that a human would never make. However, when a large dataset of natural errors is available, the improvements obtained by augmenting with synthetic errors are significantly diminished.

3. METHODOLOGY

In developing our method, we aim for it to have the following qualities:

- **High Recall Rate**: Our detection method should ideally make *no* false negatives; that is, whenever an error is present in its input, it highlights it. It is acceptable for the precision to be significantly lower than the recall as long as the user is assured that no errors lie outside the flagged regions.
- **Localization of Errors**: Within the constraint of having a high recall rate, the method should try to be as specific as possible when pointing out errors.
- **Trainable on a Small Dataset**: Many of the most common use-cases of OMR are on genres that do not have high-quality, large, digitized datasets on which we could train.
- **Takes Long Inputs**: Many errors introduced by OMR are unlikely to be identifiable in isolation from a small snippet of music, and may only be recognized on the basis of comparison with the rest of the piece.

3.1 Data Representation

What is "an error" in a musical score? Rigorously defining this notion requires some procedure to take the "difference" between two scores (in our case, an OMR output and a correct score). The characteristics of the errors that we find will depend heavily on the underlying representation we use. We also face the constraint that whatever representation we use must be suitable for input into a machine-learning model, preferably as a sequence; this disqualifies the use of methods that operate on hierarchically-structured MusicXML data [17, 18].

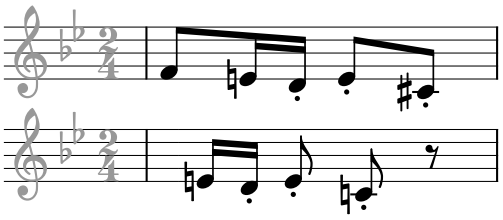
A piano roll-like representation, representing a snippet of music as a two-dimensional image, easily admits a difference operation by way of taking the pixel-wise difference of two two-dimensional piano rolls, but is memory-intensive for long inputs. Another possibility would be to use a token-based representation such as NoteTuple [19], which represents polyphonic music as an ordered list of multi-valued tokens; for example, a piece might be represented as a sequence of triples of the form `(MIDI Pitch, Onset Time, Duration)`. This kind of representation is common in deep learning applications, and there exist many notions of difference on one-dimensional sequences that could be adapted. However, to align with our goal of localizing errors, it would be better for each unit of our representation to contain as little information as possible, so that in calling a single element erroneous we pinpoint the location of the error down to a smaller region in the score.

To accomplish this, and to choose a representation that matches the domain of OMR, we opt to use an *agnostic representation* [20]. An agnostic representation of a musical score lists each of the symbols on a staff without considering their underlying musical meaning; this stands in contrast to *semantic representations* like MusicXML. An agnostic representation encodes less information per sequence element; a dotted eighth note with an accidental would typically be encoded by a single token in a representation like NoteTuple, whereas the accidental, the note, and the dot would each be a separate element in an agnostic representation, and each could be individually flagged as an error.

We write our own utility (included with the source code of this project) to convert Musicxml or Humdrum **kern files into an agnostic encoding by first parsing them with the `music21` package, and then heuristically defining an agnostic encoding that matches the resulting textttmusic21 stream. The encoding lists each symbol on a staff in order from left to right. When two or more symbols are in the same horizontal position (such as when multiple notes sound simultaneously) we list them in ascending order and add a caret symbol ^ between each of them. To handle multiple staves, we interleave them: a measure of the first violin, a measure of the second violin, the viola, the bass, and repeat. This scheme cannot encode all polyphonic musical scores into unique agnostic representations; staff lines where two rhythmically distinct polyphonic voices are notated on the same staff may contain unresolvable ambiguities, where it is not clear which notes belong to which voice. This is not an issue for our task, as we aim not to generate valid semantically-encoded scores from agnostic representations but only to *classify* notes in an existing score.

3.2 Sequence Alignment

To define a difference between two agnostically-encoded musical excerpts, we use the Needleman-Wunsch (NW) sequence alignment algorithm [21]. This algorithm takes as input two sequences and computes the shortest list of operations necessary to turn one into another, where an operation is defined as insertion, deletion, or replacement of a sequence element. By comparing a corrected score with its OMR'ed version, we can see *where* operations must be performed in order to correct it. Our strategy will be to train a model to answer the question: *Given a musical score that contains errors, where would we need to perform operations to correct those errors, according to a NW alignment?* We ignore information on *which* operations need to be performed. Preliminary experiments showed degradation in overall performance when our model was trained to identify the type of error along with its position. We surmise that to a human corrector, reliable detection of the location error itself is more important.



(a)

Original	OMR	Op.
barline.regular	barline.regular	-
8th.pos2.startBeam	–	I
accid.pos1.natural	accid.pos1.natural	-
16th.pos1.continueBeam	16th.pos1.startBeam	R
dot.pos-2	dot.pos-2	-
^	^	-
16th.pos0.endBeam	16th.pos0.endBeam	-
dot.pos-1	dot.pos-1	-
^	^	-
8th.pos1.startBeam	8th.pos1.noBeam	R
accid.pos-1.sharp	accid.pos-1.natural	R
dot.pos-3	dot.pos-3	-
^	^	-
8th.pos-1.endBeam	8th.pos-1.noBeam	R
–	8th.rest	D
barline.regular	barline.regular	-

(b)

Figure 2: (a): A correct score (top) and version with OMR errors (bottom): Mendelssohn's Quartet No. 1 in E♭ Major Op. 12, Mvt. 2, measure 10, 2nd violin part. (b): The Needleman-Wunsch alignment between the agnostic encodings of the two scores on the left. The **Op.** column shows the operations needed to correct the OMR column: **I**nsertion, **R**eplacement, and **D**eletion. Note that during training, all these operations will simply be marked as Errors.

Figure 2a contains an illustration of two snippets of music: one original and one with OMR errors. Figure 2b shows the result of their alignment with the NW algorithm. Our training data will mark each note as an error if it has an operation assigned to it; for example, the eighth rest at the end of the OMR snippet in Figure 2a will be marked as an error in training, since it must be removed to correct the score. Under this scheme we have no way of flagging a note as an error if it is not present in the OMR output. As a workaround, we flag a symbol as erroneous in training if, to correct the input, it would be necessary to insert one or more items *after* that symbol in the sequence. For example, we would mark the very first barline of the OMR output in Figure 2b as an erroneous symbol, since there is a missing `8th.pos2.start` immediately succeeding it.

3.3 Dataset and Data Augmentation

Ideally, our dataset would comprise a large, well-curated corpus of symbolic music in a single genre, each with accompanying score files in some image format, and another dataset containing versions of those score images passed through an OMR process. We could match up each symbolic music file with its OMR'ed counterpart and perform an NW alignment between them. Then, during training, the OMR'ed symbolic music would be the input to our model, and the resulting alignment would be our target.

To our knowledge, there are no large, publicly available sources of data that match these requirements. Datasets intended for training OMR systems do not include erroneous OMR outputs, as their purpose is to aid in developing the OMR process itself. Given a set of scores in symbolic and image formats, we could automate a OMR application to process a new dataset in this way, but the commercial OMR software available to us does not have the requisite batch processing functionality that would make this feasible. Instead, we take a small dataset of symbolic music files paired with their OMR'ed counterparts, and supplement it with a large dataset containing music of the same genre, but without accompanying OMR'ed files. We add synthetic errors to this larger dataset to mimic the natural errors in the small dataset.

Our main dataset, from which we derive natural OMR errors, is a set of Mendelssohn's string quartets comprising 24 movements [22], containing a total of 175 thousand tokens when encoded agnostically. This dataset contains OMR'ed versions of each movement generated using the PhotoScore software, partially-corrected versions of the OMR'ed files for each movement, and fully-corrected versions of each movement. The OMR'ed files contain a high proportion of errors; the character error rate between the fully-corrected and uncorrected files is 0.29, and the same statistic for the partially-corrected files is 0.11 [2] . We supplement this with a corpus of the string quartets of Beethoven, Haydn, Mozart, and Schubert, taken from KernScores [3] and the Annotated Beethoven Corpus [23], comprising a total of 361 movements and a total of 9.25 million agnostic tokens. It would be ideal to focus specifically on a single era of music, but we posit that for the sake of ease of training, consistency in instrumentation (restricting ourselves to only string quartets) is a more important consideration.

To augment the dataset, we add errors to the agnostic representation of our dataset in a way that mimics how natural OMR errors are distributed in the smaller dataset. To this end, we set aside a small portion of the Mendelssohn string quartet dataset and run a NW alignment between these snippets and their manually-corrected versions. From these alignments we can obtain statistics

[2] This statistic may sound high compared to the number of errors shown in Figure 1. This is explained by the verbosity of agnostic encodings; something that *looks* intuitively like a single error on the page may actually constitute several.

[3] `http://kern.ccarh.org/`

	Precision on raw OMR Output				Precision on partially corrected OMR			
	R = 0.80	R = 0.90	R = 0.95	R = 0.99	R = 0.80	R = 0.90	R = 0.95	R = 0.99
Baseline LSTUT	0.49	0.47	0.47	0.46	0.33	0.32	0.31	0.31
Architectures								
LSTM	0.49	0.47	0.47	0.46	0.11	0.11	0.10	0.10
Transformer	0.51	0.50	0.50	0.49	**0.14**	**0.12**	**0.12**	0.10
Synthetic Errors								
Random	0.47	0.46	0.46	0.45	0.10	0.10	0.10	0.10
50% fewer errors	0.53	**0.51**	0.50	0.46	0.10	0.10	0.10	0.09
Sequence Length								
64	0.47	0.47	0.46	0.45	0.10	0.10	0.10	0.09
128	0.49	0.47	0.47	0.46	0.10	0.10	0.10	0.09
512	0.50	0.49	0.49	0.48	0.11	0.10	0.10	0.10
1024	**0.56**	**0.51**	**0.51**	**0.50**	0.11	0.11	0.11	**0.11**

Table 1: A table comparing precision scores for given recall (R) scores on our error detection task. The "Baseline" model uses an LSTUT, synthetic errors made with our data augmentation method described in Section 3.3, and a sequence length of 256. Each section of the table changes a single variable from the baseline.

on how each type of symbol tends to be affected by the OMR process. For example, in the OMR process, the symbol `rest.quarter` remains the same with probability 0.527, is deleted with probability 0.142, is replaced with an eighth rest with probability 0.013, and so on. We use these statistics as a probability distribution, sampling from it for every single symbol in an agnostically-encoded piece, and then apply the resulting operation to each symbol.

This is a highly simplified model of how OMR introduces errors. It does not take into account the effects of adjacent symbols on one another, or how errors tend to group together on parts of a page that are badly scanned or damaged. However, we have a small amount of real OMR data to work from, which we plan to use for validation and testing, so we cannot spare much for the purpose of training the error generator. A more sophisticated error-generation model would likely overfit on such a small fraction of this dataset.

We train with the large dataset augmented with synthetic errors, reserving our dataset of Mendelssohn string quartets containing natural OMR errors for validation and testing. Each batch of training data in agnostic format is tokenized and is augmented with errors. Then, each training example in the batch is NW-aligned with its correct version (Section 3.2), and the alignment results are used to create a sequence of binary flags (error or no error) the same length as the training example. We train the model using the synthetic training data as input and these binary flags as the target. Validation and testing follow the same process but use existing OMR'ed data instead of adding synthetic errors.

4. EXPERIMENTS

As a baseline model, we employ the Long Short-Term Universal Transformer (LSTUT) [24] as an encoder-only model. This architectre consists of a 4-layer Universal Transformer [25] (a Transformer network where all the layers share the same weights) with 5 self-attention heads, bookended by two bidirectional Long Short-Term Memory (LSTM) layers in lieu of the positional encoding scheme used by vanilla Transformers. This architecture was designed specifically to learn from short- and long-term repetitive musical structure, and outperformed other variants tested on our tasks. We use the linear self-attention mechanism by Vyas et al. [26], implemented as part of their `fast-transformers` package for use with the Pytorch deep learning framework; this is a modification to the vanilla Transformer network with memory usage that scales linearly with input size instead of quadratically. The source code for our full workflow is available on GitHub [4] .

We test on two alternate architectures: a bidirectional LSTM and a vanilla Transformer. In the baseline LSTUT, the LSTM layers and attention heads all have an internal hidden dimension of 128. The LSTM network uses four layers, each with a hidden dimension of 512, and the vanilla Transformer network has 6 layers each containing 4 self-attention heads, all with a hidden dimension of 128. All models end in a fully-connected layer that reduces each sequence element down to a single dimension. We designed these architectures such that all would have a similar number of trainable parameters (around 8 million).

We also test on different input sequence lengths, and the type of synthetic errors used for data augmentation; the "Random" run applies uniformly random deletions, insertions, and replacements in the training data, and the "50% fewer" run uses our statistically-informed data augmentation scheme but only adds half as many errors, which we use to see if it improves performance on partially-corrected OMR. All models were trained with early stopping based on validation loss, using four NVIDIA P100-PCIE GPUs with a combined 48 GB of memory. When testing dif-

[4] `github.com/timothydereuse/transformer-omr-spellchecker`

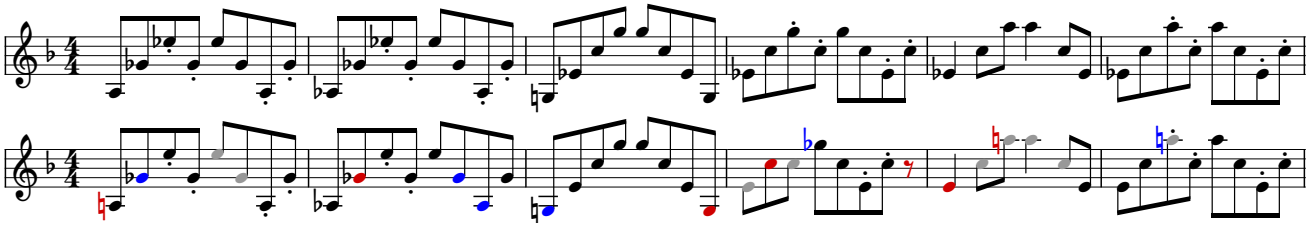


Figure 3: An excerpt from Mendelssohn's String Quartet in E♭ Major, Op. 12, Mvt 1, mm. 212-217 (1st violin part). The upper staff contains the original excerpt, and the bottom one contains OMR errors, along with the predictions of our model. Notes marked in red are errors that our model correctly identified, those marked in blue are errors that our model missed (false negatives), and those marked in grey are correct notes that our model marked as erroneous (false positives). Keep in mind that a correct note may be marked as erroneous if, to correct the score, one needs to insert a symbol *after* it.

ferent sequence lengths, we alter the batch size so as to always use the entirety of the available GPU memory. Lastly, we test on two different sets of data: one is the OMR'ed Mendelssohn quartets, and one is the same quartets after a single pass of human review. This second test set allows us to see if the model can detect the kinds of errors that humans miss when reviewing a score.

In testing, we must apply a threshold to the raw output of the network to turn it into a binary prediction. A common way to do this is to choose a threshold that maximises F1 score on the validation set and then apply that threshold to the test set. However, since we seek to maximise precision *given* a recall score near 1.0, we instead fix several scores for recall, and report the corresponding precision scores on the test set for those thresholds. We anticipate that in a real error-correction scenario, a user would be able to set a threshold themselves based on their tolerance for error; such forms of user interaction are common in high-recall information retrieval applications [27].

4.1 Results and Discussion

Table 1 shows a summary of our results. The Baseline model uses an LSTUT, synthetic errors generated with our data augmentation method, and an input sequence length of 256. We show the precision scores associated with different recall scores to represent different tolerances for missed errors. For example, our baseline model has a precision of 0.47 when the recall is 0.9, meaning that if a user wishes to be sure that 90% percent of the errors are caught, then 53% of the model's detections will be false positives.

The LSTUT model performs slightly better than the LSTM and the vanilla Transformer, though this performance difference is most notable at lower recall scores. Our statistically-informed error generation method outperforms the addition of random errors. At the highest recall score (R=0.99), our best precision score is 0.50, which we obtain at a sequence length of 1024. Precision drops only slightly as our target recall score increases, indicating that a significant fraction of each string quartet is deemed to be correct, and is ignored by the model with high confidence. Our models to perform well at identifying single errors surrounded by correct musical content, while long runs of errors, common in OMR'ed passages with tightly-spaced notes, are sometimes ignored. This may be a consequence of how our data augmentation method does not produce dense clusters of errors.

All models performed poorly on detecting errors in the partially-corrected files, though the vanilla Transformer network did marginally better. Examining our results, we note that our model struggles to notice the *absence* of elements like articulations and accidentals; these kinds of omissions comprise a majority of the errors in the partially-correct OMR files.

Figure 3 shows a violin passage from a Mendelssohn quartet and its OMR'ed version, indicating where our baseline model (at 0.90 recall) predicts errors. A common tendency of our method is illustrated here: when the model cannot pin an error to one single symbol definitively, it will mark a region around the error as erroneous (e.g., the many false positives in measures 4–5 of the excerpt). False negatives do happen (e.g., the missing staccato markings in measure 2), but when notes are missing or inserted, the model tends to make a detection that is at least in the vicinity of the true error.

5. CONCLUSION

For a machine-learning method to be useful in low-resource situations, such as those faced by many archivists and musicologists trying to make long-neglected music more accessible, it must perform a task consistently, given a small amount of data, and be easily generalizable. For this reason we have set ourselves a simple task, in hopes that we can perform it reliably. Even a low precision score may be helpful to a corrector if errors are rare; if our method flags 50% of the score as potentially erroneous, that halves the amount of material the human must check.

While our method achieves a reasonable amount of precision on Mendelssohn's string quartets, we have not proven that it can actually reduce OMR correction time. It would be useful to find ways of evaluating this type of system that speak more directly to its utility for human correctors; for example, by using the results of experiments by Pugin et al. [28] that test how long it takes to correct different types of errors. We plan to integrate our method into an existing notation application or correction interface, to see how it can best aid the symbolic encoding process.

6. REFERENCES

[1] D. Rizo, J. Calvo-Zaragoza, and J. M. Iñesta, "MuRET: A Music Recognition, Encoding, and Transcription Tool," in *Proc. of the 5th Int. Conf. on Digital Libraries for Musicology*. New York, NY: ACM, 2018, pp. 52–56.

[2] M. Alfaro-Contreras, D. Rizo, J. M. Iñesta, and J. Calvo-Zaragoza, "Omr-Assisted Transcription: A Case Study with Early Prints," in *Proc. of the 22nd Int. Society for Music Information Retrieval Conf.*, Online, 2021.

[3] A. Daigle, "Evaluation of Optical Music Recognition Software," Master's thesis, McGill University, 2020.

[4] J. Hajic Jr, J. Novotný, P. Pecina, and J. Pokorný, "Further Steps Towards a Standard Testbed for Optical Music Recognition." in *Proc. of the 17th Int. Society for Music Information Retrieval Conf.*, New York, NY, 2016.

[5] D. Byrd and J. Simonsen, "Towards a Standard Testbed for Optical Music Recognition: Definitions, Metrics, and Page Images," *Journal of New Music Research*, vol. 44, Jul. 2015.

[6] F. Rossant and I. Bloch, "Robust and Adaptive OMR System Including Fuzzy Modeling, Fusion of Musical Rules, and Possible Error Detection," *EURASIP Journal on Advances in Signal Processing*, vol. 2007, no. 1, 2006.

[7] A. McLeod, J. Owers, and K. Yoshii, "The MIDI Degradation Toolkit: Symbolic Music Augmentation and Correction," *arXiv:2010.00059 [cs, eess]*, 2020.

[8] A. Ycart, D. Stoller, and E. Benetos, "A Comparative Study of Neural Models for Polyphonic Music Sequence Transduction," in *Proc. of the 19th Int. Society for Music Information Retrieval Conf.*, Delft, Netherlands, 2019.

[9] K. Sidorov, A. Jones, and D. Marshall, "Music Analysis as a Smallest Grammar Problem," in *Proc. of the 15th Int. Society for Music Information Retrieval Conf.*, Taipei, Taiwan, 2014.

[10] Y. Wang, Y. Wang, J. Liu, and Z. Liu, "A Comprehensive Survey of Grammar Error Correction," *arXiv:2005.06600 [cs.CL]*, p. 35, 2020.

[11] N. Madi and H. S. Al-Khalifa, "Grammatical Error Checking Systems: A Review of Approaches and Emerging Directions," in *Proc. of the Thirteenth Int. Conf. on Digital Information Management*. Berlin, Germany: IEEE, 2018, pp. 142–147.

[12] D. Naber, "A Rule-Based Style and Grammar Checker," Doctoral Thesis, Universität Bielefeld, 2003.

[13] T. Ge, F. Wei, and M. Zhou, "Reaching Human-level Performance in Automatic Grammatical Error Correction: An Empirical Study," *arXiv:1807.01270 [cs]*, 2018.

[14] C. Brockett, W. B. Dolan, and M. Gamon, "Correcting ESL Errors Using Phrasal SMT Techniques," in *Proc. of the 21st Int. Conf. on Computational Linguistics and the 44th annual meeting of the ACL*, Sydney, Australia, 2006, pp. 249–256.

[15] A. Rozovskaya and D. Roth, "Training Paradigms for Correcting Errors in Grammar and Usage," in *Human Language Technologies: The 2010 Annual Conf. of the North American Chapter of the Association for Computational Linguistics*, Los Angeles, CA, 2010.

[16] P. M. Htut and J. Tetreault, "The Unbearable Weight of Generating Artificial Errors for Grammatical Error Correction," *arXiv:1907.08889 [cs]*, 2019.

[17] F. Foscarin, F. Jacquemard, and R. Fournier-S'niehotta, "A Diff Procedure for Music Score Files," in *Proc. of the 6th Int. Conf. on Digital Libraries for Musicology*. The Hague, Netherlands: ACM, 2019, pp. 58–64.

[18] I. Knopke and D. Byrd, "Towards MusicDiff: A Foundation for Improved Optical Music Recognition Using Multiple Recognizers," in *Proc. of the 8th. Int. Society for Music Information Retrieval Conf.*, Vienna, Austria, 2007.

[19] C. Hawthorne, A. Huang, D. Ippolito, and D. Eck, "Transformer-NADE for Piano Performances," in *Proc. of the 32nd Conf. on Neural Information Processing Systems*, Montreal, Canada, 2018.

[20] D. Rizo, J. Calvo-Zaragoza, J. Iñesta, and I. Fujinaga, "About Agnostic Representation of Musical Documents for Optical Music Recognition," in *Proc. of the Music Encoding Conf.*, Tours, France, 2017.

[21] S. B. Needleman and C. D. Wunsch, "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.

[22] J. Degroot-Maggetti, T. de Reuse, L. Feisthauer, S. Howes, Y. Ju, S. Kokobu, S. Margot, N. Nápoles López, and F. Upham, "Data Quality Matters: Iterative Corrections on a Corpus of Mendelssohn String Quartets and Implications for MIR Analysis," in *Proc. of the 21st Int. Society for Music Information Retrieval Conf.*, Montreal, Canada, 2020.

[23] M. Neuwirth, D. Harasim, F. C. Moss, and M. Rohrmeier, "The Annotated Beethoven Corpus (ABC): A Dataset of Harmonic Analyses of All Beethoven String Quartets," *Frontiers in Digital Humanities*, vol. 5, p. 16, 2018.

[24] J. de Berardinis, S. Barrett, A. Cangelosi, and E. Coutinho, "Modelling Long- and Short-Term Structure in Symbolic Music with Attention and Recurrence," in *Proc. of the 1st Joint Conf. on AI Music Creativity*, Stockholm, Sweden, 2020.

[25] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and Ł. Kaiser, "Universal Transformers," *arXiv:1807.03819 [cs, stat]*, 2019.

[26] A. Vyas, A. Katharopoulos, and F. Fleuret, "Fast Transformers with Clustered Attention," *arXiv:2007.04825 [cs, stat]*, 2020.

[27] H. Zhang, M. Abualsaud, N. Ghelani, M. D. Smucker, G. V. Cormack, and M. R. Grossman, "Effective User Interaction for High-Recall Retrieval: Less is More," in *Proc. of the 27th ACM Int. Conf. on Information and Knowledge Management*, Torino, Italy, 2018, pp. 187–196.

[28] L. Pugin, J. A. Burgoyne, and I. Fujinaga, "Reducing Costs for Digitising Early Music with Dynamic Adaptation," in *Research and Advanced Technology for Digital Libraries*, L. Kovács, N. Fuhr, and C. Meghini, Eds. Berlin: Springer Berlin Heidelberg, 2007, vol. 4675, pp. 471–474.