Leveraging Android Phones to Democratize Low-level Audio Programming

Carla Sophie Tapparo
Independent Researcher
Buenos Aires, Argentina
sophiecarlatapparo@gmail.com

Brooke Chalmers
Northeastern University
Boston, MA, USA
chalmers.b@northeastern.edu

Victor Zappi
Northeastern University
Boston, MA, USA
v.zappi@northeastern.edu

ABSTRACT

In this work we introduce LDSP, a novel technology capable of turning any Android phone into a high-performance embedded platform for digital musical instrument (DMI) design. Embedded platforms are powerful technologies that changed the way we design and even think of DMIs. Their widespread adoption has popularized low-level audio programming, enabling engineers and artists alike to create highly responsive, self-contained digital musical instruments that have direct access to hardware resources. However, if we shift our focus away from the wealthy countries of the *Global North*, embedded platforms become a commodity that only a few can afford. DMI researchers, artists and students from Latin America have discussed at great lengths the effects that the lack of access to these otherwise common resources have on their practices. And while some solutions have been proposed, a large gap can still be perceived. By means of appropriating possibly the most widespread and accessible technology in the world (Android) and turn it into an embedded platform, LDSP creates an effective opportunity to close this gap. Throughout the paper, we provide technical details of the full LDSP environment, along with insights on the surprising performances of the first DMIs that have been designed with it.

Author Keywords

Embedded audio, sensors, mobile, Android, LATAM, equity, sustainability

CCS Concepts

•Applied computing → Sound and music computing; •Human-centered computing → Smartphones; •Social and professional topics → Cultural characteristics;

1. INTRODUCTION

Latin America (LATAM) is a pluralistic territory thriving with music and musical instruments. The *cosmovisiones* found across this large geographical area are diverse but share similar colonization and migrations processes, with the socio-economical and political consequences those entail. LATAM musicians, researchers and designers have largely contributed to NIME through both the similarities and the specificities of this landscape, by blending technology with cultural heritage [16, 27] and often re-inventing the very concept of technological innovation [33, 35]. However, we cannot ignore that in LATAM the mere process of designing a digital musical instrument (DMI) comes with a set of challenges that are mostly unheard of within the rest of the NIME community. In particular, we refer to the increased difficulties in accessing technologies and components due to unfavorable local currency value, disproportionate import taxes and shipping costs/times, as well as a general scarcity of materials that in other regions are plenty available off the shelf [3, 33, 35].

One of the most conspicuous embodiments of this divide with the *Global North* is access to embedded platforms. Over the last decade, these technologies have radically changed the DMI ecosystem [6, 19], by introducing new best practices and performance standards that are often out of reach for designers located in LATAM. And while not applying to professionals or to creatives supported by institutional funding (see for example [27]), this gap has been openly lamented by individual researchers, independent artists and students alike [3].

As much as this can be understood as solely an economic issue that LATAM faces, we can understand it as well as an epistemological issue in which technology-based or electronic arts at NIME—but in general too—are usually tied to the notion of progress, of 'new is better'; and therefore, to capitalism and coloniality. This is clearly an unsustainable relationship [2], as the process of creating new technologies and their discard negatively affects land, water, air and in turn all living beings, including us.

In line with the much needed streams of environmental and socio-cultural awareness that have pervaded NIME over the last few years [11, 31, 33, 26], in this work we present *LDSP*, a novel sustainable technology aimed at democratizing embedded DMI design and audio programming beyond the confines of the Global North. LDSP is a free, open-source environment designed to repurpose Android phones as embedded audio programmable platforms, and capable of obtaining impressive computational and musical performance even from devices that are deemed obsolete, and would otherwise be discarded in the pile of ever-growing e-waste.

NIME'23, 31 May–2 June, 2023, Mexico City, Mexico.

2. BACKGROUND

2.1 Embedded DMI Platforms

Several embedded platforms for DMI development are available today, stemming from both academia and the industry. A non-exhaustive list may include fully open-source/open-hardware solutions like Axoloti[1], Bela [19] and Prynth [13], that leverage custom as well as off-the-shelf embedded hardware. Alternative options are the Oopsie-Daisy combo [37] and the recent RNBO[2], both enabling to run code deriving from the Max ecosystem on embedded boards. A further notable entry in this list is Elk Audio OS[3], an advanced open-source operative system (OS) that runs on both the Raspberry Pi and custom boards, and that is geared towards embedded VST design as well as wireless connectivity [34].

Many of these platforms allow for low-level audio programming. With 'low-level' we refer to the capability of applications to communicate almost directly with the audio hardware. In technical terms, this means that the designed code reads and writes samples to and from the very memory buffers that are used by the audio driver in kernel space [19]. This scenario is in stark contrast with the case of general purpose devices like laptops and mobile phones [40, 36], where the presence of several intermediate layers (e.g., audio servers, USB drivers) may heavily impinge on the timing of such audio data exchange. The 'simpler', more direct implementations of embedded platforms allow for audio applications to run with much smaller buffer sizes and, in some cases, to bring down latency below 1 ms [19, 34].

The ability to run code very close to the hardware is particularly convenient in the context of DMI design. Many embedded boards are equipped with general purpose input/output pins, that connect directly into the CPU and allow user applications to interface with a variety of external analog and digital devices, like sensors, motors and custom circuitry. When timed by the tight scheduling of the embedded audio pipeline, the transfer of signals to and from such external devices may happen in an isochronous fashion [39, 24], minimizing jitter and providing an almost instantaneous response to any control input [20].

2.2 Democratization, Needs and Resources

The release of DMI development platforms is part of a larger process, often referred to as democratization of DMIs [5, 15]. In the context of NIME, this term has slowly become synonym of growing availability/accessibility of technology and knowledge pertaining to the design and to the use of DMIs [22]. And while benefiting the whole community, this practical—as well as political—agenda has particularly targeted those people that have been historically underserved, due to factors like socioeconomic status, geographical location or gender. From a hardware standpoint, the inception of DMI democratization can be traced back to the early 2010s, when technological breakthroughs brought to the explosion of mobile phones and game controllers. Such popular consumer devices provided, for the first time, wide access to refined sensors outside the realm of academic research [30]. In a similar fashion, the availability of embedded DMI platforms that arose in the following years started democratizing the exploration of a new generation of musical devices, characterized by self-contained and highly responsive designs [6, 23].

[1]http://www.axoloti.com/

[2]https://cycling74.com/products/rnbo

[3]https://www.elk.audio/how-elk-audio-os-works

However, the impact of embedded platforms appears to be heterogeneous across populations of designers and musicians with different needs and resources. Marasco discusses the use of single board computers in laptop orchestras [18] and underlines how, in this context, many of their advantages are outweighed by the lack of screens, keyboards and trackpads. As opposed, *"on a laptop or mobile device, these features provide intuitive means for interacting with virtual instruments [...] during performances"*. Calegario et al. deem embedded platforms not ideal to turn design concepts into functional prototypes, mainly due to long design/deployment/test time frames and to high technical requirements [9]. To ease this important aspect of the DMI crafting process, they propose a toolkit for computers/laptops composed of a set of configurable sensors and modular structures, and reproducible via widely available digital fabrication techniques.

When shifting the focus to DMI designers and creatives located in the Global South, the major issue associated with the adoption of embedded platforms becomes sheer *inaccessibility*. In line with what discussed in Section 1, Tragtenberg et al. explain how in many LATAM countries the price tags of the more advanced embedded platforms are simply beyond reach for most of the population [33], while Vieira and Schiavoni provide a dramatic numerical comparison between the minimal wages in Brazil and the prices of off-the-shelf music technologies sold in the domestic market [35]. In light of these analyses, it is not surprising that in the survey distributed by Avila et al. in 2021 [3] low-level audio and sensor programming does not emerge as a very common practice within the LATAM NIME community.

2.3 Technological Disobedience

LATAM designers often resort to build their instruments using technologies that are perceived as inferior compared to the *"leading edge"* embedded solutions favored by the Global North—and by NIME in general [3]. Nonetheless, the presence of LATAM authors in NIME have been consistent over the last years [15], with many works celebrating the *"resistance to the scarcity of material resources and technological access"* [33]. Such technological disobedience can be appreciated in Arango and Iazzetta's PICO [1], an embedded audio effect for guitar that *"would run with less latency with the Bela device"*; yet the authors put accessibility and cultural meaning above sheer performance (*"the truth is that these materials are really hard to find in the region where PICO has a sense and where its possible constructors live"*). Vieira and Schiavoni go a step forward, and with Fliperama release the blueprint for a do-it-yourself MIDI controller that is economically and technologically viable even to the poorest populations [35].

Another common approach reported in the literature is the musical appropriation of old/obsolete technology, often times not even functional anymore. Tragtenberg et al. describe two LATAM DMIs belonging to this category [33]: the first one is an old phone turned into MIDI controller, with in-ear monitor and sampling capabilities via the original handset; the second one consists of a foot-stomp sound trigger, with a custom pressure sensor embedded in insoles and made out of the anti-static foam that wraps electronic components during shipping. It is though interesting to note that this type of musical appropriation and hardware *hacking* is not an exclusive prerogative of LATAM designs (see for example [31] or [41]), and it can be root-traced back to the invention of now popular acoustic as well as electric instruments [33, 42].

2.4 Android Phones and DMI Design

Android phones are powerful and accessible devices that have the potential to become a gateway to low-level audio and sensor programming across the Global South. However, their use in NIME and DMI design is limited due to various issues.

Android is one of the most widely adopted OSs for consumer devices [4, 7, 28], with billions of users. It is also the most prevalent OS in LATAM[4], where possessing an Android phone is more common than owning a computer in the least privileged areas of the region[5]. It is worth noting that even the more affordable Android phones have CPU and memory specifications that compare with or top those of many DMI platforms. Additionally, every device comes equipped with a plethora of built-in sensors, 'output devices' (e.g., LEDs/flashlight, vibration motor) and network capabilities.

From a software perspective, Android may be considered as a 'modified' version of Linux, with which it shares the kernel and most of the file system. All its components are open-source, except for *vendor blobs* that may be required to support hardware-specific functionalities, such as accessing the camera sensor or the audio card. Every Android ROM (i.e., the combination of firmware and OS) comes with an application framework that makes development effectively hardware-agnostic. This has been an attractive feature to DMI designers since the release of the first Android phones, and helped stem a new branch of NIME research that focuses on mobile music making [29, 12].

Several environments and applications have been released that are capable of turning Android phones into platforms for DMI development. Some examples are Nexus [32], exclusively based on web-technologies, and MoMubPlat [17], that capitalizes on Pure Data and libpd. Yet, the project that possibly makes the most of the capabilities of Android and smartphones overall is faust2api [21], which provides a convenient pipeline for the deployment of Faust programs on mobile phones. The application supports optimized Faust audio/sensors processing code as well as graphical user interfaces, and it is geared towards the exploration of the 'mechanical' and acoustic features of handheld devices.

However, an important issue with the development of applications targeting Android is related to *latency*. On devices pre-dating 2016, the measured audio latency is *"dreadful [...] (greater than 200ms), discarding any potential use in a musical context"* [21]. Even on newer devices, only a handful of models seem to be able to stay below the 20 ms markfootnote`https://superpowered.com/latency` [Accessed on 2023/4/07]. This issue has been known to DMI design communities for a long time (see for example [8]) and is at odds with widely accepted NIME discourses on responsiveness and intimacy [20, 39]. The reason behind this technical limit is well explained by Villing et al. [36], and it boils down to the structure of the very application framework that grants hardware-agnostic development. More specifically, even what appears to be native code (like programs written in Faust, or even in C++ directly using the Android NDK) has to go through a multi-layer audio stack before eventually being able to exchange samples with the audio driver in kernel space. During interaction, the scheduling inconsistencies caused by this mechanism are perceived as latency that is both very large and highly variable. The same Villing et al. propose a solution to keep latency deviation within a 16 ms range, but do not address the actual magnitude of the perceived delay.

More recently, Balsini et al. [4] have described a promising technique to bring down latency to values that may satisfy the requirements of DMIs. Experiments carried out on a high-end HiKey 960 board report an impressive decrease of one-way audio latency (output buffer only) from 26.67 ms to 2.7 ms. However, this solution works only on devices running Android 8.0 and above, and no tests have been carried out on actual phones.

3. LDSP

LDSP is a novel environment designed to turn Android mobile phones into accessible and high-performance embedded DMI platforms. In light of what discussed in the previous section, this target can be achieved if the following three major requirements are fulfilled: (1) the environment should grant low-level access to the hardware, to maximize resource utilization and minimize latency (ideally matching the current stat-of-the-art [4]); (2) it should be compatible with the largest number possible of phones and Android versions (much like [17] or [32]); and (3) it should include a software framework specifically designed to facilitate synthesis, processing and interaction (similar to what accomplished in [21]).

What we propose is a set of tools that allow developers to completely by-pass the Android application framework and write native C++ audio code that can be invoked via command line. Instead of creating an app that is forced to run on top of the audio stack (and within the boundaries of the Android Run-Time Environment/Dalvik Virtual Machine), LDSP builds Linux ELF (Executable and Linkable Format) files that are dealt with directly by the kernel, and have potential access to memory and drivers/hardware resources. In other words, LDSP permits to use Android phones as generic Linux embedded boards, the only caveat being that the phone needs to be *rooted*. This approach has a tremendous impact on the overall performance of the written code.

At the current stage of development, LDSP is meant to be downloaded to a host computer. Developers can use it to cross-compile their native application and then deploy it to their Android phone. The main component of the environment is a C++ framework, with a simple API and including libraries as well as examples tailored to mobile audio development (more on this in Section 3.2). The compilation process uses a set of scripts based on CMake and Ninja[6], and requires the installation of the Android NDK. The latter is needed to link the executable against Android's standard libraries. All these tools are free and cross-platform; as a result, LDSP seamlessly runs on Linux, macOS and Windows. Furthermore, they can be easily accessed via code editors or integrated development environments, allowing developers to include LDSP in their preferred coding pipeline. The LDSP environment can be accessed here: `https://github.com/victorzappi/LDSP.git`.

3.1 C++ Framework: Audio and Control

At the core of the LDSP C++ framework lies a custom audio engine. It is built around the TinyALSA library[7] and it is designed to control directly the Advanced Linux Audio Architecture (ALSA) kernel drivers. More specifically, the audio engine provides an API to open any of the

[4] `https://www.canalys.com/newsroom/latam-smartphone-market-Q1-2022` [Accessed on 2023/04/07]

[5] This has been personally witnessed by the first author (who is originally from a LATAM country) and confirmed by other members of the LATAM NIME community during informal conversations with the rest of the LDSP team.

[6] `https://ninja-build.org/`

[7] `https://github.com/tinyalsa/tinyalsa`

capture and playback devices available on the phone, to synchronize them and to set up a user-defined audio callback function—called 'render'. This render function runs on a top-priority dedicated thread; it is tightly synchronized with the hardware audio buffering mechanism and includes pointers to the regions of memory (buffers) used by the driver to exchange samples with the devices. Figure 1 depicts the equivalent audio stack and compares it with that of the modern Android AAudio API. Thanks to LDSP's 'simple' audio implementation, the use of the phones' resources can be optimized to run fairly advanced audio algorithms and/or buffer sizes that are typically prohibitive for Android apps (see next section). The first version of Android that included support for ALSA was Android 2.3 (Gingerbread), which was released in December 2010. This means that, at the time of writing, audio code written in LDSP can potentially run both on brand new phones and on phones that are 13 years old.

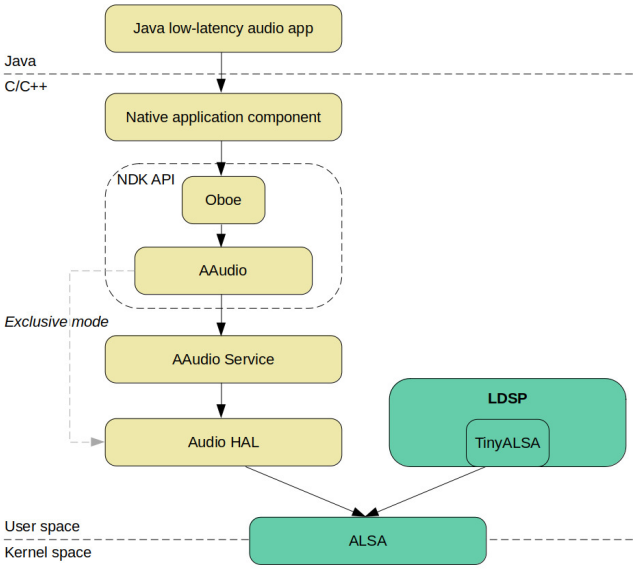

Figure 1: Audio stack comparison: LDSP stack (on the right) and simplified AAudio stack (on the left). Diagram adapted from [4].

The LDSP API exposes functions to read input data incoming from the built-in sensors too. As soon as it is launched, any LDSP application probes the hardware it is running on, with the aim to detect what sensors are present on the phone. We started by adding support for the most basic sensors, including: accelerometer, magnetometer, gyroscope, light sensor, proximity sensor, power/volume buttons (also via audio jack) and multitouch screen. However, virtually any sensor present in the Android NDK can be easily added to the framework, due to the fact that LDSP accesses sensors via the same library (*libandroid*) and the same device nodes used by the Android run-time environment. The main advantage of this approach is the complete portability of the code across phones, for vendor blobs are transparently used in the process. However, as a result sensors are sampled via a dedicated thread that runs asynchronously to the audio thread, making isochronous audio output and control input not possible. This issue (namely jitter) is though partially mitigated on LDSP by always polling sensors at their maximum supported rate, and by allowing for relatively small audio buffer sizes.

A second control component of the API deals with output data streams. These control outputs can target the built-in flashlight, the LCD screen back-light, the status LED (including separate RGB channels where available), the vibration motor and even the back-light of the front touch buttons that Android phones used to have before 2013. Differently from the case of sensors, LDSP includes a custom low-level algorithm to locate and write to output device nodes, that does not rely on vendor blobs. The implementation grants direct control of the output drivers, while still maximizing portability. As a result, any control data written by the application is dispatched right at the end of the render function, allowing for what we could call 'soft isochrony' of audio and control outputs.

3.2 Libraries, Examples and Configuration

The last component of the C++ framework is a set of libraries designed to support user application design. At the current stage of development, this set includes libraries for audio (e.g., audio file read/write, filters) and libraries for data transfer/management (e.g., Open Sound Control, JSON, XML). Since many phones can interface with MIDI controllers via USB On-The-Go, a MIDI library is also part of the framework.

LDSP also comes with a set of example projects, ranging from a basic sinusoidal bleep to fully fledged interactive applications. Developers can build and run them on their phone, but also inspect and re-use their source code. The minimal component of both user and example projects is a render file, whose template includes an empty definition of the render function (see Section 3.1). This detail, along with the overall arrangement of libraries and examples, is inspired to Bela's code structure [19].

A single entry-point script allows developers to quickly configure CMake to build a chosen LDSP project. Two main arguments need to be passed to the script: the path to the project and the version of Android running on the target phone. The script also requires the path to a hardware configuration file, that needs to be populated by the developer. This small file is based on a predefined JSON template (available in the LDSP environment) and lists important hardware details of the target phone's model, that are necessary at both compile and run time. The most relevant entries are the labels that the manufacturer gave to the hardware mixer presets for audio routing. They can be found in a standard Android XML file stored on the phone, and represent the paths to line-output, built-in speaker and handset speaker (for output streams), and from line-input, built-in mic and handset mic (for input streams). The labels are also hard-coded in the vendor blobs, along with the exact procedure to recall the presets and switch paths on and off. But, unfortunately, this part of the blobs cannot be accessed via *libandroid*. We solved this problem by reverse-engineering the mixer preset algorithms, leveraging the low-level mixer capabilities of TinyALSA. As a result, once the labels are in place, developers can choose to run their LDSP applications with any combination of input and output paths, regardless of the selected playback and capture devices.

The creation of the configuration file is probably the only non-straightforward operation required to use LDSP. To ease this process, the environment includes a set of simple scripts that are meant to be run on the phone and that help extract relevant information from the hardware. Moreover, we are gradually creating a collection of ready-made configuration files that cover all the phones where LDSP have been tested, organized by brands and models.

To deploy and test their application, developers can use the main entry-point script. The script uses the Android

Debug Bridge (which needs to be installed on the host machine) to push the LDSP executable to the connected phone and run it. Once testing is complete, any terminal emulator can be used to run and stop LDSP applications directly from the phone, without the need for a USB cable connection. An alternative workflow consists of using SSH (a secure shell) to move files to the phone and run executables remotely—a common scenario when programming generic embedded boards. This requires to install one of the many SSH server apps that are available for Android and that can leverage the WiFi capabilities at disposal on virtually any phone.

4. LDSP EXAMPLE DMIS

Aside from technical details, we believe that the best way to showcase LDSP's potential is to introduce some DMIs designed with it. As emerging from the experience of Villing et al. [36], the performance of mobile devices largely depends on the specific hardware that powers them. And more in general, the unique set of features and limitations that characterize a phone may make its use particularly convenient in a specific use-case scenario, while totally unsuitable in another one [21]. Intrigued by these challenges, over the month prior to the writing of this work, we decided to use LDSP first to probe the characteristics of three Android phones and then to build simple DMIs with them. The intent was to discover where the specific features of these phone could shine in the context of mobile music, and develop interesting designs that could capitalize on them.

The first phone we worked on is an LG G2 Mini (Figure 2), that used to belong to one of the authors. It runs a Lineage OS 14.1 custom ROM (equivalent to Android 7.1.2). This phone was released in April 2014 and has a very 'low-spec' design compared to more modern devices. Nonetheless, after little experimentation we realized that on this phone LDSP is capable of running full-duplex audio applications with buffer size of 64 sample, at a sample rate of 48 kHz. Put down in numbers, this roughly translates into 2.7 ms of round-trip latency[8]. Such an unexpected result pushed us to turn this phone into a programmable effects pedal for guitar (Figure 3). We called this device *LDSPowl*, as a tribute to Rebel Technology's OWL pedal [38].

Figure 2: The two LG G2 Mini phones used for DMI design, running Lineage OS 14.1 (on the left) and stock Android 4.4 (on the right).

Our LDSPowl pedal includes a bare-bone impedance-bridging circuit, composed of two buffers and four jack sockets, that permits to transfer the audio signal from the guitar to the phone's line-input and then from the phone's line-output to an amplifier. An inexpensive combo-jack splitter is used to separate input and output channels of the phone. The full setup is depicted in Figure 4. The first 'patch' that we tried out is a simple delay, whose feedback control is controlled by the x-axis tilt of the phone. In pure *gambiarra style* [33], we hooked up the phone to the guitarist's foot with an elastic band.

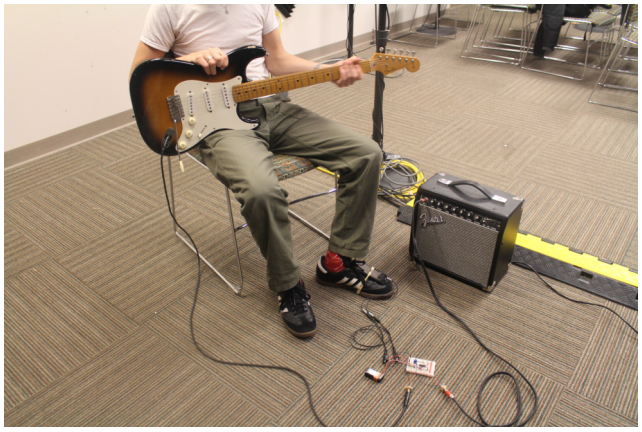

Figure 3: A guitarist playing with the LDSPowl programmable effects pedal.

The second DMI we designed is built on another LG G2 Mini that was donated to us (Figure 2). Its hardware is identical to the first phone, but it still runs the 2014 stock ROM it was shipped with (Android 4.4). This detail seems enough to alter its audio performance, for on this second phone the smallest buffer size that LDSP can support in full-duplex mode is 128 samples. However, when audio capture is disabled and the phone runs as a playback device only, LDSP applications are again capable of reaching the 64 sample mark (equal to an output buffer period of 1.4 ms). We then decided to turn this device into a synthesizer and more specifically into a touch-controlled granular synthesizer, called *DedoGranular*. The application loads from memory two audio files and uses the granulator available in the LDSP library to generate a continuous audio stream. The synthesis is triggered by the presence of the finger on the touchscreen; its y-location determines the time location of the grains, while by changing it x-location is possible to cross-fade between the granulation of the two files.

Figure 4: The do-it-yourself bidirectional direct-input box that is part of LDSPowl.

The third phone was again donated to us and consists of a 2015 Huawei P8 Lite, with a stock Android 5.0 ROM. Despite running on more modern hardware (64-bit octa-core

[8]This number reflects the latency due to the buffering mechanism only; more experiments are planned to measure the impact of hardware routing and codecs.

Cortex-A53 @ 1.2 GHz), this device showcased some quirks when tested with LDSP. The only audio device present on this phone seems capable of sustaining smooth audio playback only with a buffer of 960 samples at 48 kHz, regardless of the CPU load. Latency and jitter resulting from this configuration are hard to accept in the context of embedded DMI design. We then decided to turn this phone into a wireless controller. By discarding any audio output capability, we were able to run LDSP with a buffer of 512 samples, and leverage the resulting audio loop to read and transmit sensor data via Open Sound Control at a steady rate of 93.75 Hz—very close to the optimal interval empirically found by McPherson et al. [20]. As pointed out by the same authors, controllers—and in particular wireless ones—are by their very nature incapable of high degrees of responsiveness. Their ideal use-case consists of sound applications that are not too time-critical, like the sonification of continuous gestures. We followed this lead, and completed our design by mapping the data picked up by the 3-axis magnetometer built into the phone to timbral controls of a simple Max for Live project, running on a laptop connected to the same WiFi network as the phone. The result is the *MagSniffer*, a tetherless application for the sonification of magnetic fields (Figure 5). The device provides also vibratory feedback to signal the overall magnitude of the field being picked up. Short demos of the MagSniffer, as well as of LDSPowl and DedoGranular can be watched at this link: `https://youtu.be/y35tPl5mjGo`.

Figure 5: The Huawei P8 Lite running the MagSniffer to sonify the magnetic field generated by audio equipment.

5. CONCLUSIONS

In this paper we presented LDSP, a free and open-source environment that turns Android phones into embedded platforms for DMI design. Many examples of musical appropriation of tools and materials have long been discussed in the context of NIME [14, 42]. Yet, very few allow designers to subvert an entire *system* [10] like LDSP does with Android. Although at the current stage of development some elements of its workflow may still feel like a hack (e.g., rooting the phone, peeping into configuration files), this type of practices are no strangers to creatives in LATAM, where a tradition of technological disobedience fosters wild hacking and repurposing of technologies—out of necessity or choice [25, 33, 35].

Currently, most of our efforts are geared towards improving LDSP's usability and accessibility. We are working on extending libraries and examples, as well as on the creation of how-to guides. And since some phone manufacturers may employ firmware that diverges from Android standards, it is imperative to test LDSP on as many brands and models as possible. This is being accomplished thanks to both phone donations and remote collaborations.

The future development of LDSP revolves around moving the environment to the phones. We are starting the design of an app that will combine the compilation tools and the framework, allowing development both from a host computer and from the phone itself. This will simplify the configuration process, while making LDSP accessible even to creatives that do not own a computer.

6. ACKNOWLEDGMENTS

We are very grateful to João Tragtenberg, Juan Pablo Martinez Avila and the whole LATAM NIME community for the continuous support. We would like to thank also Spencer Rosenfeld and Jason Hoopes for their help during the development of the first LDSP DMIs.

7. ETHICAL STANDARDS

This project stems from the awareness of the uneven distribution of hardware according to geopolitical location as well as from the importance of creating more sustainable NIMEs. For this reason, this project actively targets equity and accessibility through the use of Android phones. The aforementioned awareness is rooted in the needs we found in LATAM, considering the experiences of one of the authors who is from Argentina, as well as the literature and examples we have cited. It is important to note that even if we are focusing on the LATAM experiences, other 'souths' might have similar experiences with the appropriation of technology. Our worlds are complex, where the periphery and center are no longer so identifiable or clear cut, so we believe this project can have an impactful effect in other places as well. Hopefully appropriated by the people that use it.

Our current system of manufacturing technology thrives on producing more by discarding more easily and frequently. To insist on designing and manufacturing like this is to choose not to see our limited resources and the damage this cycle creates. Designing for a different set of values, knowledges and approaches from different geographies, such as the ones discussed above, can allow us to reimagine and reconstruct more sustainable worlds. The NIME community has considered the importance of discussing sustainability in our practices, and we believe environmental issues are addressed in our project as a part of the material design of the artifact.

8. REFERENCES

[1] J. J. Arango and F. Iazzetta. Pico: A portable audio effect box for traditional plucked-string instruments. In *NIME 2019*, pages 355–360, 2019.

[2] Z. Argabrite, J. Murphy, S. J. Norman, and D. Carnegie. Technology is land: Strategies towards decolonisation of technology in artmaking. In *NIME 2022*, 2022.

[3] J. P. M. Avila, J. Tragtenberg, F. Calegario, X. Alarcon, L. P. C. Hinojosa, I. Corintha, T. Dannemann, J. Jaimovich, A. Marquez-Borbon, M. M. Lerner, et al. Being (a) part of nime: Embracing latin american perspectives. In *NIME 2022*, 2022.

[4] A. Balsini, T. Cucinotta, L. Abeni, J. Fernandes, P. Burk, P. Bellasi, and M. Rasmussen. Energy-efficient low-latency audio on android. *Journal of Systems and Software*, 152:182–195, 2019.

[5] T. J. Barraclough, D. A. Carnegie, and A. Kapur. Musical instrument design process for mobile technology. In *NIME 2015*, pages 289–292, 2015.

[6] E. Berdahl and W. Ju. 2011: Satellite ccrma: A musical interaction and sound synthesis platform. In *A NIME Reader*, pages 373–389. Springer, 2017.

[7] M. Butler. Android: Changing the mobile landscape. *IEEE Pervasive Computing*, 10(1):4–7, 2011.

[8] B. Cahill and S. Serafin. Guitar simulator: An audio-haptic instrument for android smartphones. In *The Seventh International Workshop on Haptic and Audio Interaction Design August 23-24 2012 Lund, Sweden*, pages 19–20, 2012.

[9] F. Calegario, M. M. Wanderley, S. Huot, G. Cabral, and G. Ramalho. A method and toolkit for digital musical instruments: generating ideas and prototypes. *IEEE MultiMedia*, 24(1):63–71, 2017.

[10] A. Dix. Designing for appropriation. In *Proceedings of HCI 2007 The 21st British HCI Group Annual Conference University of Lancaster, UK 21*, pages 1–4, 2007.

[11] E. Dorigatti and R. Masu. Circuit bending and environmental sustainability: Current situation and steps forward. In *NIME 2022*, 2022.

[12] G. Essl and S. W. Lee. Mobile devices as musical instruments-state of the art and future prospects. In *Music Technology with Swing: 13th International Symposium, CMMR 2017, Matosinhos, Portugal, September 25-28, 2017, Revised Selected Papers 13*, pages 525–539. Springer, 2018.

[13] I. Franco and M. M. Wanderley. Prynth: A framework for self-contained digital music instruments. In *Bridging People and Sound: 12th International Symposium, CMMR 2016, São Paulo, Brazil, July 5–8, 2016, Revised Selected Papers 12*, pages 357–370. Springer, 2017.

[14] M. Gurevich, P. Stapleton, and A. Marquez-Borbon. Style and constraint in electronic musical instruments. In *NIME*, pages 106–111, 2010.

[15] L. Hayes and A. Marquez-Borbon. Nuanced and interrelated mediations and exigencies (nime): addressing the prevailing political and epistemological crises. In *NIME 2020*, 2020.

[16] L. P. C. Hinojosa. Kanchay_yupana: Tangible rhythm sequencer inspired by ancestral andean technologies. In *NIME 2022*, 2022.

[17] D. Iglesia and I. Intermedia. The mobility is the message: The development and uses of mobmuplat. In *Pure Data Conference (PdCon16). New York*, 2016.

[18] A. T. Marasco. Approaching the norns shield as a laptop alternative for democratizing music technology ensembles. In *NIME 2022*, 2022.

[19] A. McPherson. Bela: An embedded platform for low-latency feedback control of sound. *The Journal of the Acoustical Society of America*, 141(5):3618–3618, 2017.

[20] A. P. McPherson, R. H. Jack, G. Moro, et al. Action-sound latency: Are our tools fast enough? In *NIME 2016*, 2016.

[21] R. Michon, Y. Orlarey, S. Letz, D. Fober, and C. Dumitrascu. Mobile music with the faust programming language. In *Perception, Representations, Image, Sound, Music: 14th International Symposium, CMMR 2019, Marseille, France, October 14–18, 2019, Revised Selected Papers*, pages 307–318, 2021.

[22] F. Morreale, A. Bin, A. McPherson, P. Stapleton, and M. Wanderley. A nime of the times: developing an outward-looking political agenda for this community. In *NIME 2020*, 2020.

[23] F. Morreale, G. Moro, A. Chamberlain, S. Benford, and A. P. McPherson. Building a maker community around an open hardware platform. In *CHI 2017*, pages 6948–6959, 2017.

[24] M. Neupert and C. Wegener. Isochronous control+ audio streams for acoustic interfaces. In *Proceedings of the 17th Linux Audio Conference (LAC-19)*, 2019.

[25] I. Orobitg, A. Subieta, F. Uslenghi, and F. Wiman. El desarrollo de la música electroacústica en buenos aires. 2003.

[26] L. Pardue and S. A. Bin. The other hegemony: Effects of software development culture on music software, and what we can do about it. In *NIME 2022*, 2022.

[27] J. Ramos, E. R. Calcagno, R. oscar Vergara, P. Riera, and J. Rizza. Nime 2022. In *NIME 2022*, 2022.

[28] A. A. Sheikh, P. T. Ganai, N. A. Malik, and K. A. Dar. Smartphone: Android vs ios. *The SIJ Transactions on Computer Science Engineering & its Applications (CSEA)*, 1(4):141–148, 2013.

[29] A. Tanaka. Mobile music making. In *NIME 2004*, pages 154–156, 2004.

[30] A. Tanaka. Mapping out instruments, affordances, and mobiles. In *NIME 2010*, 2010.

[31] T. Tate. The concentric sampler: A musical instrument from a repurposed floppy disk drive. In *NIME 2022*, 2022.

[32] B. Taylor, J. T. Allison, W. Conlin, Y. Oh, and D. Holmes. Simplified expressive mobile development with nexusui, nexusup, and nexusdrop. In *NIME 2014*, pages 257–262, 2014.

[33] J. Tragtenberg, G. Albuquerque, and F. Calegario. Gambiarra and techno-vernacular creativity in nime research. In *NIME 2021*, 2021.

[34] L. Turchet, S. J. Willis, G. Andersson, A. Gianelli, and M. Benincaso. On making physical the control of audio plugins: the case of the retrologue hardware synthesizer. In *Proceedings of the 15th International Conference on Audio Mostly*, pages 146–151, 2020.

[35] R. Vieira and F. Schiavoni. Fliperama: An affordable arduino based midi controller. In *NIME 2020*, 2020.

[36] R. Villing, V. Lazzarini, D. Czesak, S. O'Leary, and J. Timoney. Approaches for constant audio latency on android. In *DAFx-15*, 2015.

[37] G. Wakefield. A streamlined workflow from max/gen˜ to modular hardware. In *NIME 2021*, 2021.

[38] T. Webster, G. LeNost, and M. Klang. The owl programmable stage effects pedal: Revising the concept of the on-stage computer for live music performance. In *NIME 2014*, pages 621–624, 2014.

[39] D. Wessel and M. Wright. 2001: Problems and prospects for intimate musical control of computers. In *A NIME Reader*, pages 15–27. Springer, 2017.

[40] M. Wright, R. J. Cassidy, and M. Zbyszynski. Audio and gesture latency measurements on linux and osx. In *ICMC*, 2004.

[41] K. Yerkes, G. Shear, and M. Wright. Disky: a diy rotational interface with inherent dynamics. In *NIME 2010*, pages 150–155, 2010.

[42] V. Zappi and A. P. McPherson. Dimensionality and appropriation in digital musical instrument design. In *NIME 2014*, pages 455–460, 2014.