ChatGPT

Designing a Web-Based Musical Accompanist Tool

Learning an instrument like the viola is challenging, especially when practicing pieces that normally have accompaniment (piano, string quartet, etc.) or when working on solo repertoire without harmonic context. A web-based **“accompanist” tool** can provide that missing context – offering reference pitches, rhythm, and simple harmonies to play along with. This not only benefits viola practice but can serve **any musician** (singers, other instrumentalists) looking to improve timing, intonation, and overall musicality by simulating a band or ensemble backing. Below, we outline the scope, features, and design considerations for such a tool, followed by a detailed prompt for building it.

Scope of Accompaniment and Use Cases

The primary focus is to assist classical instrumental practice (e.g. viola pieces or etudes) with a **skeleton accompaniment**, but the tool should be flexible for broader musical use: - **Classical Repertoire Practice:** Provide harmonic and rhythmic backing for solo pieces (e.g. sonatas with piano, concertos, unaccompanied suites). This helps the player **keep tempo and pitch** by hearing the implied harmony as they play. - **Technical Exercises:** Aid in scale and arpeggio practice by making them more musical. For example, playing scales against chord progressions or drones to develop better intonation and key sense. - **General Musicianship:** Even outside classical music, the tool could act as “training wheels” for songs. For instance, a group of friends singing *“Happy Birthday”* could stay in tune better with a gentle chordal guide in the background. Having some accompaniment turns a monotone metronome exercise into *real music*, which keeps practice engaging [1] . - **Solo Performances without Accompanist:** Musicians who lack access to a pianist or ensemble can use the tool to perform pieces with a basic backing track. (The user even explored using OCR software like PlayScore to turn sheet music into MIDI and play it on a robotic music box, just to avoid needing a live pianist [2] – highlighting the demand for a convenient accompanist substitute.)

Importantly, this tool is **not limited to viola** – while we’ll tailor features to string players (e.g. intonation needs), the design will be useful for any instrument or even vocalists. In fact, practicing with accompaniment teaches you to stay in tune with other parts and maintain tonal center, a crucial skill for singers and ensemble players [3] .

Input Format and Content Creation

One challenge is how to **feed the music** into the tool. Ideally, a user could upload a score and have the app automatically generate accompaniment, but current technology makes fully automatic analysis hit-or-miss. Optical Music Recognition (OMR) tools like Audiveris or PlayScore exist, but they require clear notation and often manual correction. For example, MuseScore’s PDF import (powered by Audiveris) often needs ~15 minutes of cleanup per page [4] . PlayScore is a more advanced commercial OMR that can output MIDI [2] , but relying on it every time is cumbersome and may still produce an overly detailed accompaniment when a simpler outline is desired.

Planned approach: Start with **manual or guided input** rather than raw score uploads. This gives users control over the accompaniment's simplicity. Possible input methods:

- **Chord Progression Entry:** The user can enter a sequence of chords or harmonic markers (e.g. "C – G – Am – F…" with timing info). The app will then play those chords in time. This is similar to how apps like iReal Pro allow typing in chord charts to generate a backing track [5] [6]. A simple text format or chord chart UI can be provided.
- **Key & Scale Selection:** For scales/arpeggios, the user might just select the key (and mode) and a pattern type (e.g. "C major scale" or "G minor arpeggio"). The tool can then automatically produce a preset accompaniment (like a tonic drone or a I–IV–V chord loop) appropriate for that exercise.
- **Pre-Loaded Templates:** Common practice routines for string players can be built-in. For instance, a **Carl Flesch scale system** template (covering major/minor scales in 3 octaves, arpeggios, broken thirds, etc.) could be included. The user picks the scale/arpeggio they want, and the app already knows a simple accompaniment for it. This lowers the setup time for everyday fundamentals.

By doing the harmonic analysis **externally (by the user or via prepared content)**, we avoid the pitfalls of imperfect AI analysis. In the future, if reliable AI generation improves or if an API is affordable, we could add an "auto-generate accompaniment from audio/score" feature. But initially, **user-defined chords/harmony** ensures the output is correct and musically sensible.

Desired Features and Functionality

To make this accompanist tool effective for practice, it should incorporate several key features:

- **Rhythmic Support (Metronome & Tempo):** The tool should keep strict time like a metronome, but in a more musical way. Users can set a **tempo** (in BPM) and the accompaniment will play in time. Many musicians find playing with real backing tracks more engaging than a click [1]. We can still offer an optional metronome click (perhaps as a percussion or hi-hat sound) or accent the downbeats clearly. A **count-in** feature is essential: when the user hits play, the app can give a one or two bar count-in (e.g. audible counts "1-2-3-4") before the music starts, allowing the player to ready their instrument.
- **Harmonic Context (Chordal Accompaniment):** The core of the tool is providing harmonic reference. Unlike full-scale orchestrations, we want a **stripped-down accompaniment** – for example, sustained chords, simple arpeggios, or bass lines that outline the harmony without distracting. This could mean a **"skeleton" piano accompaniment** playing whole-note chords, or a drone of the tonic and fifth for simpler context. The idea is to give enough pitch reference to guide intonation and musical understanding, while leaving space for the soloist. (Imagine practicing a Bach cello suite while a quiet organ drone on the tonic plays, or a Kreutzer etude with a piano playing the I and V chords underneath each phrase.)
- **Adjustable Volume and Instrument Sound:** Users should be able to adjust the backing volume so it doesn't cover their playing. The **instrument voice** used for accompaniment can be simple – a piano is versatile and familiar, but we might offer a few options (e.g. string pad, organ, or just synthesized pure tones). A *"reference drone"* mode could use a sustaining string/cello sound (as some tuning drones do [7]) to provide a rich resonance. The key is a **clear, in-tune sound** that blends well (perhaps avoiding very complex timbres).
- **Intonation and Tuning Adjustments:** This feature sets our tool apart from basic backing tracks. We want to incorporate the ability to toggle between **equal temperament and just intonation** for the

accompaniment. In real ensembles, players constantly adjust pitch to achieve pure-sounding intervals, and training the ear for this is invaluable [8]. The Yamaha Harmony Director device is a model example: it lets music educators demonstrate how chords *ring* when tuned justly vs. equally [9]. Our tool can similarly offer, say, a "Just Intonation mode" in a given key. For example, if the accompaniment is set to F major and just intonation, the tool would subtly adjust certain scale degrees (the major third A is lowered about 14 cents, the fifth C is raised ~2 cents, etc.) to make pure F major chords [10]. These small adjustments mirror what string quartets or choirs do naturally and can train the user's ear to place their notes accordingly. (If a violist plays an F major scale with our drone, they can hear whether their A matches the pure third or if it's too sharp against the drone.) We might integrate a library or logic for microtonal tuning – for instance, the Tune.js library can provide frequencies for just scales easily [11] [12]. Of course, equal temperament (the standard 12-tone tuning) will remain the default, since practicing scales in equal temperament is also important [13]. Advanced users may even explore custom temperaments or historical tunings using this feature.

- **Real-Time Interaction (Controls and Looping):** At minimum, the interface will have **Play/Stop controls** to start or halt the accompaniment. Stopping should be smooth (maybe at the end of a bar, if musically sensible, unless it's an emergency stop). **Pausing** and **looping** are very useful for practice: for example, the user might loop a 4-bar tricky passage repeatedly. The tool can allow selecting a section of the progression to repeat, or simply restarting easily from the beginning or a marked point. We should also include a way to **rewind or jump** to specific bars (especially for longer pieces, so you don't always start over from bar 1 after a break). Visual feedback like a highlighted **chord chart or measure count** during playback is helpful for orientation [14]. This way if a player loses their place, they can glance at the screen to sync back up.
- **Gradual Tempo Changes:** A helpful practice feature (seen in tools like Strum Machine) is an **auto-speed-up** option [15] – e.g. the accompaniment could start at a slow tempo and increase by a few BPM each repeat, building the player's proficiency. Or conversely, a **ritardando/accelerando** feature could allow intentional tempo changes for musicality (though implementing that requires careful timing controls).
- **Transposition:** The tool should allow changing key easily. If the user inputs a chord progression in C major but later wants to practice the piece in C♯, they should be able to transpose the accompaniment up a semitone (or any interval). This is useful for accommodating different tuning standards (A=440 vs A=442 Hz, etc., which could be a separate master tuning control) or for practicing in comfortable keys. Since the accompaniment is algorithmic, transposing is straightforward – just shift all notes by the required interval.
- **AI-Generated Harmonies (Future Expansion):** In the long run, we can explore more automation. For instance, given a **solo melody line**, an AI could try to suggest chord changes or even generate a simple piano accompaniment. Research in automatic accompaniment generation has produced some results (e.g. AI models that can harmonize melodies), and tools like Band-in-a-Box use algorithmic rules to create arrangements. However, these are complex to implement from scratch and can require significant computing resources or datasets. The **Cadenza Live Accompanist** app, for example, uses sophisticated AI trained specifically on classical pieces to follow and accompany a soloist in real-time [16]. That level of nuance is beyond our scope for now (their system was in development for decades). Initially, we will **not rely on heavy AI** – both to keep costs down (no expensive API calls) and to ensure reliability. But designing our data structures now (chord progressions, timing) could lay the groundwork for plugging in an AI module later. For example, one day the user might upload an audio of them humming a tune, and an AI suggests a chord

progression; the app could then play that progression. This is aspirational, but worth keeping in mind.

Pre-Configured Practice Modules (Scales & Exercises)

To make the tool immediately beneficial, we will include some **ready-made accompaniment patterns** for common exercises: - **Scale Drone:** For any major or minor scale, the app can provide a continuous drone on the tonic (and maybe fifth). This is a classic intonation exercise – sustaining the tonic pitch helps the player center each scale note in tune [17] [7]. The user selects the key and the drone plays; the user then practices the scale or arpeggio against it. We could also offer the drone on scale degrees like the third or fifth to practice tuning those relative to the tonic. - **Scale Chord Progression:** A step up from a simple drone, we can make scale practice feel like real music by adding a chord progression. A popular approach is to pair the scale with common **I–vi–IV–V chords** in that key [18]. For example, in C major, the app could loop the progression C – Am – F – G while the user plays the C major scale up and down. This not only forces the player to hear each note in a harmonic context, but also teaches the sound of basic chord changes. Pianists often practice scales with a left-hand chord pattern [18]; our tool gives that experience to those practicing without a pianist. We can include a table of such chord progressions for all keys (as in the Pianote example) [19]. The user might even play arpeggios matching the chords, etc. - **Arpeggio Reference:** For arpeggios (like broken chords), the tool can simply play a solid chord or an outline of the chord tones in rhythm. E.g., if practicing a D minor arpeggio, the app could hold a D minor triad softly as a pad sound each measure, or play a low D and A as a bass support. - **Rhythm and Sight-Reading Exercises:** Besides pitch, we could incorporate some rhythmic patterns. For instance, the user could input a rhythm (or select a preset like “swing feel” or “march beat”) and the accompaniment could accentuate that. This crosses into the territory of a drum machine, which might be outside our main scope, but a simple **percussion backing** could be nice for styles like jazz or rock if we expand beyond classical.

All these pre-configured modules aim to **jump-start practice sessions**. A violist could open the app, select “Scale: D major, 3 octaves” and immediately get a drone or chord track to play along with, without manual data entry. This lowers the barrier to productive practice. It also reinforces broader musicianship; for example, hearing the **chord progression while playing a scale trains the ear in harmony and improves overall musical understanding** [18].

Real-Time Playback and User Interaction

Because this is a real-time accompaniment, performance and usability are crucial: - The app will generate **live audio playback** in the browser. This means using Web Audio API or a library on the client side – no need for server-side audio streaming (ensuring low latency). - We must ensure timing is accurate. JavaScript in browsers can have timing jitter if not handled carefully, so using the Web Audio scheduler or a library like Tone.js (which abstracts scheduling) is wise. Tone.js would allow us to schedule chord events, metronome clicks, etc., precisely on beat, and also offers built-in support for synths and effects. - The UI should be clean and responsive. Likely we’ll build a simple single-page application. There can be a **tempo slider or input**, start/stop buttons, a readout of the current bar/beat, and perhaps a simple “timeline” showing the structure of the accompaniment (like markers for sections or repeats). - **Accessibility:** It’s worth noting that some users (e.g. educators in classrooms) might want to use this on various devices. Keeping it web-based (no installation required) means a teacher can pull it up on an iPad in a lesson or a student on their laptop at home. We should allow keyboard shortcuts too (e.g. spacebar to start/stop, arrow keys to transpose or move sections) for convenience. *(For a violist who might be holding the instrument, using a foot pedal via* *keyboard or a bluetooth page-turner pedal to control playback could be handy – possibly an extra feature to consider.)*

Additionally, since practicing music is often an iterative process, the tool should make it easy to **save and recall settings**. If a user enters a chord sequence for a particular piece, they should be able to save that (perhaps in local storage or a small cloud DB if accounts are used) for next time. We could offer a list of “recent songs” or “favorites” they can quickly load, much like how iReal Pro provides a library of songs [20] [21].

Tech Stack and Implementation Notes

Front-end: A web application built with standard technologies (HTML5, CSS, JavaScript). We can use a front-end framework like **React** or Vue for ease of building the UI components and managing state (especially if we implement looping, setlists, etc.). However, for a simpler initial prototype, even plain JS with perhaps a UI library could suffice. The critical part is the audio engine.

Audio Engine: The **Web Audio API** is powerful enough to generate tones and schedule events. We can either use it directly or leverage libraries: - *Tone.js:* A high-level library that simplifies scheduling musical events and includes synth instruments and effects. Tone.js would allow us to create a Transport (global clock), schedule chord “events” on certain beats, and even load sound samples or synthesize tones easily. For example, we could load a piano SoundFont or use a simple FM synth for a tone. - *Tune.js:* As mentioned, for tuning adjustments, Tune.js can be used to get frequencies for just intonation or other temperaments [11]. Alternatively, we can manually calculate the few cent offsets for pure intervals (since we mostly need pure 3rds/5ths). Using Tune.js might be overkill if we’re only toggling equal vs pure in a major/minor key, but it’s a nice library if we extend to exotic scales. - *MIDI playback:* Another approach is using MIDI sounds or SoundFonts. We could have a built-in SoundFont for piano or strings and use a library to play MIDI notes through it. There’s **soundfont-player** or **FluidSynth Web** (WebAssembly) that could render MIDI to audio. However, since our accompaniment is generated on the fly (not just playing a pre-made MIDI file), using Tone.js or Web Audio directly might be simpler. - *Real instrument samples vs synthesized:* Tools like Strum Machine stitch real instrument recordings for authentic sound [22]. For our purposes, synthesized sound might be acceptable (and gives us easy control over sustain and tuning). A simple **sine wave or saw wave** could even be used for drones (ensuring no inherent beating). A piano sample could make it more pleasant, though, especially if users practice melodic pieces (piano sound is familiar for classical accompanists). We might include a lightweight piano sample set or even use the Web Audio API to mimic a piano (via oscillators and filters) – or again, let Tone.js’s built-in instruments handle it.

Back-end: Initially, **no heavy back-end** is needed. All processing can be client-side, which is great for latency and offline use. We might have a back-end if we introduce user accounts or want to store user-created accompaniment presets on a server. But a first version could store data locally in the browser (LocalStorage or IndexedDB). This also avoids any server costs – aligning with the user’s budget concerns.

Device Compatibility: Ensure it works on modern browsers (Chrome, Firefox, Safari) and on various devices. Latency on mobile browsers can be a bit higher, but since this isn’t interactive like a live instrument (it’s one-way playback), small latency differences won’t be critical. We do want the timing to be steady once started.

User Interface Considerations: Keep it simple and focused on practice: - The main screen could have a **tempo control, play/stop button, and perhaps a textual chord display** scrolling or paging as it plays. - A menu or sidebar for selecting presets (scales, patterns) or entering custom chords. - A toggle for "just intonation mode" when a key is selected (only enabled if the accompaniment knows what key it's in – which it will if you choose a scale or if you specify a key with your chords). - Possibly an option for **"guide melody"**: In some cases, the user might want the tool to also play the solo melody very quietly as a reference (especially if they are first learning the piece). This could be a midi melody line loaded by the user or a simplified version of it. However, implementing melody playback requires inputting that melody, which goes back to the score input problem. We mention this as a possible feature, but it may be low priority since the primary use is the human playing the melody.

Similar Tools & Inspiration: We draw inspiration from several existing tools: - *MuseScore/Finale:* Manual input of full accompaniment is too slow for everyday use, but shows that once you have a MIDI, you can play it. Our tool streamlines the input process by focusing on chords and patterns. - *iReal Pro:* Demonstrates how chord charts can generate a band accompaniment in any key/tempo/style [5]. We similarly allow chord chart input, though our default "style" will be a simple classical drone/piano, rather than jazz trio or rock band (future style additions are possible). - *Strum Machine:* Emphasizes simple UI, real-time chord highlighting, and the benefit of practicing with accompaniment to improve timing and confidence [23] [14]. We aim for that user-friendly experience in a browser, without expensive or cheesy-sounding hardware. - *Yamaha Harmony Director:* In hardware form, it shows the value of switching temperaments and playing sustained chords for ear training [9] [10]. We are effectively creating a software analog of some Harmony Director functions, tailored for individual practice. - *Cadenza Live Accompanist:* An advanced AI accompanist that actually follows the performer. Our tool will *not* attempt to follow in real-time (that requires complex audio analysis and score following). Instead, **our user will follow the predetermined accompaniment**, as if playing with a strict backing track. This is still extremely useful for practice. (In the future, if we integrate a microphone and some tempo detection, we could try minor "auto-pause if soloist falls behind" features, but that's beyond initial scope.)

With these features and considerations in mind, the next step is to implement the application. Below is a comprehensive prompt intended for an AI development assistant (Claude) to generate the first version of this web accompanist tool.

Prompt for Claude Sonnet 4

```
You are Claude Sonnet 4, an AI developer assistant tasked with building a web-
based musical accompanist tool for practice and learning. Please help generate
the code or project structure for this application. Here are the requirements
and guidelines:

**1. Overall Goal:** Create a **browser-based app** that plays a simple musical
accompaniment (chords, drones, or basic backing tracks) for a user who is
practicing an instrument or singing. The app should be lightweight, responsive,
and work offline if possible (after initial load).

**2. Core Features to Implement:**
- **Tempo Control & Metronome:** Allow the user to set a tempo (in BPM). The app
```

```
should play in-time according to this tempo. Include an optional metronome click
or an accent on downbeats. Provide a one- or two-bar **count-in** when playback
starts (e.g., a count voice or clicks).
- **Chord Accompaniment Playback:** Based on user input (see Input section
below), generate a sequence of chords or pitches and play them in rhythm. Chords
can be sustained or played in a simple rhythmic pattern (e.g., whole notes, half
notes). Ensure smooth audio playback timing (use Web Audio API scheduling or a
library like Tone.js).
- **Input of Chords/Progressions:** Create a simple way for the user to input
the chords or select a preset:
  - Users can type in chords with a textual format (for example: `C F G7 C` for
a simple progression, with spaces or commas separating measures). You can define
a basic syntax (e.g., chords separated by spaces, measures separated by `|` or
newline, etc.). Parse this input into a chord sequence.
  - Alternatively or additionally, provide UI controls for common patterns
(dropdowns for key and progression type, buttons for “Major scale drone”, “I-IV-
V loop”, etc.).
  - For now, focus on parsing basic chord names (like C, G7, Dm). Assume chords
are in root position and use equal temperament tuning by default (we’ll adjust
tuning in a later step).
- **Sound Generation:** Use the Web Audio API to generate sound. A piano or pad
sound is preferred for chords:
  - You can use an oscillator for simple wave (sine wave for pure tone drone) or
use a simple synthesized piano. If using Tone.js, you might use `Tone.Synth`
with a suitable waveform or `Tone.Sampler` with a small piano sample.
  - Ensure chords (multiple notes) can be played – e.g., by creating multiple
oscillators or voices, one per chord note.
  - Include a **volume control** for the output so the user can adjust
accompaniment loudness.
- **Playback Controls:** Implement Start and Stop (and Pause if possible)
controls. Start should initiate the count-in then accompaniment. Stop should
halt playback cleanly. Also provide a way to **loop** playback if a loop section
is selected (for example, always loop the entered progression). This can be a
toggle for “loop mode”.
- **Visual Feedback:** Display the chord names or progression in sync with
playback (highlight the current chord or bar). Even a simple text highlight or
underline of the current chord is fine. This helps the user follow along.
- **Preset Library:** Include a few hard-coded presets that users can load with
one click:
  - Example: “Drone: A (440Hz)” – which simply plays a continuous A pitch (good
for A major or minor scale practice).
  - Example: “G Major Scale – I IV V chords” – which preloads the chords G – C –
D – G (the I, IV, V, I in G major) and maybe sets the tempo to a moderate value.
  - Example: “C Major I-vi-IV-V” – which loads C – Am – F – G.
  - Feel free to include 3-5 such examples.
- **Tuning Mode (Just Intonation vs Equal):** Implement a mode to adjust tuning
of the chords:
  - Provide a toggle or select for “Just Intonation (Major/Minor)” vs “Equal
```

```
Temperament”.
  - For Just Intonation: When this mode is on and a key is known (e.g., the
first chord or a selected key), adjust the major third of any major chord to be
14 cents flat and the fifth to be ~2 cents sharp (relative to equal). For minor
chords, you might adjust the minor third slightly (16 cents sharp if pure minor,
since a pure minor third is 6:5 ratio).
  - Simpler: you can demonstrate just intonation for the tonic chord of the
selected key. For instance, if the key is D major and JI mode is on, ensure the
F# in a D major chord is tuned to pure (so its frequency = D * 5/4) rather than
equal-tempered.
  - Implementation: You can calculate frequencies manually. Equal-tempered
semitone ratio is 2^(1/12). Just intonation ratios for reference: major third =
5/4 of root, perfect fifth = 3/2 of root, minor third = 6/5 of root. Use A4 =
440 Hz as a base reference.
  - If using Tone.js, you might set oscillator frequencies directly or use the
`detune` property to adjust cents for specific notes.
- **Responsive Design:** Make the UI layout flexible so it works on desktop or
tablet/mobile. It can be simple (column layout with controls), but ensure
buttons are tappable on touch screens.

**3. Development Approach:**
- Use **HTML/CSS/JavaScript**. You can scaffold a basic HTML page with a
`<canvas>` or `<div>` for visual feedback, and use JavaScript for the logic. If
using external libraries (Tone.js, etc.), you can include them via CDN.
- Write clear functions or classes for:
  - Parsing chord input (e.g., a function `parseChords(inputText)` that returns
an array of chord objects).
  - Scheduling playback (e.g., a function to schedule the next chord or a loop
using `setTimeout`/`setInterval` or Tone.Transport).
  - Handling tuning conversion (perhaps a helper that given a chord and mode
returns frequencies for each note).
- Provide comments in the code explaining how each part works, for
maintainability.

**4. Example Scenario to Test:**
- User opens app, selects a preset “D major scale drone”. Tempo maybe is
irrelevant for a continuous drone, but let’s say 60 BPM. The app plays a
sustained D note (perhaps add A as well for a fifth) indefinitely. User stops it
after practicing.
- User enters custom chords: `"D / A7 / G / D"` (slashes or newlines indicating
bar separation). Sets tempo 100 BPM. Turns on Just Intonation mode in D major.
Hits play. The app counts in and then plays a bar of D major chord, a bar of A7,
a bar of G, a bar of D, looping back. The D and G chords use pure tuning (F# in
D chord adjusted down, B in G chord adjusted, etc.), the A7 (A dominant 7) can
be treated as A major for tuning its third (C#) pure.
- User can hear the difference if they toggle JI mode off (everything goes equal
temperament).
- They pause or stop as needed.
```

```
**5. Non-Functional Considerations:**
- Avoid lag or crackles in audio: use appropriate scheduling (e.g., schedule
sounds slightly ahead of time).
- Provide basic error handling for chord input (unknown chord names, etc.);
maybe default to a safe value or alert the user.
- No server-side code needed; all can run in-browser. Ensure the solution is
self-contained.

Please produce the necessary HTML/CSS/JS (and any explanatory markdown if
needed) for this application. Aim for clarity and correctness, even if not every
edge case is covered.**
```

---

[1] [18] [19] How to Practice Piano Scales: 3 Tips | Pianote
https://www.pianote.com/blog/how-to-practice-piano-scales/

[2] Re: [CUSTOMER] Re: [CUSTOMER] Thank You for Your Interest in the Muro Box – Need Any Assistance?
https://mail.google.com/mail/u/0/

[3] voice - Do singers need to practice singing without accompaniment? - Music: Practice & Theory Stack Exchange
https://music.stackexchange.com/questions/91670/do-singers-need-to-practice-singing-without-accompaniment

[4] Audiveris - MuseScore
https://musescore.org/en/node/308411

[5] [6] [20] iReal Pro - Practice Made Perfect
https://www.irealpro.com/

[7] Natural Cello Reference Tones | Online Tuning Tools For String Musicians | Online Metronome Tool
http://www.dronetonetool.com/

[8] [9] [10] [13] The Harmony Director: A Great Tool for the Music Classroom - Yamaha Music
https://hub.yamaha.com/music-educators/instruments/winds-instruments/the-harmony-director-a-great-tool/

[11] [12] GitHub - abbernie/tune: A web audio tuning library of microtonal and just intonation scales
https://github.com/abbernie/tune

[14] [15] [21] [22] [23] Strum Machine: customizable backing tracks made from real instruments
https://strummachine.com/

[16] Cadenza™ Live Accompanist: The Orchestra that Follows You
https://metamusic.ai/

[17] Teaching Intonation Part 5: Using Drones
https://happyteachinglife.com/teaching-intonation-part-5-using-drones/