AIMC 2024 (09/09 - 11/09)

Strategies for building AI-enhanced audio software with impact

Matthew Yee-King Mark d'Inverno

Published on: Aug 29, 2024

URL: https://aimc2024.pubpub.org/pub/g4bsnsx6

Strategies for building AI-enhanced audio software with impact

In this paper, we explore the challenge of increasing the uptake of AI-music technology research within academia and across the music industry. We consider two key audiences for AI-music technology: other researchers (an effect known in the UK as ‘academic significance’) and amateur and professional music practitioners (known as ‘impact’). We review previous work investigating the interactions between code repository design, repository usage statistics and citation count. We look at previous work exploring reasons for the low uptake of AI-based and other music technology research, such as poor interoperability and lack of alignment with the needs of creative practitioners. We then present a preliminary analysis of 93 AI-music-related GitHub repositories wherein we examine the interaction between repository features and uptake metrics such as citations and downloads. We find that AI-music research code repositories providing downloadable releases of plugins, which inter-operate with existing music technology, can achieve high download rates. We also verify the previous finding for non-music AI repositories that the number of forks positively correlates with the number of citations for associated research papers, noting that the number of forks is related to the design of the repository. We end the paper by describing how we are currently developing our AI-music software using a combination of C++, JUCE, PyTorch, RTNeural and plugin technology, a setup we have chosen with an aim to increase significance and impact. We connect our development setup to the findings from the review of existing work and our GitHub analysis.

Author Keywords

AI-music, JUCE, plugins, music technology

1. Introduction

Our vibrant community of AI-music researchers produces an exciting and diverse range of outputs. These include applications that provide complete user experiences, software libraries for developers to build upon, plugins that can be used in existing software such as digital audio workstations (DAWs), data sets, foundational and pre-trained neural networks, and other models.

In this paper, we refer to the software ‘output’ created by the community as ‘AI-music technology’; this incorporates music technology and involves using artificial intelligence and/or machine learning techniques to provide novel features that would not otherwise be easy to implement. Whilst there has been an “endless stream” of AI-music research since the 1960s [1] we expect that many researchers will be familiar with the problem of low uptake of their creations. Low uptake means that the underlying research will have limited traction with. or influence on, other researchers and - just as crucially - the music technology industry. By the music technology industry, we include all potential technology users across the amateur/professional spectrum. Academic and industrial uptake of this sort is essential, in the UK at least, because these are the metrics used to evaluate our work in order to assign funding.

In this paper, we analyse some of the issues that AI-music researchers face when looking to increase the uptake of their work. Then, partly based on this research but also our own experiences, we outline methodological steps both in the engineering of software and the way it is made available to the public that we believe could significantly increase the uptake for such - both in academia (for further research development) and in the industry (for building products).

To further motivate our work, we as researchers should consider the broader ethical context of AI deployment and the current climate around AI-music technology specifically, which is exciting but also problematic. Music professionals and the general public have seen a series of very impressive but, in some cases, ethically dubious systems such as deepfake singing voice synthesizers[1] and end-to-end text-to-music systems[2]. Whilst these systems can be headline-grabbing, we do not think they represent the rich diversity of work happening in our community or, indeed, many of the values of the community and the creative practitioners on whom this work can potentially have a profound impact.

The climate of excitement - daresay hype - around AI-music technology at least presents AI-music researchers with an opportunity to present their work to a larger audience. And in many cases, that work is much more aligned with the needs of creative practitioners than those other systems. But the problem we want to crack is what is the best way to present and engineer that work so that it can be easily used by this audience as well as increasing the academic significance of the work along the way. To address this problem, we consider the following broad research questions and make some initial progress in responding to them here in this paper:

1. RQ1 What are the uptake levels for AI-music research in academia and industry, and what factors might be associated with high and low uptake?
2. RQ2 What do researchers (designing) and musicians (using) of AI-music research have to say about uptake and why or why not it might be used?
3. RQ3 What is considered best practice when sharing research software more generally, and is this compatible with current practices in AI-music research?
4. RQ4 What kind of design methodological changes, focusing on technology stacks, will enable greater uptake of AI-music research across academia and the music industry?

2. Previous work investigating AI-music technology, significance and impact

2.1 The interaction between repository features and academic significance

It has become common practice for researchers who build AI systems across various application domains (such as large language models, speech recognition, and reinforcement learning) to release open-source code repositories alongside the associated research papers. The same is increasingly becoming the case for AI-music researchers. The question arises as to whether there is any consensus about best practices for sharing AI

research software with academic communities to maximise academic significance and if this can be applied to AI-music software.

Considering mainstream AI research, a recent paper from Bhattarai et al. investigated the relationship between interaction statistics on GitHub and academic significance in terms of citations[3]. The interaction statistics were the number of stars (equivalent to storing a favourite in a web browser), forks (holding a copy of a repository in one's personal account), watchers (signing up for notifications about updates to a repository) and issues (posting a question or bug report to a repository). The researchers found these metrics were positively correlated with higher citations. They do not attempt to show causation, so it is not clear if citations cause repository activity, vice-versa or both.

In their article 'What makes a popular academic AI repository', Fan et al. analysed engineering practice in popular and unpopular GitHub repositories, where popularity is the number of stars[4]. Firstly, they note that there appears to be a positive correlation between stars and citations, as did Bhattarai et al.. Then, they identify features that differentiate popular from unpopular repositories from a set of 21 numerical features relating to code, documentation, and reproducibility (after [5]). The researchers present the features as a checklist for research repositories to align their design with those of successful repositories. The checklist includes technology stack suggestions (e.g. use PyTorch, not tensorflow), provision of good documentation, providing pre-trained models and having well-engineered code. Many of these features seem like standard good engineering, but the difference here is their empirical verification.

Moving to computer music research software, McFee et al. provide detailed advice on repository design, dataset provision and engineering practice aimed at Music Information Retrieval researchers who wish to improve the reproducibility and utility of their work[6]. Their advice is strongly aligned with the empirical observations from Fan et al. noted above concerning dataset provision, good engineering practice, documentation, and so forth.

2.2 Issues affecting non-academic impact

The three papers discussed above provide excellent advice concerning the design of public-facing source code repositories that other academics can use easily and which might, therefore, lead to greater academic significance. But how should researchers go about increasing non-academic impact? To approach this, we can first review some surveys that examine uptake and opinion around AI-music software amongst non-academic groups.

In a 2021 survey of 118 people, 41% of whom were professional musicians, Frid and Ilsar investigated several themes, including desirable roles for AI within existing creative processes[7]. They "observed generally positive attitudes to ML/AI as tools in composition processes" but noted negative themes, such as not wanting ML/AI to influence certain aspects of the artistic process.

Knotts and Collins surveyed 116 music software users in 2020, examining uptake and attitudes towards AI-music technology[8]. Whilst only around 7% of the respondents reported music as their primary source of income, similar themes emerged around desirable roles for the technology in the creative process: “Few respondents were seeking their own replacements: Musicians usually craved collaborators and assistants who would help with creative workflow”. So it would seem from this analysis that creatives are interested in working with this technology, but the nature of the role it takes in the creative process is crucial to them.

Switching from AI-music technology to the design of novel Digital Music Instruments (DMIs), several researchers have analysed factors relating to impact, such as longevity and uptake. For example, in a 2018 paper, Borbon and Avila identified the reasons for the lack of longevity for DMIs as a focus on novelty over sustaining practice, lack of pedagogy and community engagement, and evaluation limited to the artefacts rather than complete ecosystems [9]. In another paper aiming to explain the limited impact of DMIs, Goudard noted the following factors: the ephemeral nature of sound, constantly evolving technology, and a focus on unique performance contexts (e.g. demonstrations at conferences) [10]. Calegario et al. discuss reproducibility in DMIs based on information in published papers and case studies; they even go as far as attempting to re-create DMIs using the published information [11]. They discuss the challenges of obsolete software and suggest using more standard/widespread programming languages and technology stacks. Digging deeper into choosing the appropriate technologies for implementing DMIs, Sullivan and Wanderley reviewed 40 years of NIMEs. They concluded that “a DMI may not be viable unless basic stability, reliability and compatibility standards have been met in the design process [12].

To summarise, there are some clear themes in the literature concerning academic significance and non-academic impact. Designing your source code repository according to best-practice guidelines is measurably associated with increased citations and can enhance the reproducibility of your work. Recent work identifies quite specific repository features, such as technology stack choice, that are associated with popularity, which is then associated with academic significance. For non-academic impact, creative practitioners are interested in using AI technology but are very concerned with the role AI technology plays in their process - essentially, what they are happy to give over to AI and what they are not. The community around the technology is also important, and other factors that researchers identify that should support increased impact include technology stack choices, stability, and reliability.

2.3 Technology stacks used by music professionals

The research presented in the papers we have cited above, which investigate factors affecting DMI uptake and longevity, clearly identifies a relationship between the technology choices researchers make and the potential impact of their work outside academia in the music industry. But which technologies, or which kinds of technologies, do music producers - both amateur and professional - tend to use? While there is some (though limited) academic work analysing the practice and range of tools used in modern-day music production, we found it hard to find a recent analysis or survey of how professional musicians make music in the 2020s.

However, it is generally accepted that the following list covers most of the available categories of technologies: traditional instruments, digital audio workstations (DAWs), plugins, synthesizers, sequencers, samplers, mixing tools, recording hardware and (MIDI) controllers[13][14].

The items in that list are not mutually exclusive or fully descriptive; several items can be hardware or software, plugins can provide the functionality of several of the other items (e.g., synthesizers and mixing tools), and DAWs package several of the items into a single application and allow for functionality extension via plugins. However, this list can be regarded as the contemporary technology ecosystem for music production.

This leads to the question of the underlying technology used to create the items in this list. Ignoring the electrical engineering aspects of the hardware devices, most, if not all, of the software in the list - DAWs, plugins, digital sound synthesis algorithms, etc.- is written in C or C++ and provided as natively compiled binaries. Music producers expect to be able to download installer programs and install and execute the software straightforwardly, often in an integrated fashion.

2.4 Technology stacks used by music and AI-music researchers

How do the technology stacks used by AI-music researchers compare to those identified above for commercial music software? In their survey, Knotts and Collins found that common languages for self-built AI-music systems were SuperCollider, Max/MSP, Python, PureData and Javascript[8]. They list other, less common languages, such as ChucK and Csound, but interestingly, given our discussion above, they make no mention of C++. Yet it is C++, as noted above, which is the chosen language used to develop commercial music software.

Tatar and Pasquier's 2019 typology of musical agents, which analysed a diverse set of recent and historical publications in AI-music, confirms this list: Max, Max for Live, PureData, Processing, SuperCollider, Python, and Java[15]. They also list several essential areas needing development to further the AI-music field, and this list is strongly aligned with our work's objectives in investigating ideal technology stacks: deployment and accessibility, standardisation and interoperability, and real-world applications.

One area of rapid growth in computer music technology stacks, and therefore worthy of mention here, is livecoding[16]. Livecoding languages form a growing list of 65 languages from Alda to xi with multiple development environments and frameworks[2]. Livecoding has a different perspective on the use of technology in that it puts the writing of code in real-time upfront in its artistic practice, which immediately sets it apart from the practice of professional musicians. Indeed, some live coders are determinedly motivated by a desire to move away from the standard technology ecosystem used by other music practitioners: 'I was also tired of making music with a DAW because it was tedious' - Mike Hodnick in [17].

Limited research and development connects livecoding with AI and machine learning, and the technology stacks tend to be more obscure and even further from those familiar to music producers. For example, Bernado

and Kiefer's web-based SEMA system allows for the creation of new languages that can interoperate between machine learning and audio components [18].

Returning to 'mainstream' AI-music research, a limitation of commonly used computer music languages such as SuperCollider, PureData, and Max/MSP is the lack of machine learning capabilities. Whilst data mining and neural network libraries for these languages do exist[19][20], they do not provide the kind of feature set required for contemporary deep-learning work found in TensorFlow or PyTorch. Another limitation is music producers' lack of familiarity with the tools and their lack of interoperability with mainstream music technology.

Another common approach, one that allows for the deployment of neural networks with contemporary features and which enables researchers to share their work with others easily, is the use of web technology. For example, the MIMIC platform provides a browser-based programming environment with integrated machine-learning capabilities[19], and the Magenta project provides a range of musical neural network models that work in the browser[20].

The browser is a powerful platform for local inference (where a pre-trained neural network executes on the local machine). Still, it is not generally appropriate for training, except for interactive training with small datasets, as seen in Learner.js[21]. For more extensive training tasks, we believe using a machine learning framework such as Tensorflow or PyTorch and appropriate hardware is necessary. This view is supported by the technology currently used, at least in the training part of many 'deep-music' systems.

2.5 Research technology aimed at creative practitioners

Aside from the web-based systems mentioned above, there are other examples of AI-music research systems designed to be used by creative practitioners. We end our review of previous work by highlighting some of these systems as their approach relates to ours. Fiebrink's Wekinator allows non-machine learning specialists to use an interactive machine learning workflow to specify and train a neural network 'meta-instrument' [22]. Wekinator has undergone various re-implementations, from its original Java-based system, through a C++ library called RAPIDMIX [23] and a Javascript library called learner.js [19]. Roberts et al.'s Magenta Studio is a set of Max for Live (Ableton Live plugins) which wrap pre-trained models running in tensorflow.js [24]. Faust can export computer music systems to many formats, including VST [25]. Other examples of interoperable software are the PureData VST hosting external [26] and the embeddable libPD [27].

Closest to the current technology approach we are employing, which we describe later in this paper, is the small but growing number of researchers who make their AI-music systems available as native VST plugins typically written in C++. Examples are the DDSP timbre transfer plugin[3], IRCAM's RAVE VST[4], Steinmetz's various neural effects plugins[5], and Atkins' Neural Amp Modeller6. Neutone is another neural network plugin, but in this case, it is designed to allow people to deploy their models easily without needing to develop their own plugin[6].

3. Analysis of AI-music software on GitHub

In this section, we will describe an analysis we have carried out of AI-music and related GitHub source code repositories. We use this study to shed some light on the levels of uptake for these kinds of repositories and how uptake interacts with the kinds of technology the researchers use.

We acknowledge that GitHub might not be the first place that music producers will visit to access music software, but it does have a significant number of AI-music repositories with downloadable binaries that can be found via a Google search for this kind of software and music technology websites do link out to GitHub, e.g. [7].

3.1 GitHub analysis method

The `data goal' of this analysis is to have a list of GitHub repositories, with associated research papers where possible, and a range of statistics and labels describing each repository. The high-level steps we took were as follows:

1. Gather a large set of recent AI-music research papers
2. Extract GitHub repositories from the research papers
3. Gather GitHub repository statistics via the GitHub API
4. Clone repositories and undertake further analysis to identify main languages and assign 'type' labels.

To gather the initial set of research papers, we identified four recent review papers with extensive reference lists covering AI-music topics [1][15][28][29]. We then extracted the reference lists from the text in the PDFs and converted this list to BibTeX format giving us a list of 660 papers. Next, we needed to gather the PDF files for the 660 papers so we could extract GitHub repository links.

Accessing the PDFs for the papers involved searching Google Scholar for the paper titles, extracting the links to potential PDFs, and then importing the links back into the BibTeX file. We ran a Javascript program in a web browser to politely 'scrape' Google Scholar - it was necessary to pause between requests and to complete captchas to avoid being blocked occasionally. Eventually, we had a folder containing 660 HTML files representing the search results for each of the papers. Having the complete search return files would allow us to extract more data later without needing to hit Google Scholar again. We extracted the top links from the HTML files and put these into the BibTeX entries for each paper. To download the PDFs, we imported the BibTeX to Zotero's desktop app and used its 'Find available PDFs' feature to download the PDFs. We found this achieved a better hit rate than directly downloading from the links on the HTML files for the papers with a script. We exported the papers as PDFs from Zotero and parsed all PDFs, looking for GitHub links. Some papers had more than one GitHub URL, so we selected the first one. Eventually, we had a list of around 140 GitHub repositories.

At this stage, we added some other repositories of interest: wekinator, Supercollider, puredata, essentia and several neural network-based plugins that had not come up in our paper search, including the top 3 plugins from the 2023 Audio Programmer Neural Audio Plugin Competition[8].

The next step was to use the GitHub API to gather statistics about the repositories in our list. We gathered the following statistics: stars, forks, downloads, watchers, open issues, network count (no. forks and forks of forks), commit count, age (days), languages used, total amount of code and percentage of code per language. We were able to gather statistics on 93 repositories as several did not exist or the repository URL extracted from the paper was not valid. 14 of the 93 repositories provided ‘releases’, which are necessary to gather download statistics. Releases generally consist of a packaged version of the software or a platform-specific installer. Some repositories do not provide releases but do provide links to downloadable installers. We did not process these as we would not be able to count the number of downloads for links to external sites.

In the final stage of data gathering, we cloned the 93 repositories and did some more analysis. We fed the README files from each repository to ChatGPT4 and used the following prompt: “read the attached readmes into your LLM and tell me what they are about. Do not use Python code to analyse the readmes - do it with your large language model capabilities. Try to come up with a tag for each one, namely application, plugin, tool, library, data set or model. Print out a table with the sub-folder name and the label you selected. ”. We had to repeatedly request continuation to get the complete list. We checked around ten outputs and were happy that it looked to be making reasonable labelling decisions.

We appreciate that there is a lot more analysis that is possible than what we have undertaken to date, including in-depth automated code analysis, but for now, we leave that to future work by ourselves and others, and we shall shift to examining the data we did gather. With our resulting list, we feel that we are presenting a window on the current ‘scene’ (in late 2023) of open-source computer music and AI-music projects. We provide data, including the list of papers, the GitHub repositories and associated statistics in a GitHub repository[9].

3.2 Results

We identified 93 repositories through the method described in the previous section. All 93 had three or more stars, and 88 had one or more network links. 14 repositories provided releases, allowing us to gather download statistics.

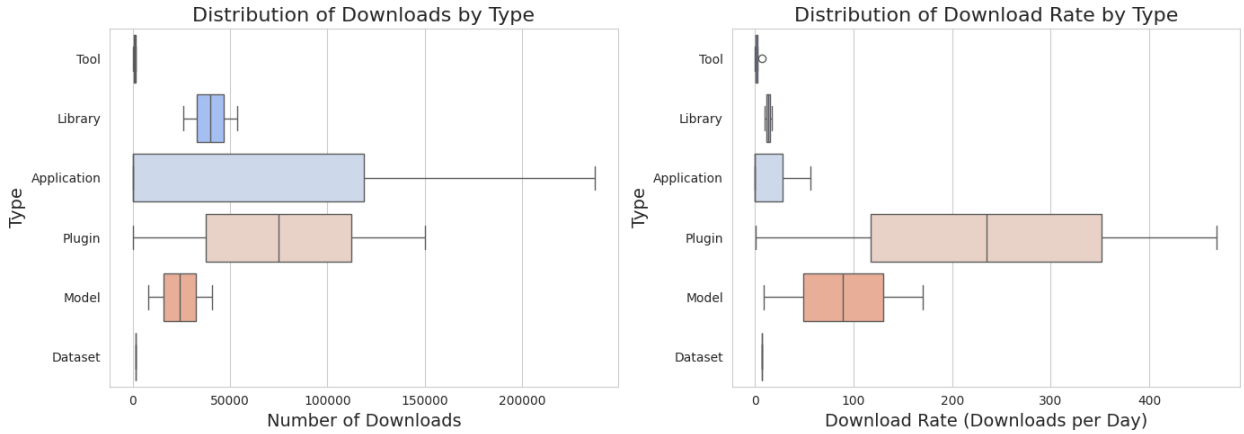

Figure 1: Type of repository against total downloads over all time (top) and download rate (bottom). Plugins and applications achieve high numbers of downloads. Plugins achieve the highest download rates. Based on 14 repositories that provided releases.

Figure 1 shows information about total downloads and download rates for different types of repositories. Although this data is from the small set of 14 repositories that provided releases and, therefore, download statistics, it clearly shows that several types, and especially plugins, can achieve high numbers of downloads and a high download rate. We carried out a non-parametric Mann-Whitney U test to evaluate the significance of the difference between these distributions. This would allow us to state if the range of download counts and rates is significantly different between the repository categories. The test indicated that the distributions are not significantly different (no p-values less than 0.05), but we believe the small sample size is a limiting factor.

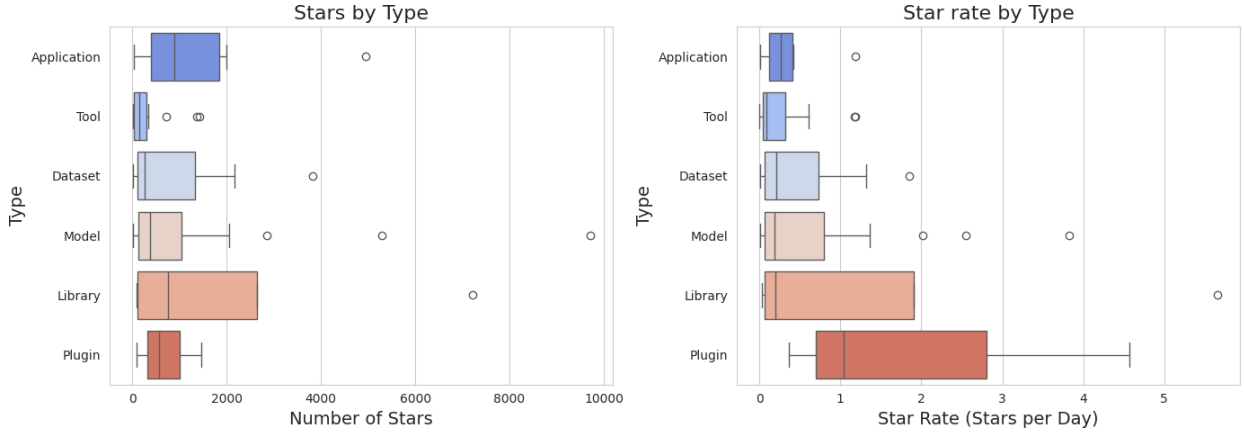

Figure 2: Type of repository against the distribution of total number of stars (top) and stars per day (bottom).

Figure 2 shows the type of repository against the number of stars and stars per day. The sample size here was 91, which is the number of repositories with one or more stars and less than 10,000 stars (we considered the two repositories with more than 10,000 stars as outliers). All repository types appear to be able to gather many stars, with the exception of tools. Plugins and libraries appear to have the highest rate of star gathering. The Mann-Whitney U test was more powerful here with the larger sample size. It found the following p-values

around or below 0.05: application stars vs tool stars (p = 0.047), model stars vs tool stars (p = 0.059), library stars vs tool stars (p = 0.059), plugin star-rate vs. tool star-rate (p = 0.058). So it appears that the distribution of stars for tools is the most different.

	Correlation	**P-value**
Age (days)	0.473	**0.000**
Forks	0.345	**0.005**
Network Count	0.345	**0.005**
Watchers	0.221	0.077
Stars	0.200	0.110
Open Issues	0.164	0.192
Commit Count	0.057	0.650
Downloads	-0.049	0.700
Main language lines	-0.087	0.493
Code total	-0.091	0.473

Table 1: Correlations between various metrics and citation count. The dataset was filtered to stars in the range 1-8,000 and citations 1-2000 to remove outliers. The sample size was 65.

Next, we investigated the interaction between various metrics and citation count. We wanted to see if we could repeat the observation for non-music AI repositories from [3] and [4] that forks and stars correlate positively with citations for AI-music repositories. We present the results in Table 1. Prior to computing the correlations, we filtered the data only to include repositories with stars in the range 1-8,000 and citations 1-2000 to remove outliers. The sample size was 65. Age correlates positively with citation count, somewhat unsurprisingly. Forks and Network size also correlate positively with citations. This confirms one of the observations from the other papers. This observation does not tell us if there is causation or which way the influence, if there is any, flows.

Watchers and stars are close but not significantly correlated with citations. So we could not verify the previously reported correlation between stars and citations. A possible explanation for the stronger relationship between forks and citations than stars and citations is that forks are a stronger action on GitHub. Forking adds a copy of the repository to your account. Starring is a weaker action, more akin to “favouriting” a website.

4. Our approach to AI-music technology

We shall now describe our approach to AI-music technology construction as a case study and connect it to the themes that have emerged in the previous sections of the paper. We do not wish to claim that the workflow is completely novel, but it is something we have developed iteratively through the development of many different systems. The workflow proceeds in several distinct phases:

Phase 1: Data preparation and model development. This stage is concerned with building the AI and machine learning models that power our tools and, where appropriate, sourcing and preparing the data. If the models involve neural networks, we build and test them using Python and PyTorch. Python is a quick and easy language to experiment with, and PyTorch provides deep learning capabilities and allows for easy export of models using its TorchScript system. This stack is extremely standard for AI research and is aligned with Fan et al.'s findings [4]. Torchscript and PyTorch are also quite interoperable, an area identified for improvement in DMI research by several studies we cited earlier.

Phase 2: Model deployment. We currently use C++, libtorch and the JUCE framework to deploy our models either in standalone native apps or as VST3 plugins. libtorch provides an interface between C++ and most of the PyTorch functionality. This includes TorchScript, which can load in a model exported from Python code. TorchScript models run quite fast in libtorch, but we also use the RTNeural library because it allows certain models to run significantly faster than libtorch. RTNeural only supports a small subset of the libtorch functionality and does not support training, only inference. The motivation for this approach to deployment is achieving non-academic significance by inter-operating with existing music technology.

Phase 3: Publishing on GitHub: The next step is sharing our work with others by putting it on GitHub. We aim to follow the best practices for making our GitHub repositories helpful for both researchers and everyday users, even though figuring out the best way to do this is still a work in progress. We described the current best practice for AI source code repository design when aiming for academic significance, and we were able to repeat some of the findings in our GitHub analysis. Things to consider are good documentation, well-organised code and other good engineering practices. Concerning non-academic impact, our preliminary GitHub data analysis showed that plugins can achieve high download rates but also that providing releases on GitHub is necessary to access those statistics.

Phase 4: Workshops and evaluation. We have started to run regular workshops with musicians to show them how to work with a range of AI-music tools, including our own plugins and applications, third-party web-based services and other AI-powered plugins. At this stage, we also gather feedback from users, which we can feed into future iterations of the software. We have found that macOS is quite difficult to work with in workshop scenarios due to security controls on the running of non-notarised software, many different versions of macOS being used by musicians and the mixture of Intel- and ARM-based hardware. Going forward, we would recommend notarised universal binary builds that target macOS 11 and later. Windows tends to be less

problematic, but one has to consider which version of Windows to target at least. We typically provide the rare Linux-using musicians with source code so they can build the software themselves.

5. Conclusions

Here we revisit the four research questions and describe the initial progress we have made against them.

RQ1: What are the uptake levels for AI-music research in academia and industry, and what factors might be associated with high and low uptake?

Our observations show that AI-music code repositories can receive many thousands of stars and downloads and that this can happen both in a short space of time and over a longer period. So putting software on GitHub works and is clearly a pathway to significance and impact. Previous work has found correlations between GitHub interactions such as stars and forks and the number of citations. We found correlations between citations and both repository age and forks but not stars. Furthermore, we did not find any obvious relationship between the chosen main language for the software and these metrics, or the category of repository, aside from tools showing low uptake.

RQ2: What do researchers (designing) and musicians (using) AI-music research have to say on uptake, and why or why not it might be used?

Based on our analysis of previous work, musicians are open to using AI-music tools, but an important factor is the role the system plays in the creative workflow. They do not seek their own replacement - they want to retain authorial autonomy over the creative process. Also, previous analyses of new digital music instruments and their uptake suggest that to see sustained use, the tools need to be interoperable and robust, and researchers should consider using mainstream music technology stacks.

RQ3: What is considered to be best practice when sharing music research software with the research community in general, and is this compatible with practices in AI-music research software?

Our review of previous work suggested general agreement on guidelines pertaining to the design of source code repositories, such as good engineering practice, examples and documentation, and provision of datasets and pre-trained models. This could be applied to the research software repositories we examined, and in some cases, it already is. We also propose that researchers provide downloadable releases of their software, which will allow them to generate DOIs for their repositories and gather download statistics.

RQ4: What kinds of design methodological changes, focusing on technology stacks, will enable greater uptake of AI-music research across academia and the music industry?

We have presented statistics which indicate that releasing AI-music systems as plugins with executable releases can lead to very high download rates. We have described our current development methodology and explained how it is designed to allow for rapid development with an output of a product that creative

practitioners are prepared and able to pick up and use. Drawing together threads developed throughout this paper, we have proposed a coherent methodology for helping with the traction (academic significance and industry impact) for new AI-music systems. We plan to develop this methodology and will report on that project and any lessons learned from doing so.

Ethics statement

The work described in this paper used data that is available via the internet, such as Google Scholar pages and research papers. It also used data accessible via the GitHub API, namely statistics relating to repository activity, forks and so forth. No ethical approval was needed for this research.

Footnotes

1. https://www.bbc.co.uk/news/entertainment-arts-65298834 ↩
2. https://github.com/toplap/awesome-livecoding ↩
3. https://github.com/magenta/ddsp-vst ↩
4. https://github.com/acids-ircam/rave_vst ↩
5. https://github.com/csteinmetz1/neural-2a ↩
6. https://github.com/QosmoInc/neutone_sdk ↩
7. https://sonicstate.com/news/2021/11/30/more-ways-to-remove-musical-elements/ ↩
8. https://www.theaudioprogrammer.com/neural-audio ↩
9. https://GitHub.com/anon ↩

References

1. Ji, S., Luo, J., & Yang, X. (2020). A comprehensive survey on deep music generation: Multi-level representations, algorithms, evaluations, and future directions. *arXiv Preprint arXiv:2011.06801*. ↩
2. Copet, J., Kreuk, F., Gat, I., Remez, T., Kant, D., Synnaeve, G., … Défossez, A. (2023). *Simple and Controllable Music Generation*. ↩
3. Bhattarai, P., Ghassemi, M., & Alhanai, T. (2022). Open-source code repository attributes predict impact of computer science research. *Proceedings of the 22nd ACM/IEEE Joint Conference on Digital Libraries*, 1–7. Cologne Germany: ACM. ↩

4. Fan, Y., Xia, X., Lo, D., Hassan, A. E., & Li, S. (2021). What makes a popular academic AI repository? *Empirical Software Engineering*, *26*, 1–35. Retrieved from https://idp.springer.com/authorize/casa?redirect_uri=https://link.springer.com/article/10.1007/s10664-020-09916-6&casa_token=JHgRoUz3LlYAAAAA:zZv4zx7EnKJZ19xEP0I3GWElo1ki91xbdQUwhZFvIR9yCBO9EE0uRTYk-mNmA1_GOyFr4MvcDe10FYU ↩
5. Borges, H., & Tulio Valente, M. (2018). What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform. *Journal of Systems and Software*, *146*, 112–129. https://doi.org/10.1016/j.jss.2018.09.016 ↩
6. McFee, B., Kim, J. W., Cartwright, M., Salamon, J., Bittner, R. M., & Bello, J. P. (2018). Open-source practices for music signal processing research: Recommendations for transparent, sustainable, and reproducible audio research. *IEEE Signal Processing Magazine*, *36*(1), 128–137. Retrieved from https://ieeexplore.ieee.org/abstract/document/8588406/ ↩
7. Frid, E., & Ilsar, A. (2021). Reimagining (Accessible) digital musical instruments: A survey on electronic music-making tools. *NIME 2021*. PubPub. ↩
8. Knotts, S., & Collins, N. (2020). A survey on the uptake of Music AI Software. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 499–504. https://www.nime.org/proceedings/2020/nime2020_paper95.pdf. Retrieved from https://www.nime.org/proceedings/2020/nime2020_paper95.pdf ↩
9. Marquez-Borbon, A., & Martinez Avila, J. P. (2018). *The problem of DMI adoption and longevity: Envisioning a NIME performance pedagogy*. Retrieved from https://www.researchgate.net/profile/Juan-Martinez-Avila/publication/326152893_The_Problem_of_DMI_Adoption_and_Longevity_Envisioning_a_NIME_Performance_Pedagogy/links/5b3b6986a6fdcc8506eb159c/The-Problem-of-DMI-Adoption-and-Longevity-Envisioning-a-NIME-Performance-Pedagogy.pdf ↩
10. Goudard, V. (2019). *Ephemeral instruments*. arXiv. Retrieved from http://arxiv.org/abs/1909.13775 ↩
11. Calegario, F., Tragtenberg, J., Frisson, C., Meneses, E., Malloch, J., Cusson, V., & Wanderley, M. M. (2021). Documentation and Replicability in the NIME Community. *NIME 2021*. PubPub. ↩
12. Sullivan, J., & Wanderley, M. (2018). *Stability, Reliability, Compatibility: Reviewing 40 Years of NIME Design* [PhD Thesis]. McGill University/Université McGill. ↩
13. Reuter, A. (2022). Who let the DAWs Out? The Digital in a New Generation of the Digital Audio Workstation. *Popular Music and Society*, *45*(2), 113–128. Retrieved from https://www.tandfonline.com/doi/full/10.1080/03007766.2021.1972701 ↩

14. Andrew Maz. (2023). *Music Technology Essentials: A Home Studio Guide*. Focal Press. ↩
15. Tatar, K., & Pasquier, P. (2019). Musical agents: A typology and state of the art towards Musical Metacreation. *Journal of New Music Research*, *48*(1), 56–105. Retrieved from https://doi.org/10.1080/09298215.2018.1511736 ↩
16. Collins, N., McLean, A., Rohrhuber, J., & Ward, A. (2003). Live coding in laptop performance. *Organised Sound*, *8*(3), 321–330. Retrieved from https://www.cambridge.org/core/journals/organised-sound/article/live-coding-in-laptop-performance/08F42B84BBCA427C345030481A3DDA0D ↩
17. Blackwell, A. F., Cocker, E., Cox, G., McLean, A., & Magnusson, T. (2022). *Live coding: A user's manual*. MIT Press. ↩
18. Bernardo, F., Kiefer, C., & Magnusson, T. (2019). An AudioWorklet-based signal engine for a live coding language ecosystem. *Web Audio Conference (WAC 2019)*, 77–82. https://webaudioconf.com/_data/papers/pdf/2019/2019_40.pdf. Retrieved from https://webaudioconf.com/_data/papers/pdf/2019/2019_40.pdf ↩
19. Grierson, M., Yee-King, M., McCallum, L., Kiefer, C., & Zbyszynski, M. (2019). *Contemporary machine learning for audio and music generation on the web: Current challenges and potential solutions*. Retrieved from https://ualresearchonline.arts.ac.uk/id/eprint/15058/1/ICMC2018-MG-MYK-LM-MZ-CK-CAMERA-READY.pdf ↩
20. Roberts, A., Hawthorne, C., & Simon, I. (2018). *Magenta. Js: A JavaScript API for augmenting creativity with deep learning*. Retrieved from https://research.google/pubs/pub47115/ ↩
21. McCallum, L., & Grierson, M. S. (2020). Supporting Interactive Machine Learning Approaches to Building Musical Instruments in the Browser. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 271–272. Birmingham City University Birmingham, UK. ↩
22. Fiebrink, R., & Cook, P. R. (2010). The Wekinator: A system for real-time, interactive machine learning in music. *Proceedings of The Eleventh International Society for Music Information Retrieval Conference (ISMIR 2010)(Utrecht)*, *3*. ↩
23. Bernardo, F., Zbyszyński, M., Grierson, M., & Fiebrink, R. (2020). Designing and Evaluating the Usability of a Machine Learning API for Rapid Prototyping Music Technology. *Frontiers in Artificial Intelligence*, *3*. Retrieved from https://www.frontiersin.org/articles/10.3389/frai.2020.00013 ↩
24. Roberts, A., Engel, J., Mann, Y., Gillick, J., Kayacik, C., Nørly, S., … Eck, D. (2019). *Magenta studio: Augmenting creativity with deep learning in ableton live*. Retrieved from http://research.google/pubs/pub48280/ ↩

25. Weinstein, J. R., & Cook, P. R. (1997). *Faust: A framework for algorithm understanding and sonification testing*. Georgia Institute of Technology. ↩

26. Ressi, C. (2021). [Vstplugin]A Pd external for hosting VST plugins. *Revista Vórtex*, *9*(2), 20–20. Retrieved from https://periodicos.unespar.edu.br/index.php/vortex/article/view/4544 ↩

27. Brinkmann, P., Wilcox, D., Kirshboim, T., Eakin, R., & Alexander, R. (2016). Libpd: Past, present, and future of embedding pure data. *Proceedings of the 5th Pure Data Convention (PdCon), New York City*. http://danomatika.com/publications/libpd_pdcon_16.pdf. Retrieved from http://danomatika.com/publications/libpd_pdcon_16.pdf ↩

28. Lee, S., Choi, H.-S., & Lee, K. (2022). Differentiable artificial reverberation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *30*, 2541–2556. ↩

29. Hayes, B., Shier, J., Fazekas, G., McPherson, A., & Saitis, C. (2023). *A Review of Differentiable Digital Signal Processing for Music & Speech Synthesis*. ↩