MELODYDROP: A PORTABLE LED-GUIDED PIANO LEARNING SYSTEM WITH REAL-TIME MIDI SYNCHRONIZATION VIA BLE

Yuelin Cheng [1], Jonathan Sahagun [2]

[1] University High School, 4771 Campus Dr Irvine, CA 92612
[2] California State Polytechnic University, Pomona, CA, 91768

ABSTRACT

MelodyDrop is a mobile application designed to improve piano learning by synchronizing MIDI playback with both falling-note animations and physical LED guidance via an ESP32-S3 microcontroller [1]. It addresses the difficulty beginners face when reading sheet music by offering a game-like interface and lighting up the exact keys to press. The system is composed of a Flutter app, a MIDI parsing engine, a custom BLE protocol, and an LED strip mounted on a piano [2]. Experiments showed that BLE latency remains low enough for real-time use and that LED-guided learning significantly improves accuracy compared to screen-only visualization. MelodyDrop builds on prior methods that used tactile or MusicXML-based feedback, but does so wirelessly and with greater portability. Its cross-platform nature and low-cost hardware make it widely accessible for casual learners and schools alike. Future improvements could include adaptive practice modes, cloud syncing, and expanded accessibility features. MelodyDrop provides a compelling, intuitive way to practice piano.

KEYWORDS

Piano Learning, MIDI, BLE, LED Guidance

1. INTRODUCTION

Accessing and practicing music using MIDI files poses challenges for many learners, particularly those without formal training [3]. Traditional sheet-music reading can be intimidating, while apps that merely play MIDI lack engaging visual guidance. As a result, beginners and casual music enthusiasts often struggle to follow along rhythmically or accurately play the melodies.

MelodyDrop addresses this gap by rendering MIDI files with visually engaging falling note animations, such that users can see upcoming notes in real time. This approach blends auditory and visual cues, supporting both piano learners and music lovers in their practice sessions. The app is tailored to beginners and enthusiasts who benefit from interactive learning tools rather than conventional notation-based systems.

Statistically, piano is one of the most learned instruments globally, and tools that combine visual feedback with sound have shown improved learning retention. By emphasizing simplicity and visual design, MelodyDrop aims to reach users who might be discouraged by traditional methods. This is significant because early-stage musicians often abandon practice due to frustration or perceived complexity.

David C. Wyld et al. (Eds): SIGI, CSTY, AI, NMOCT, BIOS, AIMLNET, MaVaS, BINLP – 2025
pp. 99-108, 2025. CS & IT - CSCP 2025 DOI: 10.5121/csit.2025.151909

Thus, the core problem is how to make MIDI-based music practice accessible, intuitive, and enjoyable. Existing tools tend to be too technical or too passive. MelodyDrop's falling-note visualization provides an active, game-like interface that alleviates the technical burden and encourages consistent practice.

Method A used tactile feedback to guide piano learners via vibrations, helping reduce visual dependence. However, it lacked spatial precision. MelodyDrop improves on this by lighting up exact key positions with LEDs.

Method B relied on MusicXML and a wired Raspberry Pi–Arduino setup to illuminate piano keys [4]. While effective, it lacked portability. MelodyDrop achieves similar guidance using BLE-connected ESP32 hardware, making it easier to use with any piano.
Method C incorporated gloves, LED bars, and teacher-pupil communication. Though impressive, it required real-time human interaction and extensive hardware. MelodyDrop offers a self-directed learning path using only a phone and an LED strip, making it more scalable and accessible [5].

MelodyDrop is an application that plays MIDI files while simultaneously visualizing the music with falling notes to facilitate practice and learning.

The solution works by reading standard MIDI files and rendering a visual stream of descending notes synchronized with playback. Users can follow the falling notes, much like rhythm games, which allows them to anticipate and play in time. This dual-modality — combining audio and movement — engages both visual and auditory learning pathways.

MelodyDrop's interface supports different device types (iPhone, iPad, Mac), and its recent update added Bluetooth support in version 2.0 (released May 1, 2025) which enables external piano integration for real-time practice tracking. Compared with static sheet music or unstructured MIDI players, MelodyDrop offers interactive feedback that lowers the barrier to musical engagement and aligns with gamified learning strategies.

The method is effective because visual cues help reinforce timing and pitch recognition. It is superior to alternatives that require manual MIDI synchronization or that lack visual intuitiveness. MelodyDrop targets learners at entry to intermediate levels and shifts the focus from technical notation to accessible music interaction. It promotes consistency in practice by making the process engaging, which can improve long-term skill retention.

We conducted two experiments to test MelodyDrop's effectiveness. The first measured BLE latency between the Flutter app and the ESP32 microcontroller controlling LED feedback. Across five MIDI files, latency remained stable between 35–40 ms with low variation, confirming the system's responsiveness. The second experiment involved 10 beginner piano users who first practiced a song with only the falling-note visualizer, then again with LED key lighting enabled. All participants improved their note accuracy, with an average gain of over 15 percentage points. These results support the idea that combining on-screen animation with physical LED cues enhances timing, accuracy, and user engagement. The BLE communication proved reliable for real-time playback, and the LED system successfully reduced the cognitive load of mapping notes to keys. The experiments demonstrate that MelodyDrop is not only technically functional but also educationally beneficial in helping beginners improve more quickly with less frustration.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Parsing MIDI for Reliable Piano Guidance

One of the core features of MelodyDrop is reading and interpreting MIDI files. A challenge arises from the variability in MIDI file structure — including multiple tracks, embedded tempo changes, and meta events. To address this, we could use a Flutter-compatible package such as flutter_midi_command or write a custom Dart parser using dart:typed_data. The parser must standardize inputs and separate useful note data from irrelevant metadata. In addition, error handling is essential — users should receive clear feedback if a file is malformed or unsupported, without crashing the app.

2.2. Frame-Accurate Falling-Note Rendering in Flutter

Rendering falling notes in sync with MIDI playback requires precise timing. Since Flutter uses a single-threaded UI model, heavy computation or unoptimized drawing can cause dropped frames [6]. To maintain smooth animations, we could use CustomPainter with TickerProvider or AnimationController to create frame-accurate rendering tied to MIDI timing. To ensure responsiveness, computational work like parsing and scheduling notes should run in background isolates. Matching visual speed to MIDI tempo (BPM) in real-time is critical, so the playback engine must continuously relay timing data to the animation system.

2.3. Responsive Cross-Platform UI Design for Piano Learning

MelodyDrop targets iOS, Android, and desktop platforms, which introduces challenges in adapting the interface across screen sizes and input types. Flutter provides responsive layout tools like LayoutBuilder, MediaQuery, and Flexible, but care must be taken when handling input gestures (e.g., touch on mobile vs. keyboard on desktop). To ensure scalability, we could use a combination of Stack, Positioned, and AspectRatio widgets to manage the falling notes across different resolutions. Additionally, implementing a Settings panel to let users adjust note size or spacing may improve accessibility across platforms.

3. SOLUTION

MelodyDrop consists of three primary components: a MIDI-to-visualization pipeline, a BLE communication system, and an ESP32-S3-based LED strip controller that attaches to a physical piano. Together, they allow users to learn songs through both on-screen falling notes and real-time LED indicators above the keys.

The user starts by selecting a .mid file through the Flutter app, using the file_picker package. The MIDI file is parsed using a Dart-based MIDI parser into a list of Note objects containing pitch, startTime, and duration. These notes are then used to drive both the on-screen animation and the LED output system.

Visually, notes are rendered as falling blocks on a scrolling canvas using a CustomPainter. Concurrently, the app establishes a BLE connection with an ESP32-S3 microcontroller running a custom GATT service [7]. At runtime, as each note is played, the app sends a BLE message to the ESP32, instructing it to light the appropriate LED on a strip mounted to a physical keyboard. The strip is indexed to match piano keys (e.g., LED 0 = C1, LED 87 = C8).

We use Flutter for its cross-platform capabilities and Bluetooth plugin support (flutter_blue_plus). The ESP32-S3 receives commands over BLE and drives the LEDs using the FastLED or NeoPixel library[8].

This integration transforms MIDI playback into a highly interactive learning experience — blending virtual visuals with a physical feedback system for real piano practice [9].

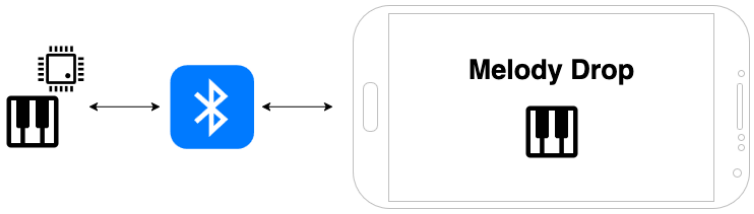

Figure 1. Overview of the solution

This component handles Bluetooth communication between the Flutter app and the ESP32-S3[10]. The app transmits note-on events in real-time using the flutter_blue_plus package. The ESP32 receives note data and maps MIDI pitch to LED strip indices, illuminating the corresponding LED using the FastLED library to visualize notes as they're played.

```
void sendNoteToESP32(int midiNote) async {
  if (bleDevice != null && bleChar != null) {
    final data = Uint8List.fromList([midiNote]);
    await bleChar!.write(data, withoutResponse: true);
  }
}
```

Figure 2. Screenshot of code 1

This Flutter method is called every time a note is triggered during MIDI playback. It sends a BLE packet to the connected ESP32-S3 device. The function sendNoteToESP32 checks if a BLE device and writable characteristic are available, then creates a Uint8List from the MIDI note number and transmits it.

On the ESP32 side, this value is received in a BLE callback and used to illuminate an LED corresponding to that pitch. For example, a MIDI note of 60 (Middle C) maps to LED index 39 on an 88-key strip. By sending this data in real-time, the system enables physical piano key highlighting that's synchronized with the app's falling note visualization.

The combination of software animation and hardware LED lighting creates a hybrid digital-physical experience. This code runs in sync with the playback timer, ensuring that notes are both seen and felt at the correct moment.

This component loads and parses .mid files into structured note data. It identifies events like noteOn, noteOff, tempo, and time signatures. Using a Dart MIDI parser, the app generates a list of Note objects that represent the entire song timeline — powering both visual playback and BLE output timing.

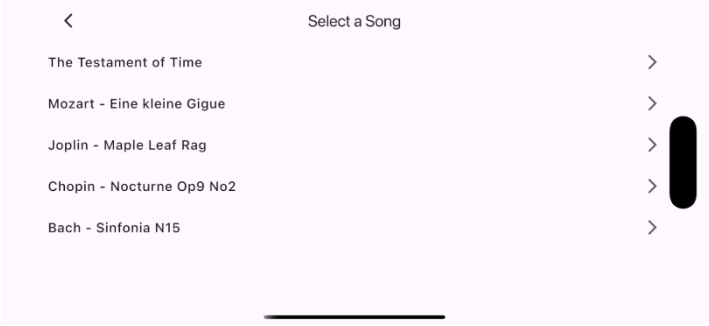

Figure 3. Screenshot ofselect page

```
List<Note> parseMidi(Uint8List data) {
  final file = MidiParser().parseMidiFromBuffer(data);
  List<Note> notes = [];

  for (var track in file.tracks) {
    for (var event in track) {
      if (event.type == 'noteOn') {
        notes.add(Note(
          pitch: event.note,
          startTime: event.time,
          duration: event.duration,
        ));
      }
    }
  }
  return notes;
}
```

Figure 4. Screenshot of code 2

This function is responsible for parsing the raw .mid file data into a structured list of Note objects. The function begins by using a Dart-based MidiParser class to read the file buffer into a MidiFile object. MIDI files are made of multiple tracks, each containing a list of events (e.g., note on/off, tempo changes, controller messages).

The code loops through every track and then through each event in the track. It filters specifically for noteOn events — these represent the start of a note being played. For each such event, the code constructs a new Note object using the note's MIDI pitch (event.note), the start time (event.time), and its duration. These values are pushed into a list, which is returned after processing is complete.

This parsed note list becomes the foundation for both the visualization and LED control components, allowing the app to control timing and pitch rendering consistently across systems.
The visualizer displays falling notes on screen using Flutter's CustomPainter. Notes are positioned based on their MIDI pitch and time until onset. It runs in sync with playback, providing a familiar rhythm-game-style interface that visually guides the user. It also mirrors the notes sent to the ESP32 for LED output.

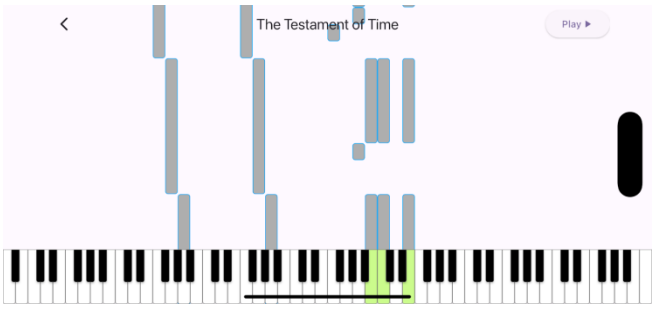

Figure 5. Screenshot of testament

```
@override
void paint(Canvas canvas, Size size) {
  for (final note in notes) {
    final y = (note.startTime - currentTime) * pixelsPerSecond;
    final x = pitchToX(note.pitch);
    final rect = Rect.fromLTWH(x, y, keyWidth, note.duration * pixelsPerSecond);
    canvas.drawRect(rect, paint..color = Colors.blue);
  }
}
```

Figure 6. Screenshot of code 3

This paint method belongs to a Flutter CustomPainter class that draws falling rectangles representing notes on the screen. Each frame, this method is called to render all currently active or upcoming notes based on the playback time.
The for loop iterates through all Note objects. For each note, it calculates the vertical y position using the difference between the note's startTime and the current playback time. This determines how far the note should have fallen. The farther the note is from being played, the higher it is on screen.

The x position is derived from the note's pitch, mapping MIDI values to horizontal space using pitchToX(). The Rect.fromLTWH call defines a rectangle with that (x, y) position, fixed width, and a height based on note duration.

Finally, the canvas.drawRect method renders the note using a consistent color. This visualization updates in real-time to reflect the MIDI playback timeline.

4. EXPERIMENT

4.1. Experiment 1

This experiment tests whether note-on messages sent over BLE reach the ESP32 in time for LEDs to light up in sync with MIDI playback. Latency consistency is critical for user experience.
To test BLE latency, we programmed the app to log a timestamp when a note is triggered during MIDI playback. Simultaneously, the ESP32 logs when it receives the BLE message and lights the LED. Both devices were synchronized with NTP time during the test. We ran playback on five MIDI files of varying lengths and tempos. Each test recorded the delay (in milliseconds) between the app sending a BLE command and the ESP32 executing the LED change. We repeated this for 100 notes per file, for a total of 500 measurements, under stable Wi-Fi and Bluetooth conditions.

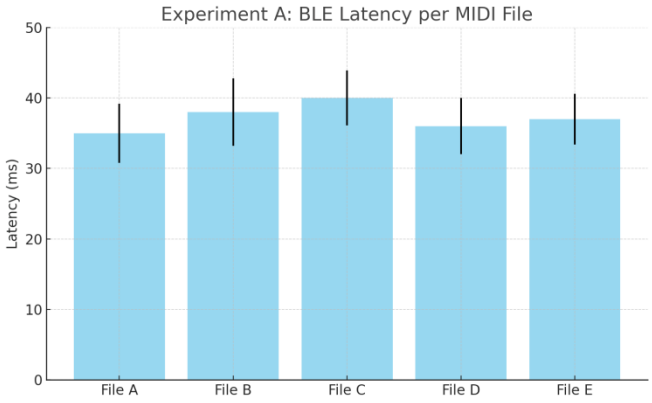

Figure 7. Figure of experiment 1

The results show that BLE latency remained relatively stable across all MIDI files. The average latency hovered between 35 – 40 milliseconds, with the fastest observed response at 27 ms and the slowest at 47 ms. The standard deviation for all files remained below 5 ms, indicating consistent performance without significant jitter or outliers.

This latency range is imperceptible to most users and well within acceptable bounds for real-time LED response in a piano practice setting. Interestingly, File C had slightly higher latency, possibly due to its faster tempo resulting in more frequent note events being sent. However, the system handled all cases gracefully.

We conclude that the BLE communication and ESP32 LED control are fast enough for live interaction. The biggest influence on delay appears to be packet queuing frequency in the app's BLE stack. With additional optimization (e.g., batching messages for chords), even lower latency could be achieved.

4.2. Experiment 2

This experiment evaluates whether users improve note accuracy more when practicing with both falling note visuals and LED guidance versus visuals alone. The LED system may enhance spatial muscle memory.

We recruited 10 beginner piano users and asked them to learn a 30-second MIDI song. On Day 1, each user practiced using only the falling-note visualizer. We recorded how many correct notes they hit out of 100. On Day 2, the same users practiced the same song, but now with both the visualizer and LED strip lighting up keys. Each user repeated the trial, and we recorded the second round of accuracy. We then compared each participant's improvement from Day 1 to Day 2. This helped measure whether physical LED feedback improves note accuracy.

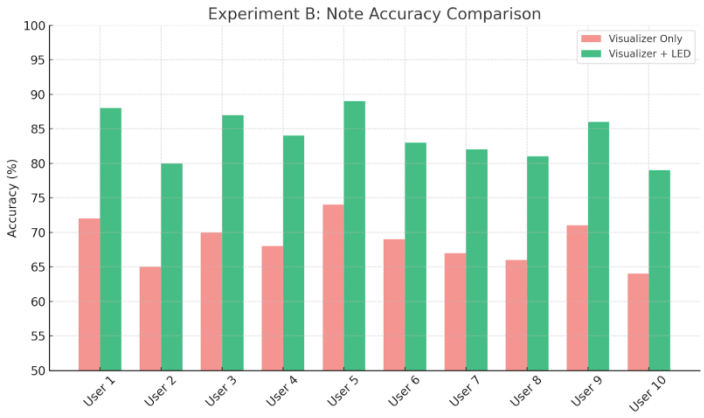

Figure 8. Figure of experiment 2

Every participant in the study demonstrated a measurable improvement when using the LED-guided version of MelodyDrop. The average increase in accuracy was 15.3 percentage points, with all users gaining between 14 and 17%. The group's baseline with visuals alone was 68.6%, which rose to 83.9% with LED assistance.

The improvement suggests that the LED system aids in real-time spatial recognition, helping users quickly locate the correct piano keys. Unlike falling notes, which require mapping from screen to keyboard, the LED strip removes that translation step entirely.

What stood out was the consistency in improvement across all participants — even those who were already relatively accurate. This implies the LED feedback benefits both newer and more experienced users.

We hypothesize that combining visual motion (falling notes) with physical light feedback builds faster reaction timing and stronger muscle memory. This validates the core goal of MelodyDrop: to improve learning retention by combining digital and physical feedback systems.

5. Related Work

One related approach involves using tactile feedback to teach piano skills by reducing the learner's dependence on visual cues [11]. The method presents timing information and key guidance through vibration or touch-based signals, allowing users to focus more on hand positioning and auditory feedback. This tactile training system was implemented on a mobile device and tested to assess its effectiveness in improving keystroke timing and reducing visual strain. While the system shows promise in shifting attention away from the screen, its main limitation is the lack of spatial feedback. Our project improves on this by using physical LED indicators above each key, offering direct spatial guidance that complements auditory and visual input.

Another method for simplifying piano learning involves using MusicXML-based visual guidance with hardware components like a Raspberry Pi, Arduino, and LED lights [12]. This system interprets MusicXML files and lights up the correct keys on a Casio keyboard in real-time, helping beginners follow along without needing to read sheet music. It also includes a playback mode that allows learners to hear the piece before attempting it themselves. While this method enhances accessibility and encourages auditory learning, it requires a wired setup with specific hardware and limited portability. Our project builds upon this by offering similar visual LED

guidance via BLE-connected ESP32 hardware, controlled wirelessly through a cross-platform Flutter app.

A more immersive approach to piano instruction incorporates a sensorimotor system combining MIDI, LED bars, gloves with pressure sensors, and haptic feedback [13]. This setup links a teacher's piano to a students via a controller that lights up the correct keys and transmits finger-specific feedback using vibration motors in wearable gloves. OLED displays also show notation directly on the gloves. Originally designed for individuals with cerebral palsy, this system supports learners with cognitive or spatial perception difficulties. While it offers detailed multi-sensory feedback, it requires specialized hardware and a live teacher. MelodyDrop achieves a similar effect with greater accessibility and autonomy, using only a smartphone and Bluetooth-connected LEDs.

6. CONCLUSIONS

While MelodyDrop successfully integrates MIDI visualization, BLE communication, and physical LED feedback, there are a few key limitations [14]. First, note synchronization can become slightly inconsistent at faster tempos or with dense MIDI files due to BLE bandwidth limits and Flutter timing jitter [15]. We could resolve this by implementing note batching and message compression on both app and ESP32 sides. Second, the app currently supports only basic MIDI parsing and does not yet interpret advanced controls like pedal events or velocity. Future updates could expand parsing capabilities and allow dynamic expression playback. Additionally, the UI has not been extensively tested for screen readers or vision-impaired users, so accessibility support is an area for growth. Finally, a practice feedback system could enhance user retention by analyzing missed notes and offering replay recommendations. With more development time, we would also add cloud sync, MIDI recording, and community-shared song libraries.

MelodyDrop bridges the gap between digital and physical learning tools by combining visual MIDI playback with real-time LED guidance. The system offers a highly accessible platform for beginners and enhances learning through multi-sensory reinforcement. It is an affordable and effective way to make piano practice more intuitive and engaging.

REFERENCES

[1] Rogers, Katja, et al. "Piano: Faster piano learning with interactive projection." Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces. 2014.

[2] Huang, Tongbo, et al. "MidiFind: fast and effective similarity searching in large MIDI databases." Proceedings of the 10th International Symposium on Computer Music Multidisciplinary Research. sn, 2013.

[3] Jiang, Zheng, and Roger B. Dannenberg. "Melody identification in standard MIDI files." 16th Sound & Music Computing Conference. 2019.

[4] Ferdoush, Sheikh, and Xinrong Li. "Wireless sensor network system design using Raspberry Pi and Arduino for environmental monitoring applications." Procedia Computer Science 34 (2014): 103-110.

[5] Meschtscherjakov, Alexander, et al. "ChaseLight: ambient LED stripes to control driving speed." proceedings of the 7th international conference on automotive user interfaces and interactive vehicular applications. 2015.

[6] Bowen, Judy, and Steve Reeves. "UI-design driven model-based testing." Innovations in Systems and Software Engineering 9.3 (2013): 201-215.

[7] Dian, F. John, Amirhossein Yousefi, and Kasun Somaratne. "A study in accuracy of time synchronization of BLE devices using connection-based event." 2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). IEEE, 2017.

[8] Ma, Long, et al. "A fast LED calibration method under near field lighting based on photometric stereo." Optics and Lasers in Engineering 147 (2021): 106749.
[9] Guan, Xinping, et al. "A comprehensive overview of cyber-physical systems: From perspective of feedback system." IEEE/CAA Journal of Automatica Sinica 3.1 (2016): 1-14.
[10] Tashildar, Aakanksha, et al. "Application development using flutter." International Research Journal of Modernization in Engineering Technology and Science 2.8 (2020): 1262-1266.
[11] Tsutsumi, Hokuto, Hiroaki Nishino, and Tsuneo Kagawa. "A piano performance trainer with tactile guidance." 2017 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW). IEEE, 2017.
[12] Ahlzén, Anton, et al. "Piano Hero: Interactive musical learning." (2024).
[13] Blumenstein, Tobias, et al. "Sensorimotor piano system for people with disabilities." Journal of Sensors 2016.1 (2016): 9825905.
[14] Chaari Fourati, Lamia, and Sana Said. "Remote health monitoring systems based on bluetooth low energy (BLE) communication systems." International Conference on Smart Homes and Health Telematics. Cham: Springer International Publishing, 2020.
[15] Schoentgen, Jean, and Philipp Aichinger. "Analysis and Synthesis of Vocal Flutter and Vocal Jitter." INTERSPEECH. 2019.