ChatGPT

Problem Definition (Technical Scope)

Audio–Visual Guitar Transcription refers to building a system that converts a guitarist's performance into **sheet music and tablature** by analyzing both audio and video. This is a multifaceted problem that spans several sub-problems:

- **Polyphonic Pitch Transcription:** Capturing *all* notes being played at any given time from the audio. Guitar music is often polyphonic (multiple notes at once), so the system must perform *multi-pitch estimation* to identify simultaneous pitches. This includes detecting fundamental frequencies and distinguishing them from overtones or noise.

- **Onset/Offset Detection:** Determining *when* each note starts (onset) and ends (offset) in time. Accurate onset detection is crucial for rhythm and notation timing. Offset detection defines note duration, which is needed for proper sheet music representation. The challenge is to detect onsets in complex audio (e.g. notes that begin during others' sustain) and offsets that may be blurred by reverberation or distortion.

- **Chord Extraction:** Identifying chords (groups of notes played together) from the audio. This involves mapping combinations of pitches to chord labels (e.g. "G major") and is complicated by the guitar's tuning and voicings. Chord extraction must account for *different inversions and fingerings* that yield the same set of pitches.

- **Fretboard Localization & Geometry:** In video, locating the guitar's fretboard in the scene and understanding its geometric orientation. The system must identify the region of interest (the neck of the guitar) and possibly correct for perspective (via a homography or 3D projection) so that frets and strings can be mapped consistently. Many solutions assume the entire fretboard is visible and use geometric constraints (e.g. the 12th-root-of-2 rule for fret spacing) to enforce a realistic fretboard shape.

- **Finger Detection and Mapping to String/Fret:** Tracking the musician's fingers on the fretboard via computer vision. This entails detecting **left-hand fingertips** and determining which string and fret each finger is pressing. Key challenges include distinguishing fingers (index, middle, etc.) and mapping their 2D image coordinates to specific guitar strings/frets, especially under occlusions or poor lighting. Modern approaches use hand pose estimation (keypoint detection of finger joints) to get fingertip positions, then project those onto the fretboard model to infer the pressed fret.

- **Disambiguating Multiple Possible Fingerings:** A given pitch or chord can often be played in multiple places on a guitar's fretboard (pitch redundancy). For example, the note *C4* can be played on the 5th string 3rd fret or 6th string 8th fret, etc. The system must decide which *string-fret combination* was actually used by the performer. This disambiguation can leverage visual info (which string the finger is on) and playability constraints (guitarists favor certain positions for ease). Vision is extremely helpful here: incorporating video allowed early multimodal systems to deterministically resolve ~89% of note ambiguities by seeing the hand position.

- **Tablature (Tab) Generation:** Constructing guitar tablature from the transcribed notes. Tablature is a guitar-specific notation with six lines representing strings and numbers for frets. The system must output a sequence of fret numbers aligned in time, per string. This involves formatting the detected pitches into the tab layout, assigning each note to a string and fret, and possibly grouping simultaneous notes into vertical stacks (for chords) in the tab.

- **Symbolic Music Representation:** Creating a structured, machine-readable representation of the transcription, such as **MIDI**, **MusicXML**, or **GuitarPro** format. This symbolic representation includes note pitches, timings, and optionally fingering (string/fret) annotations. It forms the basis for rendering standard sheet music and for further processing like editing or analysis.

- **Music Engraving:** Finally, producing human-readable sheet music and tab. This step uses the symbolic representation to layout notation on staves (for standard notation) and tab lines, with proper rhythmic notation, bar lines, etc. The engraving process must handle formatting details (e.g. stems, beams, fingering notation, bends, slides marks) to yield a polished score. Tools like LilyPond or MuseScore's engraving engine can be employed here to ensure professional-quality output.

- **Audio-Video Alignment:** Ensuring that the audio and video inputs are synchronized, so that events detected in audio (notes) correspond to the correct video frames. In many cases, the audio and video come from the same recording (thus inherently synced), but explicit alignment may be needed if there is drift or if combining separate sources. For multimodal processing, aligning onset times with video frame indices is essential for fusing information (e.g. knowing which video frame shows the finger when a given note sounds). Alignment can involve finding a constant offset or using cross-correlation between audio features and motion if necessary.

In summary, the system must *hear* the pitches and timing (audio domain) and *see* the finger placements (visual domain) to produce a transcription. It addresses ambiguities that audio-only or video-only methods face by combining both modalities. The ultimate goal is a **real-time or offline system** that takes a guitar performance and outputs a fully notated score with tablature, correctly indicating *what notes were played, when they occurred, and how they were played on the guitar* (which string/fret, and possibly which techniques like bend or slide).

Academic Research & Scientific Literature

2.1 Audio-Only Guitar Transcription

Early work on Automatic Music Transcription (AMT) dealt largely with **audio signals** alone. The classical problem is challenging, especially for polyphonic instruments. In audio-only guitar transcription, research has built upon general polyphonic AMT advances while addressing guitar-specific issues:

- **Polyphonic AMT Foundations:** Classical approaches included signal processing methods like spectral peak picking, filterbanks, and matrix factorization to estimate multiple pitches. For example, Barbancho et al. (2009) identified pitch candidates by spectral peak analysis and then inferred the likely string for each note by analyzing overtone inharmonicity. Benetos et al. (2013) and others introduced **non-negative matrix factorization (NMF)** variants to separate polyphonic spectra into note activations. These pre-deep-learning methods struggled with guitar timbres and overlapping harmonics. A seminal deep-learning approach for piano, **Onsets and Frames** (Hawthorne et al. 2018), introduced a dual-objective model predicting *onset events* and

frame-wise pitches with a CNN+RNN architecture. Onsets and Frames achieved state-of-the-art on piano by explicitly handling note onsets and sustaining frames, which inspired similar strategies for guitar.

- **Guitar-Focused AMT Models:** Guitar presents *pitch redundancy* (same pitch, different strings) and unique articulations, so specialized models emerged. **TabCNN** (Wiggins & Kim 2019) was a convolutional neural network designed to map guitar audio directly to a tablature grid output. It treated transcription as an image-like multi-label classification per time frame (six channels for six strings) and trained on GuitarSet data. TabCNN provided a direct audio→tab approach, combining pitch detection and string assignment in one model. Another model by Chen et al. (ICASSP 2022) used a **multi-loss Transformer** for polyphonic electric guitar transcription. It introduced a dataset (EGDB, see Datasets section) and trained a Transformer to predict a piano-roll representation with multiple loss functions (onset loss, pitch loss, etc.). Kim et al. (EUSIPCO 2022) proposed an *attention-based CNN* for note-level guitar AMT, incorporating an attention mechanism to focus on note events. These guitar-specific models often fine-tune architectures from piano transcription to handle guitar timbre and add a post-processing step for string assignment if they output just pitches.

- **Tablature-Specific Modeling:** Beyond predicting notes, some works explicitly output tablature. Early methods (pre-deep learning) used a two-stage approach: first transcribe pitches, then solve string assignment as an optimization problem. Sayegh's 1989 *Optimum Path Paradigm* introduced a dynamic programming method to find a minimal "cost" fingering path for a given note sequence, penalizing large hand jumps. Later, Burlet & Fujinaga (2013) built a web-based system that combined an existing multipitch algorithm with an A *search to find feasible tabs. Deep learning now allows end-to-end tablature inference. Wiggins & Kim's TabCNN (2019) essentially merges both steps, while Fretting-Transformer (Hamberger et al. 2025) treats string/fret assignment as a* sequence-to-sequence translation *from a MIDI note sequence to a tablature sequence. The Fretting-Transformer uses an encoder-decoder Transformer (inspired by T5) with a unified vocabulary of note and fret tokens, and achieved state-of-art tab accuracies by learning from large symbolic tab datasets. Similarly, Edwards et al. (2024) introduced MIDI-to-Tab** as a masked language model that infers tablature from a given note sequence – this is useful for post-processing audio transcriptions into tabs.

- **Pitch Tracking Methods:** Within audio-only approaches, *pitch detection algorithms* play a role. Classical methods like YIN or pYIN (for monophonic pitch) are accurate for single notes but not directly applicable to chords. CREPE (2018) is a notable deep learning method that estimates pitch from short audio frames at a very fine resolution (continuous frequency). CREPE works *monophonically* but is highly accurate and has been used for detecting prominent pitch and even note onset times by looking at pitch changes. For polyphonic content, models instead produce a *piano-roll* matrix (time vs. pitch activation). Some recent approaches incorporate *self-attention* (Transformers) to segment instances of notes in a mixed signal, essentially performing *instance segmentation* in time-frequency space. Despite improvements, audio-only pitch transcription for guitar often faces errors with **note overlaps, harmonics, and distortion** (e.g., a heavily distorted guitar produces complex overtone spectra, confounding pitch models).

- **Limitations of Audio-Only Approaches:** Audio-based guitar transcription often mis-identifies the correct string/fingering because the audio *pitch* alone doesn't reveal position. Without visual cues or string-specific signals (like a hexaphonic pickup), an audio model may output correct pitches but an *arbitrary or default string assignment* (e.g. always choosing the lowest possible fret). Also, expressive techniques like bends or slides can fool audio models – a bending note changes pitch continuously, which might be tracked as multiple discrete notes or a wrong pitch if

not modeled. Data scarcity is another issue: high-quality labeled guitar audio is limited compared to piano (e.g. less than 3–5 hours of aligned guitar audio in datasets, versus hundreds of hours for piano). This scarcity has historically limited the accuracy of guitar AMT. Domain adaptation and synthetic data (like **SynthTab**) have been explored: Zang et al. (2024) generated synthetic guitar audio from thousands of GuitarPro tabs to pre-train models, but found that models pre-trained on purely synthetic data didn't outperform those trained on real data alone. In summary, audio-only transcription can achieve high note recognition rates, but struggles with finger placement ambiguity and capturing performance techniques, motivating the inclusion of video.

2.2 Video-Only Fingering/Chord Detection

Using *video* alone to infer guitar playing is an intriguing approach: the idea is to "read" a guitarist's fingers to know what notes/chords are being played, even without sound. Several research efforts and methods exist:

- **Fingertip Detection and Hand Pose:** A core task is detecting the guitarist's left-hand fingertips on the fretboard. Traditional computer vision methods applied edge detection or color markers: e.g., some experiments placed colored markers on finger tips or fret positions for easier tracking (marker-based approaches). Modern approaches prefer *markerless hand pose estimation*. Tools like **MediaPipe Hands** (by Google) can output 21 hand landmarks (finger joints) in real time. By identifying the tip of each finger and the position of the wrist, one can infer where each finger is touching the fretboard. In a guitar context, researchers adjust such models – for instance, Bae *et al.* (UIST 2025) used MediaPipe Hand Landmarks to get finger joint coordinates, then refined the fingertip positions relative to frets (adding a padding, since guitarists press just *behind* the fret). High-level pose models ensure each finger is consistently identified (index vs middle, etc.), which is useful for mapping fingers to known hand biomechanics (like which finger tends to play which string in a given position).

- **Fretboard Tracking:** Before identifying fingers, the system needs to know *where the fretboard is* in the video frame. Paleari *et al.* (2008) developed a module to automatically detect the fretboard by finding the guitar's neck and frets in the first video frame. They looked for long, parallel lines (the strings) and shorter perpendicular lines (fret wires) – applying Hough transforms or line detection on a high-contrast fretboard image. Many guitars have metallic fret wires and inlaid position markers (dots) which can be detected: Bae *et al.* (2025) avoided direct fret detection (frets are thin and often occluded) and instead detected the **position marker dots** on the fretboard using a circular Hough transform. From the known positions of markers (e.g., typically at frets 3,5,7,9,12, etc.), they interpolated all fret locations using the musical 12-TET ratio formula. Once the fretboard corners and fret lines are known in the image, a geometric transform can "rectify" the fretboard – essentially mapping pixel coordinates to a standardized guitar fret/string grid. Paleari's system enforced that detected fret spacing must obey the 12th-root-of-2 rule (each fret distance shrinking by that factor), improving robustness against spurious line detections. With a tracked fretboard, even if the guitar moves in the video (translation, rotation, some perspective changes), the system updates the mapping in each frame.

- **Chord and Fret Recognition from Images:** Given clear images of the left-hand on the fretboard, one can attempt to recognize **chords** directly by pattern. Early CV research included training classifiers on static chord hand shapes. For example, a system might detect which frets have fingers on them and match that pattern to a chord dictionary ("if fingers on 2nd fret of A&D

strings and 1st fret of G string, that's a D major chord"). Such approaches can work for common open chords, but are less general (they need a library of chord shapes). More robustly, one can treat it as an object recognition or keypoint classification problem: detect each fingertip and map it to a (string, fret) coordinate, yielding a set of pressed positions at time T. This set of positions can then be interpreted – if multiple simultaneous presses and the strumming hand (right-hand) moves, that suggests a chord strum. **Markerless methods** have successfully extracted tabs from video alone in constrained settings. For instance, *Mark Asmar* (Polytechnique Montréal, 2022) developed a system using an **RGB-D camera** (depth sensor) to track fingertip positions in 3D on the fretboard and produce tablature. Depth data helps resolve occlusions (finger covering another) by providing 3D coordinates of each fingertip.

- **Occlusion Handling:** A major difficulty in video analysis is that fingers often occlude each other or parts of the fretboard. A single camera front view might not see a fingertip if another finger or the hand blocks it. Solutions include using multiple cameras at different angles or using depth. Multi-camera setups (e.g. one camera looking at the fretboard from the player's perspective and another from the audience's perspective) can provide complementary views so that at least one camera sees each finger. The **Guitar-TECHS dataset** (Pedroza et al. 2024) even uses an *egocentric* head-mounted camera plus an exocentric front camera to capture performances from two angles [1] [2]. If only a single camera is available, algorithms rely on temporal continuity: even if a finger disappears briefly, its last known position can be propagated (tracked) until it reappears. Kalman filters or optical flow tracking can follow a finger's blob through short occlusions. Hand pose estimation models also help: even if fingertips aren't directly visible, the model might infer their position from partial evidence (e.g. seeing the knuckle positions). Nonetheless, fast finger movements (like a rapid solo) and extreme occlusions (hand covering all fingertips from view) remain open challenges for video-only transcription.

- **Silent Video Transcription:** Remarkably, research by Goldstein and Moses (2018) showed it's partially feasible to transcribe guitar from **silent video** alone. Their approach leveraged the physics that string vibrations cause subtle visual signals. Using high-speed camera footage or rolling shutter effects, one can detect string oscillation patterns and estimate their frequency (and thus pitch) – a method previously applied to *silent piano videos*. For guitar, they focused on **polyphonic note identification from video** by combining finger position detection with analysis of string motion in the footage (the string might blur or show interference patterns when vibrating). This is an advanced technique requiring good resolution and possibly high framerates, and works best for clear nylon strings (classical guitar) where motion is visible. While intriguing, silent-video transcription is not yet as reliable as audio-based methods; it underscores the complementary nature of audio and video modalities.

In summary, video-only guitar transcription research has evolved from simple image processing of fretboards to sophisticated hand tracking. Video methods excel at determining *where* on the fretboard a note is played (resolving the ambiguity of guitar fingering). However, they struggle to determine *when* exactly a note sounds or if a string is plucked without also having audio – for example, a finger might be on a fret but that note might not actually be plucked yet, or could be being held from a previous note. Thus, video alone can identify chord shapes and fret positions, but timing and distinguishing between sounding vs muted strings are difficult without the audio. This naturally leads to combining both sources.

2.3 Audio-Visual Fusion Research

Combining audio and video ("**multimodal fusion**") for guitar transcription has shown great promise in tackling the limitations of single modalities. Several research projects and prototypes have explored audio-visual (A/V) guitar transcription:

- **Early A/V Systems:** Paleari *et al.* (2008) pioneered an audio-visual guitar transcription system. They implemented separate modules for fretboard tracking, hand detection, and audio transcription, then fused the results to output tablature. In their fusion step, audio provided candidate notes (pitches, onsets) and video provided constraints on string positions. Essentially, whenever the audio transcription found a pitch that could come from multiple strings, the vision system's detected hand position would narrow it down to the one string the left hand was actually pressing. Their results were impressive for the time: 89% of ambiguous notes were disambiguated *deterministically* by the visual data, meaning the system could directly see which string was used. This established that even a relatively simple fusion (rule-based combination of audio note events with visual finger locations) hugely improves guitar transcription accuracy.

- **Joint Audio-Visual Models:** More recent research attempts to **learn** the fusion via machine learning. Instead of treating audio and video pipelines entirely separately, a joint model can take synchronized audio and video inputs and output a transcription. One approach is *late fusion*, where separate networks process audio and video, and their outputs are combined at a decision level (e.g., combining probability distributions over notes). Another approach is *mid-fusion*, where intermediate features (like an audio embedding and a video embedding) are concatenated and fed to subsequent layers. For example, vision-assisted chord recognition models have used CNNs on video frames (to detect chord fingering shapes) alongside audio CNNs on spectrograms, merging their features to predict chord labels – showing improved accuracy over audio-only, especially in noisy audio conditions. Transformers have also been extended to multimodal input: a *multimodal Transformer* might have one branch attend to audio spectrogram sequence and another to a sequence of video frame features, with cross-attention linking the two. This can learn temporal alignment (e.g., ensuring the video info is used at the correct time in the audio sequence).

- **Example – Bass Guitar A/V Framework:** Bae, Kwon, and Park (UIST 2025) presented an audio-visual system for **bass guitar** transcription and fingering estimation. In their framework, the audio side uses CREPE to get pitch curves and detect note onsets, while the video side detects which string/fret the bassist's fingers are on. They fuse this information to produce a sequence of notes with string assignments. Additionally, they trained a Bi-LSTM model that takes the detected sequence and computes an "optimal fingering" – essentially refining finger choices over time to minimize hand movement. This system not only transcribes (like which notes were played) but also suggests the *ideal fingering sequence*, merging visual evidence (actual finger used) with a learned model of ergonomic fingering. The inclusion of video was crucial to reliably identify which string was plucked for each note (their evaluation separately measured pitch recognition accuracy and finger recognition accuracy). The multimodal system could achieve over 81% accuracy in pitch detection and around 77% in finger (string) recognition on test songs, outperforming what either modality alone would do.

- **Late Fusion with Graphical Models:** Some approaches use probabilistic models to fuse modalities. For instance, one can create an HMM or factor graph where the hidden state is the combination of note + string, the audio gives likelihoods for notes, and the video gives likelihoods for strings. Then decoding the HMM finds the most likely sequence of note+string

that explains both streams. Such constraint-based fusion ensures that the final transcription is consistent with both audio (pitch correctness) and video (finger placement). In practice, implementing this requires careful synchronization and maybe assumption that each audio onset aligns with a discrete video frame state.

- **Multimodal Transformers and Sequence-to-Sequence:** A cutting-edge direction is to feed both modalities into one sequence model. For example, a sequence of audio feature frames and a sequence of video features can be interleaved or used as separate token types in a Transformer. The model could output a sequence of notes or a sequence of events (like "finger 3 pressed fret 5 on string 2 at time t"). Some initial work on audio-visual *alignment* uses cross-modal attention: the model learns to attend to the video frames that correspond to each note onset in the audio. This can implicitly perform alignment and fusion. Though research specifically on a single model transcribing guitar from raw audio+video is still emerging, analogous efforts exist in piano (aligning player's hand video with audio for transcription) and have shown that adding video reduces error rates in note detection in certain cases (for instance, distinguishing piano pedaling via key motion visible on video). For guitar, a future multimodal model might directly output tablature given spectrogram frames and, say, hand keypoint sequences as inputs.

- **Challenges in A/V Fusion:** Synchronization is a prerequisite – if audio and video are even slightly out of sync, the fusion model may mistake which note corresponds to which finger position. There's also the issue of **modality reliability**: if the audio is very noisy (e.g., background band or crowd noise), the system might need to rely more on video; if the video is poor (blurry or occluded), it should defer to audio. Some research explicitly models this, weighting the confidence from each modality. Another challenge is the *increased data requirement*: a model that uses both audio and video ideally needs training data that has both modalities aligned with ground truth transcriptions. Such datasets are scarcer (though a few exist, see Datasets). Despite these challenges, A/V fusion is compelling because it combines the precise timing and multi-note detection strength of audio with the disambiguation and technique-identification power of video.

In summary, audio-visual research has demonstrated that combining modalities yields superior transcription of guitar performances. By using vision, ambiguous audio notes can be resolved (which string/fret), and by using audio, vision's uncertainties (like whether a finger actually produced a sound) can be confirmed. Modern deep learning is just beginning to tap into joint A/V models, and early results in domains like bass transcription and technique recognition show significantly improved outcomes over single-modality systems. The fusion approach moves us closer to a system that *"sees" and* "hears"* like an expert human, leveraging all available clues to understand a guitar performance.

2.4 Datasets

A variety of datasets have been developed to support research in guitar transcription and technique analysis. These datasets provide aligned audio (and sometimes video) with annotations like notes, tabs, or technique labels. Below is a list of prominent datasets, including those up to 2025:

- **GuitarSet (2018)** – A reference dataset for guitar transcription. GuitarSet contains ~360 excerpts (~3 hours) of solo **acoustic guitar** performances recorded by 6 guitarists, covering a range of genres. Uniquely, it was recorded with a **hexaphonic pickup** (separate audio channel for each string) plus a mic, enabling precise note annotation per string. It provides time-aligned ground truth in MIDI/tab format (each note's string, fret, pitch, onset/offset). *Pros:* High annotation accuracy, string-level detail, diverse playing (chords, solos, etc.). *Cons:* All pieces are in standard

tuning and mostly clean acoustic sound; relatively small size by deep learning standards. License: CC BY (free to download from the researchers' website).

- **EGDB (Electric Guitar Dataset) (2022)** – Introduced by Chen et al. 2022, EGDB is a synthetic-but-real dataset of **electric guitar** recordings. It comprises ~240 GuitarPro tablatures rendered by a *professional guitarist* using a hexaphonic pickup and direct input. Each piece was recorded multiple times with different tone settings. The audio is re-amped through 6 different amplifier simulations to yield a variety of guitar tones. In total ~2 hours of annotated audio (with MIDI note labels) are provided. *Pros:* Electric guitar focus (including distortion, effects), multiple tones per performance for robustness. *Cons:* Limited to the specific songs (tab-derived) and one player; annotation quality is high (coming from MIDI) but expressive deviations are limited since played to tab. Available via the authors (likely on request or via their institution).

- **IDMT-SMT-Guitar (2012)** – A dataset from Fraunhofer IDMT, originally created for instrument recognition and playing technique detection. It includes recordings from 3 guitarists on 6 guitars (5 electric, 1 acoustic) playing scales, chords, and solos. Both **DI (direct input)** and **microphone** signals were recorded. However, only a subset of the audio has detailed note annotations (mostly single notes and some chords). *Pros:* Contains varied guitars and recording conditions, useful for audio effect and technique studies. *Cons:* Not comprehensive for transcription (limited annotated material), and mostly isolated notes or short phrases. It is available on request/Zenodo (often referenced in papers for specific tasks like note onset detection, chord ID).

- **IDMT-SMT-Audio-Effects (2014)** – A related dataset focusing on audio **effects**. It offers ~20 hours of electric guitar recordings of notes, riffs, and chords with various effects pedals and amp settings applied. The task is automatic detection of effects (distortion, flanger, etc.), but it's also useful for training transcription models to be robust to effect-altered timbres. No note-by-note annotation (it's more about classifying effect settings), but could be used as additional training audio for augmentation. Available via IDMT.

- **Guitar Playing Techniques Dataset (2014)** – Collected by Li Su et al. for guitar technique classification. It contains 6,580 audio clips of **individual notes**, each labeled with the technique used to play it (normal, palm mute, slide, bend, vibrato, hammer-on, pull-off, etc.). These are monophonic notes on electric guitar recorded clean. *Pros:* Focused on techniques, sizable number of examples per technique, good for training a classifier to detect techniques from audio. *Cons:* Notes are isolated (not in musical context), so not directly a transcription dataset. However, it's valuable for the "expressive technique" aspect of transcription. Often referred to in literature as a precursor to more comprehensive technique datasets.

- **Magenta Guitar (2019)** – Not a formal dataset name, but Google Magenta's work on guitar transcription produced some data. For example, Hawthorne et al. included some guitar in their multi-instrument transcription model training. No widely released standalone "Magenta guitar" dataset, but some pre-trained models (like **MT3**, see Tools) have been trained on a mix of GuitarSet and other data.

- **Guitar-Aligned Performances and Scores (GAPS) (2024)** – A dataset focusing on **classical guitar**. GAPS contains ~14 hours of real nylon-string guitar recordings aligned with musical scores. Over 200 players in diverse recording conditions contributed, making it very diverse. It provides audio and the aligned notation (but **no video**, despite the name implying "performances"). It's essentially an audio+symbolic dataset: each recording is aligned to a known score (MusicXML). GAPS was used with a benchmark transcription model that adapted piano transcription to guitar, achieving high accuracy. *Pros:* Large and diverse real-guitar audio, score-

aligned (good for training or evaluating transcription). *Cons:* Mostly classical genre, which may not cover techniques like bends/tapping; no explicit string labels in the annotation (classical guitar score doesn't specify string, although alignment + knowledge of classical positions might infer some). GAPS is open-access (Zenodo) and intended to fuel data-driven guitar transcription research.

- **Guitar-TECHS (2024)** – A **comprehensive electric guitar dataset** covering techniques, musical excerpts, chords, and scales [3] [4]. It provides 5+ hours of recordings by 3 pro guitarists, each using different guitars and recording setups, to maximize diversity [5] [6]. Crucially, Guitar-TECHS is multi-modal: it includes *four audio tracks* per performance (direct input, amp mic, head-mounted mic, room mic) [1] [2], plus MIDI annotations for every note (onset, offset, pitch, string via MIDI from a hex pickup) [7] [5]. The content ranges from single notes in various techniques (e.g. sustained, palm-muted, harmonics) to scales, chords (various types), and short musical phrases. *Pros:* Rich acoustic diversity (different hardware, perspectives), precise note labels, covers many playing styles and techniques. Useful not only for transcription but also timbre transfer or source separation research. *Cons:* Still only ~5 hours and mostly isolated or short pieces (not full songs). It's very new (announced ICASSP 2025) and released under CC BY 4.0 [8], available via a dedicated site/Zenodo [8].

- **AG-PT (Acoustic Guitar Playing Technique) Dataset (2024)** – Collected by Stefani et al., aimed at *real-time technique recognition*. It contains ~10–15 hours of monophonic acoustic guitar notes, each labeled with one of 8–12 expressive techniques (normal, staccato, harmonics, rasgueado, etc.). Recorded by 7 guitarists, this dataset emphasizes precise onset annotation (to the millisecond) because it was used to evaluate onset detection's importance in technique ID. *Pros:* Large and diverse for acoustic techniques, fine temporal labeling. *Cons:* Like other technique datasets, it's note-centric rather than full songs. Publicly available on Zenodo (AG-PT-set).

- **Electric Guitar Playing Techniques (Mitsou 2024)** – Often referred to informally (sometimes called "Magcil" in literature), this is a **multimodal** dataset with **549 synchronized video+audio clips** covering 9 different electric guitar techniques. All clips were recorded by one guitarist using a smartphone camera (so realistic setting). Techniques include things like vibrato, bending, tapping, etc., each demonstrated in short exercises. The dataset also provides corresponding **MuseScore files** of the exercises. *Pros:* Contains video, so it's great for training audio-visual models (e.g. to detect techniques from both sight and sound). Real-world capture (phone) increases diversity (lighting, background). *Cons:* Single player, limited musical context (exercises not full pieces). It's open access under CC (on Zenodo as "Guitar Style Dataset" or similar).

- **GOAT (Guitar On Audio and Tablatures) (2025)** – A new dataset of paired **electric guitar DI audio and tablatures**, to be presented at ISMIR 2025. GOAT consists of 5.9 hours of **high-quality direct-input recordings** of various electric guitars, played by different guitarists, with each recording aligned to a GuitarPro tablature (including string/fret and technique info). Furthermore, GOAT employs *data augmentation with amplifiers*: they generated an additional 29.5 hours of audio by reamping the DI tracks through many amp/effect settings, greatly expanding the tonal variety without needing new performances. GOAT provides both the tablature in **GuitarPro format** and a text-like token sequence, making it convenient for training sequence models. *Pros:* Multi-player, variety of instruments, substantial size (~35 hours total audio including augmented), rich annotation (tab with techniques). It's poised to become a standard for training deep models, covering modern electric guitar idioms. License likely CC or similar (authors indicate it's open for research).

- **SynthTab (2024)** – A synthetic dataset by Zhiyao Duan's group. They took ~2,100 GuitarPro songs (from the DadaGP dataset) and synthesized them to audio using high-quality guitar sample libraries. The result is a large paired dataset of audio and ground-truth tabs/MIDI. SynthTab's creators used it for pre-training a transcription model, but noted that a model pre-trained on SynthTab plus fine-tuned on real data did not dramatically outperform one trained on real data only. Nonetheless, SynthTab provides potentially millions of note examples and can be useful for data augmentation or pre-training, especially for neural networks that need lots of training data. It covers a wide range of genres (whatever exists in GuitarPro corpus, e.g. rock, metal, jazz, etc.). The dataset can be reconstructed from GuitarPro files using virtual instruments; the authors have also shared pre-rendered data on request.

- **DadaGP (2021)** – Not an audio dataset but a **symbolic dataset** of 26,000 guitar tablatures (GuitarPro files) tokenized for sequence models. Sarmento et al. compiled this from publicly available tabs, providing a large corpus for training models like Transformers to understand tab sequences. It was used to train the MIDI-to-Tab model and in SynthTab's selection. *Pros:* Huge vocabulary of real-world guitar playing (all genres), includes many techniques (whatever tab notation had). *Cons:* No audio. But it's very useful for any symbolic modeling or as source material to synthesize audio. Available under CC licenses (since based on user-contributed tabs) via ISMIR 2021 paper repo.

Each dataset above has specific strengths. For building a state-of-the-art A/V transcription system, one might use *GuitarSet* and *EGDB* for audio model training (for note transcription), *AG-PT* and *Mitsou's dataset* for training technique classifiers (audio or video), *Guitar-TECHS* and *GOAT* for training/fine-tuning on diverse electric guitar scenarios (and validating multimodal aspects, since Guitar-TECHS has multi-mic and GOAT has multiple players), and possibly *GAPS* for classical guitar focus. Having video data is still relatively rare; for that, the Mitsou dataset (electric techniques) and any self-collected videos or smaller sets (like Asmar's RGB-D recordings, or URMP multi-instrument video dataset which has some guitar duets) may be utilized.

Licensing in academic datasets is generally permissive (most are CC BY or similar). Download locations: GuitarSet and Guitar-TECHS have dedicated websites or GitHub (e.g., GuitarSet on github.io, Guitar-TECHS on guitar-techs.github.io [8]), IDMT datasets on the IDMT website/Zenodo, GOAT on arXiv/ISMIR repo (likely Zenodo), Mitsou's on Zenodo (DOI in Data in Brief paper), AG-PT on Zenodo, etc. In practice, assembling a training set for a project might involve pulling from multiple of these sources to cover all bases (acoustic, electric, techniques, etc.).

2.5 Surveys, Theses, and Meta-Literature

For those seeking an overview of AMT and guitar-specific research, several survey papers, doctoral theses, and meta-studies provide valuable insights:

- **Automatic Music Transcription Surveys:** A comprehensive survey by Benetos *et al.* (2018) covers the state of AMT up to the deep learning era, focusing on algorithms for multi-pitch detection and notation transcription (though not guitar-specific). It discusses evaluation metrics and datasets in AMT. Another good reference is the book *"Fundamentals of Music Processing"* by Müller (2015) which includes chapters on music transcription and alignment techniques.

- **Music Information Retrieval (MIR) Surveys:** The field of MIR has periodic overview papers; e.g. *"Deep Learning for MIR"* (Choi et al. 2017) touches on transcription as one task among many,

highlighting how CNNs/RNNs were applied to audio tasks. While not specific to guitar, these give context on where guitar transcription sits in the MIR landscape.

- **Guitar-Specific Overviews:** Wiggins' 2019 ISMIR paper intro provides a concise overview of guitar transcription challenges (redundant pitch layout, variety of techniques) and mentions prior work such as pitch-first-then-tab approaches and HMMs for chord estimation. For a deeper dive, *Andrew Wiggins' thesis or technical report* (if available) might expand on TabCNN and its background. Similarly, Chen's 2022 paper introduction (and references) summarize recent guitar AMT efforts and datasets, and Chieppa et al. (IEEE Access 2025) devote sections to related work and dataset availability, effectively functioning as an up-to-date literature review in their article.

- **PhD Theses:** A few dissertations are seminal in guitar tech:

- *Jakob Abesser (2013)* – "Automatic Transcription of Bass Guitar Tracks…" which tackled bass transcription and even genre classification using transcribed features. He explored signal processing and machine learning for bass, including identifying plucking vs slapping techniques.
- *Dmitry Dzhambazov (2017)* – worked on guitar transcription and chord recognition (several papers on multi-pitch and tablature decoding).
- *Mark Asmar (2022 MSc)* – "A Computer Vision-Based Automatic Transcription of Guitar Music from RGBD Videos." This thesis focuses entirely on the vision side: detecting fretboard, fingers, and outputting tabs from video, which is a unique perspective.
- *Anna Hamberger (2025)* – likely her dissertation (in progress) on symbolic guitar transcription (since she's behind Fretting-transformer and related work). Keep an eye on research from her group for thorough treatment of MIDI-to-tab problems.

- *Domenico Stefani (2024)* – Though his focus is real-time technique recognition, his work (PhD in progress) and related publications give a modern perspective on guitar technique datasets and latency considerations.

- **MIR/Audio Conference Tutorials and Workshops:** ISMIR often has tutorials; for instance, ISMIR 2019 had a tutorial on deep learning in music transcription. Also, the "Late Breaking/Demo" sessions often feature overviews of ongoing projects – scanning ISMIR proceedings for "guitar" yields many hits from 2019–2024, indicating active research. There are also workshops like the International Guitar Technology Workshop or AES (Audio Engineering Society) conventions where overviews of guitar signal processing are presented.

- **Meta-Papers on Evaluation:** For understanding evaluation, papers by Bay, Ehmann, and Downie on MIREX evaluations of transcription (piano focus but methodology-general) can be useful. Also, *Ewert, Müller, and others (2019)* wrote about evaluation metrics for transcription, onset F-measures, etc., which is meta-literature that helps interpret research results.

- **Theses on Symbolic Music and Guitar:** There are some works on generating guitar tab from MIDI (symbolic AI side) like studies by Radisavljevic (2004) on guitar fingering DP, or more recently, Drew Edwards (PhD student at QMUL) working on sequence models for tab – their arXiv 2024 paper and possibly thesis might be a resource.

- **Music Engraving & Notation**: While not MIR, there are texts on music notation (e.g., Behind Bars by Gould) that are useful when dealing with how transcribed output should be formatted, and LilyPond's community documentation for engraving rules.

In summary, for a broad understanding: start with AMT surveys for background, then read guitar-specific papers (Wiggins 2019, Chen 2022, Hamberger 2025, Chieppa 2025) for focused insight. Technique classification papers (Su 2014; Keun Lee 2019 on detecting techniques) give another dimension. The field is evolving rapidly (as evidenced by multiple 2024–2025 references), so staying current with conference proceedings (ISMIR, ICASSP, DAFx, AES) is important. Fortunately, many authors like Xavier Riley, Hegel Pedroza, etc., have personal pages summarizing their work which can serve as up-to-date guides.

Tools, Models, Libraries & Existing Software

Developing an audio-visual transcription system can leverage a host of existing tools and libraries. These span from general-purpose audio processing and machine learning frameworks to specialized music and vision toolkits. We organize them by category:

3.1 Audio Transcription Tools

- **Onsets & Frames (Magenta)** – An open-source implementation of Hawthorne et al.'s piano transcription model. It's available as part of Google Magenta (TensorFlow) and has pre-trained weights for piano. Users have fine-tuned Onsets & Frames on guitar (e.g., on GuitarSet) to create guitar transcription models. The model outputs a pianoroll of notes with onset indicators, which can be decoded into MIDI. *Strengths:* High note accuracy for polyphonic transcription, time-aligned note events; fairly lightweight to run on a GPU in real-time for single instrument. *Weaknesses:* Out-of-the-box, it's piano-trained (so fine-tuning is needed for guitar timbre). It doesn't output string numbers – just pitches and timing.

- **MT3 (Multitask Multitrack Transcription)** – A Transformer-based model from Google that can transcribe multiple instruments and output instrument labels (like "note X by instrument Y"). It treats transcription as a sequence-to-sequence problem (audio spectrogram in, MIDI-like events out). For guitar, one could train a specialized MT3 to output notes with a guitar instrument tag. *Strengths:* Very general and powerful; can incorporate prior knowledge (e.g. a vocabulary for strums vs single notes). *Weaknesses:* Heavy model (Transformer) requiring a lot of training data; not guitar-specific output (no tab, only MIDI), and might need post-processing to map to tab.

- **CREPE** – A pre-trained deep model for *monophonic* pitch tracking. It operates on short windows of raw audio and outputs a continuous pitch estimate (frequency) with confidence. CREPE is great for tracking melodies, detecting bends (as continuous pitch glides), and detecting vibrato rate/depth. In the bass guitar A/V system, CREPE was used to get the pitch trajectory and segment note onsets. *Strengths:* State-of-the-art accuracy in single-f0 tracking, easy to use (available in Python via `crepe` library), real-time capable (can do 100+ estimates/sec). *Weaknesses:* Cannot handle chords (only one pitch at a time). Needs careful post-processing to decide discrete notes and onsets from the continuous output.

- **Librosa** – The go-to Python library for audio DSP. Provides routines for loading audio, computing spectrograms, constant-Q transforms, onset strength envelopes, tempo detection, etc. One can quickly prototype an onset detector using `librosa.onset.onset_detect` or a simple pitch tracker via autocorrelation/YIN provided. *Strengths:* Very easy to use, well-documented, extensive functionality for feature extraction. *Weaknesses:* It's more of a building block toolbox, not a ready-made transcription system. Also, for very large audio or real-time, core routines in librosa (numpy-based) may be slower than optimized libraries.

- **Torchaudio / Tensorflow Audio** – These offer GPU-accelerated audio processing and integration with ML frameworks. For example, Torchaudio can compute mel-spectrograms on the fly in a PyTorch data pipeline (useful for training models without preprocessing). They also have some pre-trained models (e.g., HuBERT or Wav2Vec acoustic features that could be adapted for music tasks). *Strengths:* Speed and seamless integration into model training. *Weaknesses:* Lower-level (one still needs to design the model that consumes these features).

- **Essentia** – A C++/Python library for audio and MIR, developed by MTG. It has algorithms for pitch detection (YIN, multi-pitch via HPCP), onset detection, tuning estimation, key detection, etc. For instance, Essentia's multi-pitch algorithm could give a set of pitch candidates per frame, which one might use as input to a custom decoding algorithm for tablature. *Strengths:* High-performance implementations, many MIR algorithms out-of-the-box. *Weaknesses:* Not deep learning based (so may not reach SOTA accuracy in transcription tasks), and using it in Python requires compiling or using their pre-built binaries.

- **Sonic Visualiser + Vamp Plugins** – Sonic Visualiser is a GUI tool, but its engine (Vamp plugins) include various transcription-related plugins (like Tony for monophonic pitch, polyphonic note trackers, onset detectors). As a developer, one can use Vamp plugin libraries to run these algorithms on audio files. For example, QM Vamp Plugins provide a **note transcription plugin** (for piano, which can somewhat apply to guitar) and chord detection plugin. *Strengths:* Quick way to get baseline transcriptions or visualizations for analysis/validation of your own results. *Weaknesses:* Not really a library to integrate into a pipeline (more for offline analysis), and accuracy is limited compared to latest ML models.

- **MIDI Processing Libraries** – Once notes are detected, libraries like `pretty_midi` (Python) or `miditoolkit` help to assemble them into MIDI files. They handle timing, velocity assignment, etc. For guitar, one might set the instrument to a guitar patch and even encode string number via MIDI channels or standard MIDI guitar meta-data.

- **mir_eval** – A Python library with standard evaluation routines for transcription (e.g., precision/recall for notes within a tolerance), as well as metrics for overlap, onset-only scoring, etc. Useful to evaluate audio transcription models against ground truth in development.

In practice, a developer might use a combo: e.g., use Torchaudio/Librosa to get spectrograms, feed into a PyTorch implementation of Onsets & Frames or a custom model, use CREPE for fine pitch when needed (e.g., to detect bends), and use `pretty_midi` to output initial MIDI. That output could be refined or combined with other steps (like a string assignment algorithm). For a quick start, one could even use a pre-trained Onsets&Frames model (or an available checkpoint fine-tuned for guitar) to get MIDI and then apply a separate tab inference model.

3.2 Guitar-Specific Modeling Tools

- **Fretting-Transformer** – An encoder-decoder model (T5 architecture) that translates a sequence of MIDI notes into a sequence of guitar tablature actions. Although a research model (by Hamberger 2025), its concept can be implemented with libraries like Hugging Face Transformers. The model's *vocabulary* includes pitch tokens and string-fret tokens, enabling end-to-end string assignment learning. The authors provide post-processing code for overlap correction (ensuring no impossible finger overlaps). If open-sourced, this model would be a great plug-in component: feed it the note sequence (with timing) output by an audio model, and get back a possible tablature. *Strengths:* Learns realistic fingering patterns from data (e.g., it

might implicitly learn position shifts, avoiding extreme stretches). *Weaknesses:* Requires a large symbolic training set (they used SynthTab+DadaGP) and fine-tuning on real data to avoid unrealistic results; it outputs tab without knowing the audio, so if audio transcription had errors, it will still tab those errors.

- **TabCNN** – Wiggins & Kim's CNN model (2019) for tablature estimation. They haven't released code widely, but one can reimplement it: it's basically a convolutional network taking a time-frequency representation and outputting a 6-channel piano-roll (one channel per string, indicating fret number or rest). A simplified version could be built in PyTorch. *Strengths:* Direct mapping to tab, no separate string assignment step needed. *Weaknesses:* Hard to enforce physical consistency (their model might predict impossible combos, which then need filtering). Also, CNNs have trouble modeling long-term dependencies (e.g., a consistent position shift).

- **Electric guitar tone models** – The 2024 paper by Pedroza et al. mentions using *real electric guitar tones and effects* to improve robustness. That indicates they possibly trained models with augmented audio (re-amped through pedals/amps). While not exactly a tool, the idea is to incorporate guitar effects chains in training. Tools like **Guitarix (Linux)** or VST plugins can be automated to generate distorted versions of clean samples. So one might integrate an *effects modeling pipeline* during training to randomize the amp settings (a form of data augmentation).

- **Playability Constraint Solvers:** Prior to deep learning, various algorithmic solvers for string assignment exist (Sayegh's DP, A *search on fretboard graphs). Libraries implementing these are not off-the-shelf, but one can find code in some research repos (e.g., Burlet's guitar tab finder code, or the music21** library's guitar module, if any). One notable mention: `pytablature` is a Python lib that had some basic tablature processing, and `guitarPractice` (a project on GitHub) that given notes and tuning can enumerate possible fingerings. These can be used to post-process transcription results: e.g., generate all feasible fingerings for a note sequence and then select the one with minimal shifts.

- **MIDI-to-Tab models:** The 2024 "Midi2Tab" by Edwards et al. is on arXiv. It uses a BERT-like masked modeling to infer missing string assignments. As a tool, one might repurpose their released model (if available) – feed it a MIDI and get tablature proposals. Alternatively, train a new one using HuggingFace's Transformers library with their methodology (mask certain tokens in tab sequences and train the model to fill them, then use it to generate). *Strengths:* Language-model approach can capture statistical patterns of guitar tab (e.g., common chord shapes, scale fingerings). *Weaknesses:* It doesn't directly use audio, so it's only as good as the input MIDI (garbage in, garbage out for poorly transcribed notes).

- **Neck Profiles / Guitar Profiles:** Some software like **GuitarPro** or **TuxGuitar** have internal logic for suggested fingerings and can import MIDI and attempt to map to tab. These aren't exposed as libraries, but it's worth noting they exist. For example, MuseScore's guitar staff type will attempt to place notes on strings automatically (usually defaulting to first position). These can serve as a baseline, but custom coding is needed for optimal results.

- **Machine Learning Libraries:** Frameworks like PyTorch and TensorFlow are essential to implement custom models. Specific to our task, **PyTorch Lightning** can speed up training prototypes for transcription models. Also, **sklearn** might be handy for simpler tasks (SVM classifier for techniques if one is not using deep learning, as done in Mitsou et al. 2024 who tried SVM/CNN for technique classification).

- **Pre-trained Models / Checkpoints:** Some researchers release trained models: e.g., Kong et al.'s High-Resolution Transcription model (which Riley adapted for guitar) might have a public checkpoint. Facebook AI Research's **Omnizart** or **Esitar** models (if any) could be checked. Always look at ISMIR and ICASSP papers' repos.

In summary, while there isn't a one-click "guitar transcriber" library yet, we can piece it together: use an *audio model* (Onsets & Frames variant or Transformer) for note detection, then use a *string assignment model* (like Fretting-Transformer or a simpler DP) for tabbing, then perhaps a *technique classifier* (could be a small CNN or an ensemble: e.g., detect vibrato by audio periodic modulation, detect bends by continuous pitch curve from CREPE, detect slides by noting consecutive semitone changes with legato in audio). Each of these components has either an existing codebase or at least a well-documented method we can implement with available libraries.

3.3 Computer Vision Tools

- **MediaPipe Hands** – A state-of-the-art solution for real-time hand tracking from Google. It provides 3D coordinates for 21 hand landmarks (five fingers, each with joints up to fingertip, plus palm base). It's extremely fast (runs on CPU/mobile) and robust under a variety of conditions. For guitar, one can use MediaPipe to get left-hand finger positions per frame. Many projects (like Fret-Nav 2025) use it as a first step. *Strengths:* No training needed (pre-trained model), high accuracy for typical hand poses, easy integration (Python, C++, even web via TF.js). *Weaknesses:* Tends to lose tracking if the hand is rotated in unusual ways (e.g., extreme angles relative to camera), might confuse finger identity if two fingers stick very close together, and doesn't explicitly provide which object the hand is interacting with (we must map it to guitar fretboard ourselves). Also, if the fretboard is far from the camera or blurry, MediaPipe might fail to detect the hand.

- **OpenPose** – An open-source library (C++) for pose estimation that can be extended to hand keypoints (with an additional model). OpenPose can detect multiple people and their body poses; for guitar, one would focus on one person's hands. It's heavier than MediaPipe and slower, but offers more flexibility (you can finetune models, etc.). *Strengths:* Multi-person capable (if you had a scenario with two guitarists?), and customizable. *Weaknesses:* Computationally intensive, and the hand module may require a good GPU to run in real-time; for a single hand, MediaPipe is usually sufficient.

- **MoveNet** – A newer pose estimator by Google (for whole-body) that's very fast. It doesn't output detailed hand joints (only wrist position), so it's not directly useful for finger tracking. If one needed to track the guitarist's overall posture or right-hand strumming motion, MoveNet could be used for the arm movement, while another method handles the fingers.

- **YOLO / Detectron2** – These are object detection frameworks. How are they relevant? They can detect objects like "guitar" or "hand" as bounding boxes. One use-case: YOLO could detect the guitar body and fretboard in the frame, providing a quick way to isolate the Region of Interest for further processing. For example, you could train a YOLO to detect the rectangle of the fretboard, which might be easier than trying to detect fret lines directly. Or detect "left hand" and "right hand" positions on the guitar. *Strengths:* Real-time capable detection of custom classes (with training). *Weaknesses:* Requires annotated images to train (unless using a pre-trained model for generic guitar detection). Might not be necessary if simpler image processing suffices.

- **OpenCV (Geometry & Tracking)** – OpenCV comes into play for tasks like edge detection, Hough transforms (to find strings/frets), perspective transforms (to flatten the fretboard). For instance, one can use `cv2.HoughLines` to find string lines in a binary image, or `cv2.findContours` to detect the outline of the fretboard. Once the corner points of the fretboard are identified, `cv2.getPerspectiveTransform` can compute a warp matrix to a canonical fretboard shape (e.g., 6 strings horizontal, frets vertical). This rectification makes subsequent analysis (mapping finger px to fret number) easier – many systems do this as a first step. OpenCV is also useful for **color detection** (some guitars have dot markers – one can threshold to find them) and for basic optical flow if we want to track motion of a finger blob frame-to-frame.

- **Marker-based Tracking** – If one has the luxury of adding markers: placing small colored stickers on the fingertips or fret positions. Then simple color filtering (OpenCV in HSV color space) can track those points. It's not research-y (since it simplifies the problem), but it can be a pragmatic solution for a prototype. ArUco markers (fiducial tags) could be placed on the fretboard (e.g., one on first fret, one on 12th fret) to allow automatic calibration of the fretboard plane.

- **Depth Cameras** – Tools like the Kinect SDK or Intel RealSense SDK allow capturing depth maps. For instance, using a RealSense camera, one can get the 3D position of fingertips relative to the guitar. Mark Asmar's work used an RGB-D camera to mitigate occlusion (the depth data can see which finger is closer even if overlapping in image). Though not a library per se, using depth requires calibration (aligning depth to color image) which libraries provide utilities for.

- **Vision ML Libraries** – If customizing a model, frameworks like PyTorch also cover vision models. One could fine-tune a **keypoint RCNN** (Detectron2 has one) to detect finger positions on a guitar specifically. Or use **MediaPipe BlazePose** in a custom way by adding additional keypoints for guitar context.

- **Fretboard-specific Heuristics** – Implementations from research: e.g., Bae et al.'s approach of clustering lines and circles. One can replicate these with SciPy (for clustering) and OpenCV (for detection). Another nifty trick: If the fretboard is fairly flat and dark, one can do a projection profile – sum image pixels horizontally to find peaks where frets occur (frets are slightly reflective lines). If camera quality is high, sometimes a simple image filter (like Canny edge) and Hough can get both strings (long horizontal lines) and frets (short near-vertical lines), then one can intersect them to form a grid.

In sum, for an implementation: **MediaPipe Hands** will handle most of the heavy lifting for finger tracking. **OpenCV** will help locate and warp the fretboard, and possibly refine exact fret positions. If needed, **YOLOv5** could be trained to detect the fretboard region for robustness to background. These tools are all open-source and well-documented, making it feasible to put together a vision module that takes video frames and outputs something like: hand pose (finger tips) in world coordinates (string/fret units).

3.4 Music Engraving and Symbolic Tools

- **music21** – A Python library for music theory and analysis that also handles creating and manipulating symbolic music (notes, chords, streams). One can use music21 to assemble a score: create a `Stream` for a guitar part, insert notes with specified string and fret (music21 has a `guitarTab` instrument that can encode tabs). Music21 can output MusicXML or even LilyPond format. It's quite powerful for algorithmic composition and could be used to transform raw transcription output into proper notation (assigning noteheads, inferring key signature from

accidentals, etc.). *Strengths:* High-level interface to musical concepts; can auto-generate common notation from pitches and durations. *Weaknesses:* Guitar tablature support is a bit limited (music21 might not automatically handle all tablature notations like bends or slides without custom code). Also, rendering to actual staff images requires MusicXML+external software or its internal archaic TinyNotation.

- **GuitarPro and PyGuitarPro** – GuitarPro is a popular tablature editing software (.gp file format). The library **PyGuitarPro** can read/write GuitarPro files (supports versions 3-5 reasonably well, and partially GP7). If one wants to output a tab that users can open in GuitarPro or TuxGuitar, this is useful. It supports encoding fingering, techniques (bend, slide, hammer, etc.), and even rhythmic values. *Strengths:* The .gp format is widely used by guitarists; encoding output here means instant accessibility (user can tweak in GuitarPro app). *Weaknesses:* The format is somewhat complex and undocumented (PyGuitarPro handles most basics but might not support every nuance of newer versions). Also .gp is proprietary (though older versions are reverse-engineered).

- **MuseScore** – An open-source notation software which supports tablature staves. MuseScore has an API (via plugins in QML/JavaScript) and can also be invoked via command line to convert files (MIDI to MusicXML, etc.). One approach: output MusicXML from your system and let MuseScore render it to PDF or display. *Strengths:* Free and high-quality engraving, with support for tab + standard notation linked. It also has a large community; the user could upload the MusicXML to MuseScore.com for sharing. *Weaknesses:* As a library, it's not easily embedded (you might automate the GUI or use command line to render, but it's not a Python library). Also, automatic placement of fingering in standard notation might not match a guitarist's preference, so some manual or heuristic adjustment may be needed.

- **LilyPond** – A text-based engraving software that produces beautiful sheet music. One can programmatically generate a LilyPond file (.ly) describing both standard notation and tab. LilyPond supports guitar-specific notations (fingering numbers, string numbers, bends, slides, even quarter-tone bends). *Strengths:* Arguably the best engraving output; fully customizable. Can be run from command line to produce PDF or SVG output. *Weaknesses:* Steeper learning curve to get the syntax right. Also, compiling LilyPond in large batches is slower. If fine visual control is not needed, might be overkill compared to MusicXML.

- **MusicXML/MEI** – These are interchange formats for music notation. **MusicXML** is widely supported (importable into Finale, Sibelius, MuseScore, GuitarPro, etc. to varying degrees). A transcription system could output MusicXML: include a tablature part with `<technical>` tags for string and fret. MusicXML supports bent notes, slides (as notations tied to notes). **MEI** is a more academic XML format that can capture even more detail and annotations; however, fewer programs support MEI out of the box. Typically, you'd choose MusicXML for compatibility. Libraries like `music21` or `sketchpy` can help generate MusicXML.

- **MIDI** – The lowest common denominator symbolic format. A straightforward output of a transcription is a MIDI file with notes. MIDI is great for listening to the transcription and for some evaluation, but it loses guitar-specific info (all it has is pitch, timing, velocity). Standard MIDI can indicate different strings by using different channels (MIDI guitar uses channels 1-6 for strings 1-6, often), but this is a convention not universally used. There is an extended format called **MIDI Tuning Standard** which can specify string tuning and string assignment via SysEx, but it's complex and not widely supported. Nevertheless, outputting MIDI is easy and you can then input that into other software for human-readable notation.

- **Engraving Challenges:** When converting to sheet music + tab, one must decide on some editorial matters:

  - Rhythm quantization: The raw transcription times have to be quantized to the nearest suitable note value (e.g., a note at 1.237s might be intended as a triplet quarter note). Some tools or libraries (like `pretty_midi` has a crude quantize, and MuseScore can quantize on import to some extent) but often this is non-trivial. For initial research, one might keep the transcription as a sequence of time-stamped notes and not worry about rhythmic notation.
  - Multi-voice notation: Guitar music often has polyphonic voices (bassline and melody). Transcription output as just a list of notes won't group those. Properly splitting voices might require algorithmic grouping by pitch range or onset clustering – beyond basic tools, possibly manual or heuristic processes (some research has looked at voice separation in polyphonic transcription).
  - Tablature layout: Deciding line breaks, etc., is usually handled by the engraving software, but if a certain fingering is super awkward, a human engraver might choose an alternate. Automated systems might not catch all those nuances. There are projects on *automatic arrangement simplification* (as TART mentions training an LSTM to simplify fingerings for playability).

- **Integration Tips:** A pragmatic pipeline:

  - Use your AI model to get notes + strings + techniques.
  - Create a MusicXML: list each note with its duration (estimated or quantized), put fret number and string number as technical indications. If technique like bend, add `<bend/>` element; slides can be notated with slur or glide lines in MusicXML.
  - Open that MusicXML in MuseScore or GuitarPro to check. Iterate on representing special techniques correctly (each program has quirks in how it shows them).
  - Export PDF or share as needed.

Alternatively, for a quick tab-only output, one could output an **ASCII tablature** (just text). Indeed, TART's final stage generates ASCII tab text, which is human-readable in a monospaced font. ASCII tabs use lines of numbers and can include special characters for techniques (e.g., `3h5` for hammer-on from fret 3 to 5). ASCII tab is not as formal but is easy to produce. Libraries like `taborama` can help format ASCII tabs.

- **Evaluation Tools:** In symbolic domain, there's `tab_eval` (if any) or one might have to create custom scripts to compare a generated tab against ground truth (taking into account equivalent fingerings, etc., which is complex).

In summary, the symbolic output stage has robust tools: for polished output, MusicXML + MuseScore (or LilyPond) is recommended. For quick prototyping, ASCII tab or MIDI is fine. The key is ensuring the string and fret info from the transcription is preserved through to the final notation. Fortunately, standards like MusicXML were built with guitar in mind, so they support most of what we need (six-string tablature staffs, etc.). With these tools, the end result can be a professional-looking score that belies the complexity of the system that created it.

Architecture Patterns (How SOTA Systems Are Built)

Based on the literature and existing systems, several common architectural patterns emerge for designing an audio-visual guitar transcription system. We outline these patterns in terms of pipeline components and how they fit together:

4.1 Audio → MIDI Pipeline (Audio-Only Subsystem)

Overview: The audio pipeline converts raw audio into a sequence of **note events** (pitch + onset time + offset time). This often mirrors a piano transcription pipeline but with some guitar-specific tweaks.

- **Input Processing:** Audio is first transformed into a time-frequency representation (typically a spectrogram or constant-Q transform). High-resolution methods use finer time steps around onsets for accuracy. Some pipelines use multi-band or multi-resolution spectrograms (to capture both attack transients and sustained harmonics).

- **Pitch Detection Model:** A neural network (CNN, RNN, Transformer) then estimates the probability of each pitch being active at each time frame. The *Onsets & Frames* architecture splits this into two outputs: an onset activation output (high when a new note starts at frame) and a frame activation output (high when a pitch is sounding). Combining these yields discrete notes. Transformer models, on the other hand, might output a sequence of note events directly, thus performing a sort of end-to-end detection with context (like MT3 enumerating a list of notes).

- **Post-Processing & Smoothing:** Raw model outputs often have spurious detections or noisy durations. Common post-processing includes:

- *Thresholding* frame activations and connecting consecutive active frames for the same pitch into one note (stopping at offsets).
- *Non-maximum suppression* on onset predictions to avoid double-triggering for the same note.
- *Temporal smoothing:* some use HMMs or simple rules to avoid jitter (e.g., require a minimum duration for a note or a minimum gap between same pitch re-attacks).

- *Tuning adjustment:* guitars can be slightly off standard tuning; some systems estimate overall tuning (e.g., via detection of spectral peaks across the piece) and adjust pitch centers accordingly, so that, say, 329 Hz is labeled as E4 not D#4.

- **Output Representation:** The result is typically a list of MIDI-like notes: each with start time, end time, and pitch (plus possibly a confidence or velocity). These can be represented as a MIDI file or kept in memory as a list for further processing. Multi-instrument systems might also assign an instrument label, but here we assume all notes are guitar.

Best Current Pitch Models: Onsets & Frames (2018) set a strong baseline. Kong et al.'s High-Resolution model (2021) improved piano transcription by directly regressing onset and offset *times* with higher temporal resolution. Applied to guitar by Riley (2024), the high-res model helped capture subtle timing (like arpeggio as slightly rolled chords). More recent models like the Transformer by Chen (2022) used multi-loss: it had a loss for onset prediction, a loss for frame prediction, and even one for offset, to better learn note lengths. In terms of accuracy, these models can reach >80% note F1 on guitar

benchmarks (GuitarSet test splits). They still make mistakes in dense chords or very fast passages (sometimes missing a note that is faint or mis-segmenting two close notes as one).

Error Modes: Audio models often confuse octaves or fifths in chords if the harmonic series overlaps – e.g., distinguishing a power chord (root+5th) can be tricky if the 5th's overtone blends with the root's. Also, percussive **string slaps or muted strums** might be falsely transcribed as notes (onset detector triggers but pitch is noise). For guitar, *polyphonic sustain* (notes ringing together) is hard – models might falsely think a note ended when it's just softer, or vice versa, especially without pedal to damp like piano has. Another error mode is missing very **fast notes** (if they're shorter than the frame hop or blurred, an onset might be skipped) – some research emphasizes precise onset annotation to train models to catch quick notes. Finally, if the audio has **multiple guitars or backing track**, an audio model might transcribe everything – isolating the target guitar is assumed but not always guaranteed.

Temporal Considerations: Some pipelines treat transcription as *frame-wise classification* (each 10 ms frame predicts which pitches are on). This yields high time resolution but requires grouping into notes post-hoc. Others treat it as *sequence labelling* where onsets are events on a timeline. For guitars that often have clear plucked onsets, the onset-based approach is very useful. Many use a combination: first identify onsets, then for each onset, decide what pitch(s) started, and when they end (looking until next onset or until energy decays below threshold). This ensures each note has a well-defined attack time, aligning with how guitar notation works (attack-driven).

Multi-instrument vs Single-instrument assumptions: If our audio model is guitar-specific (single-instrument), it can exploit expected characteristics: e.g., a maximum polyphony of 6 notes (since 6 strings) – though practically rarely all 6 sound distinctly in a chord due to overlap. It can assume the instrument range E2–E6 (for standard tuning), which simplifies output space. Multi-instrument models (like general AMT models) might waste capacity on pitches or timbres not relevant to guitar; however, they offer flexibility if other instruments leak into the audio. For our system, since we have video focusing on a guitarist, we assume single-instrument (target guitar) scenario.

In block diagram form, the audio pipeline is: **Audio waveform → Spectrogram → Neural model → Note events.** This then feeds either directly to a tab mapper or into fusion with video. Key is robustly extracting notes with timing as the foundation for everything else.

4.2 Video → Fingering/Position Pipeline (Visual Subsystem)

Overview: The video pipeline takes frames (or a video stream) and produces information about which string and fret the guitarist's fingers are on over time. It essentially tracks the **left hand finger positions** and sometimes the **right hand picking** if needed.

- **Fretboard Localization:** Initially, the system must detect the fretboard area in the video. If the camera is fixed and we know the fretboard position from the first frame (either manually or via detection), we can lock onto it. Automatic detection might use identifying features: e.g. find the long straight lines of strings (using Hough transform) or detect the headstock (guitar tuning pegs area) as a reference point. Once found, we often define a transform to a normalized coordinate system where one axis is along the strings (nut to bridge) and the other along the frets (E string to e string). For instance, after detection, many systems do a **perspective warp** so the fretboard in the image becomes a rectangle of size (string_count × fret_count). This compensates for camera angle. For tracking across frames, techniques include:
- *Template tracking:* e.g. track the position of the fretboard by optical flow or by re-detecting the known fretboard corners each frame.

- *Assume static camera and stationary guitar (for at least left-hand region):* many demos assume the guitarist mostly stays in one spot relative to camera, simplifying things.

- **Hand Detection:** Identify where the left hand is. This can be via pose estimation (like MediaPipe that directly gives hand joints) or simpler: find the largest skin-colored blob near the fretboard. A hand detector (like a Haar cascade or an SVM on skin texture) can also be used. Knowing the hand's bounding box can focus subsequent processing.

- **Finger Segmentation/Pose:** Two main approaches:

- **Keypoint Pose (Kinematic):** Use a model to get joint coordinates (thumb, index knuckle, index tip, etc.). From these, extract fingertip positions of interest (index through pinky tips). This gives direct coordinates for each fingertip.
- **Vision Segmentation (Image):** Use image processing to identify finger regions – e.g., subtract background, threshold by color to isolate fingers, maybe use morphological operations to separate fingertips as blobs. However, differentiating multiple fingers in one blob can be hard.

Pose methods have essentially taken over due to reliability. Once fingertip pixel positions are known, we map them to the fretboard coordinate system. E.g., if after perspective warp the fretboard is normalized to (string index, fret distance), the fingertip (x,y) in that space can be mapped to a specific string (x closest to one of the 6 string lanes) and fret (y falls between two fret lines). Some systems actually compute distances: e.g., distance of finger to each fret line, pick the nearest fret line **behind** the finger (towards the headstock) as the fret being pressed. Bae et al. added a padding (10% of fret distance) around each fret to improve mapping stability – essentially to ensure if finger is just behind a fret it still counts as that fret.

- **Occlusion Handling:** If a finger is occluded, pose models might still hallucinate it in a plausible location based on others. But sometimes one or more fingertips might not be detected at all. In such cases:
- The system can maintain state: e.g., if last frame a finger was on fret 5 string 3, and this frame it's missing, perhaps assume it's still there unless we see it clearly elsewhere.
- Multi-view can help (one view might see what the other cannot).

- Filtering: If index and ring finger swap occlusion, sometimes pose will confuse which is which – tracking the continuity of each finger's path over time (assignment problem) can catch sudden impossible jumps and swap identities back.

- **Temporal Tracking:** We likely want a smooth estimate of finger positions over time. A Kalman filter for each fingertip can be used to predict next position and smooth the noise. Also, by tracking, we get finger **velocity** – which can hint at actions (fast movement may indicate an upcoming slide or position change). If multiple fingers are down, tracking their coexistence can identify chords (e.g., if index, middle, ring are all pressing around the same time within a fret or two, that forms a chord shape).

- **Multi-Camera vs Single-Camera:** A single camera (front view) is simplest, but as noted, occlusions and depth ambiguities occur. Multi-camera setups:

- *Front + Side:* A side camera could see finger placement from a different angle, resolving some occlusions (used in some research labs).

- *Stereo (3D reconstruction):* Two cameras can triangulate the 3D position of fingertips on the guitar, which directly gives you string and fret (since guitar fret positions in 3D are known via geometry).
- *Egocentric (head-mounted):* Offers a view similar to what the player sees – this often shows the fretboard from above, which can clearly show which fret a finger is on (though finger might hide the fret marker).

Multi-cam fusion could mean simply averaging detections or using one as fallback for the other. It increases complexity and cost, so many systems stick to one camera plus smart processing.

Right-Hand (Picking) Considerations: The prompt didn't emphasize it, but a full transcription might also encode right-hand techniques (fingerpicking patterns, whether a note was plucked with thumb or finger, or with a pick, whether it was a downstroke or upstroke). Capturing this requires a camera on the strumming hand. Some research (esp. for classical guitar) tries to identify which right-hand finger plucked a string by video. This is an open problem and often not attempted in transcription systems, as it doesn't reflect in standard notation (except for special markings like p,i,m,a for classical fingerings if one wanted to annotate that).

Output of Video Pipeline: Typically a time series of "finger events". For example: - At time 1.20s: Index finger on 3rd string, 5th fret. - At time 1.25s: Middle finger on 3rd string, 6th fret (maybe sliding?). - etc.

This could be distilled to "note events with string+fret" if you also incorporate onsets. But often, the video pipeline doesn't inherently know *when* a note is played, only where the fingers are. So the fusion stage is where timing from audio is combined with positions from video to label each note event with a string+fret. There are exceptions: if a video shows a finger lifting off, that could mark a note release (offset), but audio handles that better.

Fretboard Geometry and Calibration: Many systems pre-calibrate the mapping from pixel to fret number by some known scale. E.g., measure the pixel distance between nut and 12th fret in the image, compare to 12th-root-of-2 formula, adjust scaling. Bae's approach using fret markers then formula is one robust way. Because once you have 3rd fret marker and 5th fret marker pixel positions, you can align them to actual fret distances and get the rest by interpolation. This means the system can output actual fret indices even if it never explicitly sees fret lines.

Latency & Real-Time: If aiming for live transcription, the vision pipeline must run efficiently (MediaPipe can run ~30 FPS on CPU). It might also need a short buffer or lookahead to confidently label something (e.g., confirm a finger has settled on a fret for a few frames before declaring it a note position). For live feedback, that's a trade-off between accuracy and latency.

4.3 Fusion Approaches (Combining Audio and Video)

Finally, the audio and visual streams must be combined to produce the final transcription (notes with string assignments, and detection of techniques):

- **Late Fusion (Post-Decision Fusion):** In this approach, we independently get:
- A list of note events from audio (with timings and pitches, maybe confidences).
- A timeline of finger positions from video (e.g., for each moment, which fret each left-hand finger is on).

Then, we combine them by matching timings. For each note onset from audio, we look at video at that time (or slightly before, since finger usually is placed just before sound): - Find which left-hand finger is

down and where. If multiple fingers are down (like playing a chord), match multiple audio notes at that onset to those fingers. - Assign each audio note the string/fret of the corresponding finger.

If audio found a pitch but video shows no finger on a corresponding fret (maybe an open string or a missed detection), the system can infer that note is an open string (since no finger is pressing, open string must have rung). Conversely, if video shows a finger movement but audio had no note (e.g., ghost movement or error), we might ignore it or use it to correct audio (like if audio missed a note but video clearly shows a distinct finger pluck, maybe add a note).

This deterministic fusion works if alignment is precise. Paleari's system basically did this and achieved 89% disambiguation of notes. It essentially solves string assignment by look-up: "the note of pitch X that started at time t came from string Y because the finger on string Y fret Z was engaged at t".

- **Joint Model Fusion:** This is where a single model (like a neural network) ingests both modalities. For example, a Transformer could have token sequences from both audio and video: audio tokens might encode spectrogram frames, video tokens encode finger locations at those frames. The model then outputs a sequence of notes with finger labels. This approach can learn subtle correlations (e.g., maybe it can learn that a bend in audio (gliding pitch) correlates with a visible finger slide motion). Some experimental work uses *multimodal attention* to achieve this, but it's complex and data-hungry. So far, many audio-visual systems still lean on late fusion or simpler mid-level fusion because of limited training data.

- **Probabilistic Graphical Model Fusion:** Consider a Hidden Markov Model where the hidden state is (pitch, string) for each note. The audio gives likelihoods for pitch events; the video gives likelihoods for string/fret being pressed at time. One can formalize this as a Bayesian inference: P(string | video) * P(pitch | audio) -> P(string, pitch | both). If a note pitch can be produced on multiple strings, audio alone can't choose, but video gives a prior (like 90% chance finger was on string 3 vs 10% on string 4 at that time). Multiplying yields a posterior favoring one string strongly. This could be solved with a Viterbi algorithm if doing sequential decisions (e.g., impose that successive notes follow physical constraints). Dynamic Bayesian Networks can also incorporate *playability constraints* as prior knowledge (e.g., penalize unlikely string jumps) and use both audio/video evidence.

- **Constraint Satisfaction / Optimization:** Another angle is to formulate: *find the sequence of tablature (string assignments for each audio-detected note) that best fits the video observations and is physically playable*. This can be set up as an optimization problem:

- For each audio note at time t with pitch P, there is a set of possible string+fret options (those that produce P).
- The video at time t (plus/minus some delta) likely points to one of those options (maybe by score 0 or 1 if a finger is there).
- Add constraints like "if two notes overlap in time, they cannot occupy the same string unless one is open string ringing while finger moves" or "a finger can only be on one fret at a time".
- Then solve (e.g., with ILP – integer linear programming – or DFS search with costs).

This approach can guarantee consistent output (no physically impossible combos) and integrate soft evidence (video can give a cost: e.g., cost 0 if option matches video finger, cost high if not). It might need

to search a large space if many notes, but algorithms like A* or ILP solvers can manage moderate sequences (especially since polyphony is limited).

- **Handling Uncertainties:** A robust fusion should account for the confidence of each modality. For example, if the audio model is very certain about a pitch but the video is unclear (blurry), trust audio more and maybe pick the most plausible string from context. Conversely, if the audio is noisy (say the performer used heavy distortion and the pitch tracker is iffy, but the video clearly shows positions), lean on video.

- **Technique Fusion:** Some techniques need both modalities to identify. For instance, a **slide** is evident as a continuous pitch glide in audio *and* a continuous finger motion in video. Detecting slides could be done by a rule: if a note's pitch changes gradually (audio sees frequency ramp) and the same finger icon moves from one fret to another (video sees motion), mark that note as a slide. A bend would be audio: pitch goes up without new onset, video: finger stays on same fret but maybe increased tension (which might be subtle visually, sometimes you can see the string bend sideways). A combination of slight finger lateral motion plus audio pitch change could signal a bend. These are examples where neither audio nor video alone might be conclusive (audio alone can confuse a bend with a slide if it doesn't catch that the string didn't change; video alone might not detect a bend easily), but together they can confirm the technique.

- **Error Correction via Fusion:** If one modality has an error, the other can correct it. E.g., audio falsely detects an extra ghost note (maybe a percussive hit) – video might show no finger action or right-hand hit, so the system might discard that note or classify it as a percussive "dead note" (some tabs notate dead notes as "x", which could be deduced by audio onset + no clear pitch + video shows a strumming motion with no left-hand press). Conversely, video might "see" a finger press that didn't produce sound (perhaps the guitarist put a finger down but didn't pluck that string); audio confirms no note, so the system doesn't output anything (or outputs a tied note if it was already ringing).

Playability-aware Decoding: Incorporating guitarist ergonomics in fusion ensures the output tab is one a human could feasibly play: - Many systems incorporate a penalty for large hand jumps in decoding. E.g., if one note's fret assignment is far from the previous note's fret assignment, add cost unless video explicitly showed a jump. - Ensure that not more fingers are simultaneously required than the hand has. If audio gives 5-note chord and video only shows 4 fingers down, likely one note is open string (the system can assign one note to an open string if its pitch corresponds and that string wasn't fretted). - If a sequence is fast, it likely is played in one position, so favor staying on the same fretboard position for successive notes (this was the idea of Sayegh's DP path costs). - TART's approach actually trains a model to output simpler fingerings for novices, which is an interesting post-processing: they use an LSTM to adjust the tab for ease at cost of exact correspondence. In an architecture, this could be an optional "re-ranking" step: generate multiple tab candidates and score them on playability.

Diagrams & Pseudocode: Visually, one can imagine two parallel tracks (audio pipeline and video pipeline as discussed in 4.1, 4.2) merging into a fusion module. A pseudocode for late fusion might be:

```
audio_notes = AudioTranscription(audio_signal)
finger_positions = TrackFingers(video_frames)

result_tab = []
for note in audio_notes:
    t = note.onset_time
```

```
    pitch = note.pitch
    candidates = PossibleFingerings(pitch, tuning=E_standard)  # e.g.
[(string, fret), ...]
    # Check video around onset
    vid_info = finger_positions.get_state_at(t)
    # If vid_info has a finger on one of the candidate positions:
    if vid_info.matches_any(candidates):
        assigned = vid_info.match  # pick the matching candidate
    elif vid_info.no_finger_on_pitch:
        assigned = ("open_string", determine_open_string(pitch))
    else:
        # fallback: choose candidate that minimizes distance to a finger
position seen (maybe the closest finger)
        assigned = choose_min_distance(candidates, vid_info.positions)
    result_tab.append((note, assigned))
```

This pseudocode handles straightforward fusion. A more advanced approach would incorporate dynamic programming to enforce consistency across notes (especially if multiple notes occur at once or very close together).

Comparison of Fusion Strategies: Deterministic late fusion is easier to implement and interpret – you can trace why a note was assigned to a string (because finger X was there). Probabilistic or learned fusion could be more optimal in theory, but are harder to get right without lots of training data. For initial systems, a rule-based fusion with perhaps some scoring (like DP with hand position cost) is effective. For future SOTA, a neural network could potentially output tablature in one go from both inputs, but until there's enough audio-visual data to train that, hybrid pipelines remain popular.

In summary, the fusion architecture ensures that the audio's strengths (timing, pitch) and video's strengths (which string/fret) come together. It resolves ambiguity, adds technique annotations, and filters out mistakes. A well-designed fusion is what elevates a system from "just notes" to a true transcription with tablature that a guitarist can use.

Open Problems & Research Gaps

Despite progress, fully automatic guitar transcription (especially in general, real-world scenarios) still faces many unsolved challenges. Here are some open problems and areas ripe for further research:

- **Fast Passages with Occlusion:** When a guitarist plays very fast (e.g., shredding a solo) and/or uses wide finger motions, the video can become a blur and fingers occlude each other frequently. Current vision systems struggle to track fingers reliably at high speed, and audio models might merge fast note onsets. Handling **fast, technical solos** (16th notes at 200 bpm, etc.) is an open challenge. Approaches might include high-frame-rate cameras, more sophisticated predictive tracking, or specialized audio models that can separate fast notes (like using higher sampling rate features to capture quick transients). Real-time needs here clash with the detail needed.

- **Extended Techniques (Slaps, Harmonics, Percussive effects):** Guitarists use techniques like **slap and pop** (esp. on bass, but also acoustic percussive hits), **natural or artificial harmonics** (touching string lightly to get flute-like tones), **pinch harmonics** (pick technique producing a

squeal), **feedback sustains**, etc. These produce very different audio signatures (percussive hits may have no clear pitch; harmonics have very high pitches or pure tone; feedback is a sustained howl) that confuse standard transcription models. Detecting these and notating them correctly (e.g., with symbols or text like "Harm.") remains tough. Datasets for such techniques are limited (though Guitar-TECHS and others start to cover them). Particularly, deciding whether a sound is a note or a percussive hit is tricky – sometimes a slap doesn't have a pitch and should be notated as a rhythmic "X" in tab. No mainstream model handles that robustly yet.

- **Distortion and Pitch Masking:** Electric guitar with heavy **distortion** compresses the dynamic range and adds harmonics, making notes blend together. Distortion can cause a phenomenon where the fundamental frequency is less prominent than higher harmonics (e.g., a power chord's perfect fifth might dominate over the root's fundamental). Transcription models trained on clean or mild overdrive often falter on high-gain metal tones. Research by Pedroza et al. (2024) tried to address this by including such tones in training, but generally, *robustness to any tone/effects* (fuzz, wah-wah, pitch shifters, etc.) is an open area. Possibly using multi-track inputs (if available, like separate pickup signals) or incorporating models of the effect in the transcription process could help.

- **Generality Across Guitars & Tunings:** Guitars come in many varieties: 7-string or 8-string guitars (common in metal), alternative tunings (Open D, DADGAD, etc., common in folk and rock), capo usage, bass guitars (4 or 5 string) which are similar but lower range, even other fretted string instruments (ukulele, etc.). A truly general system would detect the instrument's tuning automatically (for example, by analyzing the open string ring frequencies or looking at fretboard markers positions which differ e.g. bass often has dots differently after 12th fret). Current research mostly assumes standard tuning 6-string. *Tuning detection* itself is a sub-problem – one can imagine a model scanning the audio for likely open-string resonances or using video to see if dots at certain frets align with known tuning patterns. Capo detection is also tricky – a capo effectively shifts all notes; a vision system might literally see a capo device on the neck (and could deduce its fret position), or audio could notice all notes are shifted by a fixed interval. Adapting transcription to arbitrary tunings (and then reflecting that in notation properly) is not fully solved. Most systems require the tuning be given or stick to standard.

- **Chord Voicing Ambiguity:** Even if you get the correct pitches for a chord, deciding *which voicing/inversion* the guitarist used is hard. For example, a simple E minor chord (E–G–B pitches) can be played in multiple locations and string sets. Audio alone cannot tell the difference; video can if it clearly sees the fingers, but if one finger obscures others, it might miss one string's status (some chords only use 2 fingers but involve 4 strings, meaning some open strings – if video tracking only keys on fingers, it could forget the open strings). This ambiguity extends to **barre chords vs open chords** that yield same pitch sets. Resolving it sometimes requires contextual knowledge (e.g., position before/after, or common guitar practices – a human transcriber uses knowledge that certain chord transitions are easier in particular voicings). No AI yet encapsulates all that expert knowledge. This is an open area where perhaps integrating a knowledge base of chord fingerings or using sequence models that learn common chord progressions on guitar can help.

- **Right-Hand Technique & Articulation Inference:** As mentioned, figuring out what the picking hand is doing is largely unaddressed. That includes identifying whether notes are finger-picked or flat-picked, detection of rasgueado (flamenco strum), identifying strumming patterns (which strings are hit in a chord strum and whether it's up or down stroke). These are important for faithfully reproducing style but are very subtle in audio (up vs downstroke might change the order of string attack slightly, which one could detect with high-res onset detection) and video (the motion of the hand). There's also **palm muting**, often done with the right hand resting on

the bridge – audio hears a shorter, muffled note, which can be detected by envelope shape, but video might see the palm on strings (if camera angle shows it). Some works classify palm-mutes from audio, but in a piece, detecting exactly which notes are palm-muted is tricky, especially under distortion where every note is somewhat compressed. This remains a gap: integrating right-hand video, or advanced audio features, to mark notes as muted or not. Similarly **fingerstyle patterns** (like Travis picking) – identifying which finger plucked a note (p, i, m, a designation in classical guitar) could be valuable for educational feedback, but it's essentially unresearched due to lack of data and difficulty (maybe high-speed cameras on the right hand could catch finger movements).

- **Polyphonic Separation and Multiple Guitars:** If there are multiple guitars playing (e.g., duet) or guitar + other instruments, separating the target guitar's notes from the mixture is an open challenge. It crosses into source separation: one might need to separate the audio into stems first. That's a whole MIR field of its own, and currently transcription often assumes isolated or dominant solo guitar. An advanced system could attempt to do separation and transcription jointly (some recent MIR papers do multi-instrument transcription by also identifying instrument sources). In video, if two guitars are visible, identifying which sound belongs to which (audio-visual association) is an additional complexity.

- **Expressive Timing and Tempo Drift:** Guitarists (especially in solo performances) don't stick to a strict metronomic tempo – they speed up or slow down (ritardando, rubato). Aligning a transcription to a metrical grid (bars and beats) under tempo drift is hard. While not strictly needed for tab (tab can be time-based rather than beat-based), for standard notation it's needed. So, automatically detecting tempo changes and beat positions (MIR's beat tracking) and aligning the transcription to a score grid is still imperfect. Combining beat tracking with transcription could help (transcription gives note onsets, then beat tracking can align those to a tempo map), but doing it robustly in one system is not solved.

- **Synchronizing Multi-Modal Data:** Collecting richly annotated audio-visual data itself is challenging, and any slight sync mismatch can degrade a model. Methods for automatic fine synchronization of video and audio (like aligning a video of guitar playing to a known audio, as Riley did using an alignment model) are improving. But if a generic user video's audio is slightly out of sync, the system might assign notes to wrong frames. Thus, either building models robust to sync errors or having a quick sync calibration step (e.g., detect a percussive strum onset in audio and the corresponding motion in video) is something to consider. This is both an engineering problem and a research one for multimodal coherence.

- **Generalization to Complex Genres:** Most research so far deals with fairly "clean" scenarios: a single guitar, well-recorded. But consider techniques like **tapping (both hands on fretboard)** as in some rock/metal – here the right hand comes into fretboard, potentially confusing a left-hand tracker. Or **using a slide bar** (which changes the shape of playing – continuous slides and no distinct frets). Or **scordatura** (irregular tunings including microtonal adjustments). These are edge cases but matter for covering all music. The current SOTA doesn't cover them well. A truly comprehensive system would either detect these scenarios or allow user input (like "mode: slide guitar") to adjust processing (for slide guitar, maybe treat every note as potentially a continuous slide and look for different visual cues).

- **User Interaction and Feedback Loop:** As a design note, often fully automatic transcription might never reach 100% for all music. Having a way for the user to correct or guide the system (e.g., user taps a fret in the interface if system got string wrong, and system re-adjusts subsequent analysis) could greatly improve practicality. However, how to incorporate such

feedback dynamically is an open design problem. Research could explore interactive machine learning for MIR.

In essence, while we can now fairly reliably get “notes on a guitar” in constrained settings, achieving a **truly universal guitar transcription** system (any player, any style, any environment, real-time) has many unsolved pieces. Each of these open problems is an active research opportunity – for instance, robust technique detection competitions, or datasets focused on heavy distortion transcription, or integration of symbolic music knowledge for better chord fingering decisions. Solving them will move the field from impressive demos to an everyday tool for musicians.

Engineering Roadmap

Building a full audio-visual guitar transcription system is an ambitious project. A stepwise approach – from a basic prototype to a robust production system – helps manage complexity. We outline a phased roadmap with increasing scope and capabilities:

6.1 MVP (Minimum Viable Prototype)

Scope: A simplified scenario to prove core functionality. Assume: - Single fixed camera, clear view of fretboard. - High-quality, *clean* audio (e.g. direct input or mic in quiet room). - Standard tuning, 6-string guitar. - One guitarist, playing relatively simple material (e.g., monophonic melody or slow polyphonic pieces). - Focus on combining audio+video for basic note identification in tab form.

Architecture & Model Choices: - **Audio:** Use a pre-trained model like Onsets & Frames (fine-tuned on GuitarSet) to detect notes. For MVP, monophonic notes could even be detected via a simpler method like CREPE + onset detection (since initial scope is simple polyphony or single-line). Onsets & Frames can give you pitch and onset fairly reliably for one guitar. - **Video:** Use MediaPipe Hands to track left-hand fingers. Simplify by perhaps marking the fretboard manually so you know region of interest. Since simple music, one could even assume if one finger is down, that corresponds to the note (no complex chords initially). - **Fusion:** Late fusion approach. For each detected note from audio, assign it to the string nearest the fingertip detected at that onset time. For MVP, if only one note at a time, this is straightforward: identify which finger is down and on what string – that’s your string. If no finger is down at onset (MediaPipe might lag slightly or perhaps it was an open string), classify it as open string. - **Output:** Generate a basic tablature output. For MVP, ASCII tab could suffice (e.g., print out something like "E|----0---", etc.). Or just a textual list of note -> string.fret.

Integration and UI: Possibly a simple GUI that displays the video with detected finger positions (for debugging) and prints recognized notes. But MVP could also be offline: record a short video+audio, run it through the pipeline, output tab text.

Training Strategy: MVP might not require training from scratch. Use off-the-shelf models: e.g., Onsets&Frames model (pretrained on piano or multi-instrument) fine-tuned on a small guitar dataset (like the open GuitarSet) – this fine-tuning can be done beforehand. MediaPipe is pre-trained (no training needed for vision). So mostly it's about calibrating the mapping from MediaPipe coordinates to strings/frets (which can be done via a one-time calibration: e.g., have the guitarist play a known note and use its audio pitch to label which fret that corresponds to in video to scale the geometry).

Evaluation Metrics (MVP): Because scope is limited, you can manually verify a few test recordings. Metrics: *note accuracy* (did it get the right note pitch?), *string assignment accuracy* (for each note, correct

string?), for single notes this is trivial (pitch implies a correct string if correct, except in redundant cases). Since only one guitarist and known conditions, we expect near 90%+ correct for simple pieces.

Risks (MVP): Even MVP can stumble if sync is off by a few frames. Or if MediaPipe loses track when hand goes off camera. To mitigate, constrain test: e.g., guitarist plays in first 5 frets only in view. Another risk: Onsets&Frames might output multiple octaves for one note (false double), which could confuse mapping. We handle by taking highest-confidence pitch at onset. Computationally, MVP can run on a normal PC CPU/GPU (MediaPipe on CPU ~10% and Onsets&Frames on CPU can be slow but for short recordings it's fine; on GPU it’s realtime).

Outcome: MVP demonstrates that “we can get a tab for Twinkle Twinkle Little Star played on guitar” for example, aligning each note to correct string using video.

6.2 Mid-Stage Prototype (More Complex Music & Conditions)

Scope Expansion: - Handle moderate polyphony (e.g., two-note intervals, simple chords). - Include **fingerstyle acoustic** pieces where multiple strings plucked. - Introduce some **occlusion**: faster runs or more hand movement, possibly partial occlusion by hand. - Support common **altered tunings** (maybe drop-D) by detection or user input. - Audio may include minor noise or reverb (still not heavy distortion or band mix). - Possibly allow a simple **capo** case (like if capo on 2nd fret, user can inform system).

Upgrades in Architecture: - **Audio:** Move to a more robust model (maybe the high-resolution CNN or a Transformer) that can handle chords. Onsets & Frames can actually do polyphonic, so continue with that but ensure it's fine-tuned on data including chords (like include EGDB data for chords or GuitarSet chords). Add a *chord onset detector* or refine onset grouping: if multiple pitches have very close onset times, treat them as a chord event. - **Video:** For chords, multiple fingers will be down. The system needs to detect all relevant fingers. MediaPipe gives all 5 fingers; use those joint positions to figure which strings are pressed simultaneously. E.g., at time t, find all fingers whose tip is near the fretboard. Determine their string positions. This yields a set of pressed strings at particular frets. The challenge: sometimes a finger might barre across multiple strings (index finger covering a whole fret). MediaPipe might show index fingertip only, but it covers several strings – a limitation. We might need logic: if index finger is nearly horizontal and covers say fret 2, perhaps assume it bars and thus strings 1-5 at fret 2 are pressed. Recognizing a barre from video alone is non-trivial but perhaps the finger orientation/ curvature can hint (straight line across fret as opposed to curved finger). - **Fusion:** Now more complex. When audio says a chord of pitches [E, G#, B] occurred, video might show index on 2nd fret 5th string, ring on 4th fret 4th string, pinky on 4th fret 3rd string (that would be an E major chord shape). The fusion algorithm should match each audio pitch to a finger such that the pitch matches the fret of that finger. This is a matching problem – which pitch corresponds to which detected finger – solvable by trying all assignments (which is small since maybe 3 fingers and 3 pitches) or by known mapping (the fret positions directly give target pitches if tuning is known, we could forward calculate what pitch each detected finger produces and match to audio). For drop tuning: need correct tuning info to do this match properly; if not auto-detectable, might allow user to set tuning in mid-stage. - We should incorporate a basic **playability constraint** now: e.g., if audio gives a combination that video finds impossible (finger positions don’t match pitches), perhaps audio misdetected a pitch – we could drop or adjust it (maybe an overtone mistaken as a note). - Possibly start integrating a *string assignment model* like a lightweight version of Fretting-Transformer or DP as a backup. For example, after initial fusion, run a DP smoothing on the sequence of string assignments to avoid big position jumps that video didn’t corroborate.

New Features: - **Occlusion Mitigation:** If a finger gets occluded, mid-stage might include logic like: predict finger position by interpolation. Or use both **left and right hand** info: if right-hand plucks a string and left-hand finger not seen, maybe it was an open string (somewhat guess). - **Technique handling:** Introduce detection of slides and bends. E.g., if audio has continuous pitch glide and video sees a finger moving between frets, label that as a slide (and mark in tab as such). If audio pitch goes up but finger stays on same fret (perhaps slight lateral motion), label as bend. Also detect **hammer-ons/ pull-offs**: these are tricky but if two notes in audio are legato (no pluck in between, right-hand stationary, left-hand fingers changed), video might catch that a new finger came down or lifted without a right-hand action. Start marking these (e.g., with "h" or "p" in tab). - **Multi-tuning:** To support drop-D, allow the system to attempt figuring out tuning by analyzing bass string pitch (if an open low string is plucked early on, and audio finds it's D2 not E2, system registers drop-D). Or user can select from presets.

Evaluation & Metrics (Mid-stage): Now we measure things like *multi-note transcription F1* (notewise precision/recall including chords), *chord identification rate* (correct set of pitches for chords), and *tab accuracy* which includes correct string assignment. Also *technique classification accuracy* for those we started tagging (slides, etc.). Use a slightly larger test set (perhaps parts of GuitarSet or a small set of real recordings with known ground truth tab). Aim for note-level F1 > 0.8, tab accuracy maybe ~0.7 for chords (some errors allowed), technique detection maybe lower (~50-60% if just starting out).

Risks & Mitigations: - Occlusion might cause missing notes (if a finger covers another, the second finger's press might not be detected and system might think an open string rung instead). Could mitigate by focusing on audio for number of notes and using video as guide for which ones fretted. - False positive notes from audio (esp with slight noise or resonance) could lead to ghost notes in tab; consider filtering low-confidence audio notes unless a corresponding finger is found. - Increased complexity might hurt real-time performance: ensure computational steps still manageable. Perhaps run video processing every frame (30fps) but audio model in small batches (like every 100ms). - Integration testing on various songs to find failure patterns (maybe the system doesn't know how to handle fast arpeggiated chords – those appear as staggered onsets which might confuse chord grouping logic).

6.3 Production-Scale System

Scope: "In the wild" usage – robust to virtually any solo guitar performance scenario: - Multiple camera angles or just a phone camera that might not be perfectly positioned. - Possibly support **multi-angle input** if available (e.g., user can use two phone cameras or a dual camera rig). - Works in **noisy environments** (some background noise or other instruments quietly present). - **Real-time inference** or near-real-time (low latency) so it can be used as a practice tool. - Can handle both **acoustic and electric**, with effects (distortion, etc.). - Handles **virtually all techniques** (bends, slides, tapping, harmonics, muting, tremolo picking). - Possibly extend to **bass guitar** mode or **other string instruments** with minor tweaks.

Architecture Enhancements: - **Audio Model:** Possibly use a **streaming transcription model** (like a causal or frame-by-frame model) for low latency. For instance, a Transformer model can be chunked or an RNN-based model can run continuously. Might use a combination: an onset detector that's very quick (to note new notes) and a slower refinement network for pitch if needed. Also, integrate a **tonal analysis** that can adapt to distortion – e.g., a model or mode specifically for distorted guitars, or use a front-end that separates the distorted guitar from noise (maybe a small source separation preprocessor). - **Vision Model:** Add **robustness** features: if only one camera but angle is weird (say from the right side instead of directly in front), maybe an initial calibration step where the user shows the

guitar to the camera and the system finds it. Possibly incorporate a trained model to detect fretboard keypoints in arbitrary orientations (this could be done with a small CNN trained on synthetic images of guitars). Use multi-angle if available: e.g., combine MediaPipe from both angles and merge results (some sensor fusion). - For occlusion, consider using a **depth camera** or **stereo** if possible, or ultra-wideband sensors (experimental, but likely beyond scope – stick to optical). - **Multimodal Model:** At production scale, maybe introduce a learned fusion model (like a multimodal Transformer) to refine outputs. For example, after initial late fusion, a Transformer could take the draft transcription plus raw inputs to correct errors (like a second-pass that says "given audio+video and an initial tab guess, output the final tab"). This could catch subtle mistakes by learning from large training data (requires such data). - **Scalability & Performance:** Optimize each component: quantize the audio model for speed, use GPU for heavy tasks if available. Possibly split tasks into threads: one doing vision, one doing audio, then synchronizing results. If real-time, maintain a short rolling buffer (maybe process audio/video in 0.5s chunks). - **Playability & Formatting:** Integrate a dedicated *tab formatting module* that also checks playability. For example, apply a final optimization to minimize hand jumps (like TART's LSTM approach to simplify for beginners). Possibly allow user to specify difficulty or skill level and adjust tab (e.g., for a beginner, choose an easier fingering even if it deviates slightly from original positions). - **User Interface:** Provide an interactive UI where the transcription is displayed in standard notation + tab (perhaps using an embedded MuseScore engine or custom canvas). The user can play live and see notes pop up in near real-time (like a scrolling tab). They can also start/stop and export results to PDF or GuitarPro. Include controls for tuning, instrument type (6-string, 7-string, bass 4-string), and calibration (maybe strum open strings so system calibrates tuning and audio levels). - **Export Polished Output:** Use MusicXML or directly integrate LilyPond for a nicely typeset score. Possibly auto-add performance directions like "Capo 2" if detected, or "Tune low E to D" if drop tuning found. Provide GuitarPro file export (since many guitarists use GP).

Training & Data: - Train on a **much larger dataset** combining everything: GuitarSet, EGDB, GAPS, Guitar-TECHS, GOAT, etc. Possibly do semi-supervised learning by generating synthetic training examples: e.g., take a clean audio and apply random effects to augment, or simulate different camera angles on video by 3D transforms of existing labeled videos (if any). - Incorporate **transfer learning**: for vision, use models pre-trained on hand pose (like MediaPipe's backbone) and fine-tune on any available labeled guitar images (maybe synthetic labeling: generate images with CGI hand and guitar?). - Continuously improve with user feedback: allow the app to collect anonymized corrections (if user edits the tab afterward, those could be fed back into training to improve the model).

Evaluation: - Extensive tests on real music pieces covering various genres: classical guitar (nylon acoustic), jazz (with complex chords), metal (distorted fast riffs), etc. Use reference transcriptions (perhaps from published scores or expert transcribers) to compute precision/recall, but also do **user studies** – have guitarists rate the quality of transcription (since a purely numerical score might not capture if something is playable but different position). - Benchmark the latency: ensure that if aiming real-time, the system outputs notes within e.g. < 200ms of them being played, so the delay is not noticeable for practice scenarios.

Risks & Mitigations (Production): - **Generalization:** The model might still fail on extremely novel input (like a guitarist using bow on guitar, or prepared guitar with objects on strings). These corner cases are rare; communicate clearly the expected use cases and maybe detect when out-of-distribution (if audio looks nothing like guitar, system might indicate uncertainty). - **Misidentification:** A big risk is wrong string assignments in complex chords that the system can't fully observe. Mitigation: perhaps highlight notes that could be on multiple strings and allow user to toggle if it sounds off. - **Performance on older hardware:** Real-time processing can be heavy; perhaps allow a "record then transcribe" mode on slower devices, whereas high-end PCs can do on-the-fly. Optimize by using C++ for heavy parts or using hardware acceleration (on mobile, use GPU or DSP). - **User Experience:** If the output has many errors,

users will lose trust. It’s crucial at this stage to minimize obvious mistakes. Possibly constrain outputs: for instance, if audio detection is uncertain between two pitches, maybe don’t output either or output the more likely one but mark it uncertain. Or if the system isn't confident, it could ask the user (like “Was that chord X or Y?"). However, an automated system asking might frustrate users; better to do best guess with a confidence indication.

Real-time Feasibility: Achieving true real-time is tough but not impossible. MediaPipe runs in real-time on phones. A lightweight audio model (like a smaller CNN or RNN) can run in a few milliseconds per frame on CPU. The fusion step is negligible in compute. The bottleneck might be the deep model if it's large – a Transformer might be too slow, so might use a CNN or RNN for live use. Alternatively, use a two-tier approach: a real-time simplistic model to show something live, and then a refined transcription done after stopping recording using the heavier model (so user gets instant feedback and then a finalized accurate transcription after).

Outcome: The production-scale system would allow a guitarist to record themselves (or even live-process as they play) and get a reasonably accurate sheet music/tab output, needing minimal manual correction. It would be robust enough for most styles and could be packaged as an app or software. This is the stage where the system could be released to users or used as a basis for further research (e.g., as a tool to generate labeled data from unlabeled videos by partially transcribing them).

This roadmap illustrates incremental addition of complexity: first nail the basic note finding, then chords and common techniques, then broaden to any situation and optimize performance. At each stage, testing and feedback guide the improvements for the next phase.

Learning Guide (Full Curriculum to Understand Everything Above)

To undertake this project or enter the field of audio-visual guitar transcription, one needs a strong foundation in several areas: music theory, signal processing, machine learning, computer vision, and more. Here is a structured curriculum, divided by levels, to build up all the required knowledge and skills:

7.1 Foundations (Prerequisites)

Music Theory Basics: - *Intervals & Scales:* Learn what intervals are (e.g., major third, perfect fifth) and how they form scales. Understand diatonic scales, chromatic scale. - *Chords & Voicings:* Study how chords are constructed (triads, seventh chords) and what voicing means (different arrangements of the same chord tones). Specifically for guitar, learn common chord shapes and inversions. - *Guitar-Specific Theory:* Know the standard tuning (E A D G B e), how notes repeat on the fretboard, the concept of positions. Learn how the same pitch can appear on multiple strings (e.g., middle C can be played on 5th string 3rd fret or 4th string 8th fret). Understand basic guitar notation and tablature, and techniques like bends, slides, hammer-ons. - *Rhythm & Notation:* Be comfortable with rhythmic values (quarter, eighth, triplets, etc.), time signatures, and how these appear in standard notation and tab (stems on tab numbers, etc.). Practice by looking at simple sheet music and clapping the rhythms.

Digital Signal Processing (DSP) Fundamentals: - *Sampling Theory:* Understand how continuous sound is sampled to digital. Concepts of sample rate (44.1kHz, etc.), Nyquist frequency, aliasing. - *Fourier Transform:* Learn the basics of Fourier series and transform; how to interpret a frequency spectrum.

Specifically, go through the Short-Time Fourier Transform (STFT) to see how we get a spectrogram from audio. - *Spectrograms & Time-Frequency:* Know what a spectrogram is (magnitude over time-frequency plane). Understand resolution trade-offs (window size vs time precision). Learn about mel-scaled spectrograms and constant-Q transform (which is essentially a log-spaced spectrogram more aligned to musical pitch). - *Harmonic structure:* Understand that musical notes produce harmonics (overtones) and how these appear in spectra. Recognize patterns like the fundamental and its integer multiples. This is crucial for understanding multi-pitch detection (e.g., distinguishing two notes' harmonics interleaving). - *Onsets & Offsets:* Learn methods for onset detection (like measuring sudden increases in energy or using spectral flux). Perhaps implement a simple onset detector on a waveform (e.g., differentiate energy envelope and threshold). Similarly, understand offset detection (harder, often done via noting decay or silence after note). - *Basic filters:* (optional but helpful) Understand how high-pass, low-pass filters work; could be relevant in cleaning audio or isolating frequency bands (for example, isolating bass string frequencies vs treble).

Machine Learning Foundations: - *Probability & Classification:* Understand what classification is vs regression. Get comfortable with concepts of training data, loss functions (like cross-entropy for classification). - *Deep Learning Basics:* Study neural networks (perceptron, multi-layer perceptron) and then CNNs (convolutional neural nets, which are key for image and spectrogram processing) and RNNs/LSTMs (key for sequence modeling like music over time). Then learn Transformers (attention mechanism, sequence-to-sequence modeling), since latest models use them for transcription. - *Sequence Modeling:* Specific to our task, understand how a sequence model can represent music. For example, an LSTM generating a sequence of notes one by one, or a Transformer treating transcription as language (with an output "vocabulary" of note events). - *Loss Functions for sequences:* Learn about frame-level vs note-level loss. E.g., frame-wise binary cross-entropy for each pitch class at each time frame vs specialized losses that only penalize onsets. Also, CTC (Connectionist Temporal Classification) loss is worth knowing, as it's used in some transcription to align output with uncertain timing. - *Evaluation Metrics for AMT:* Get familiar with precision, recall, F1. Specifically, frame-wise metrics (what percentage of frames got correct pitches) vs note-wise metrics (what percentage of notes in ground truth were correctly found by model within some tolerance). Understand the concept of onset tolerance (like 50ms window for matching an onset). Also metrics for multi-pitch like multipitch detection accuracy (combination of precision/recall for frames). - *Overfitting & Generalization:* Basic ML concept – ensure understanding of train/val/test splits, cross-validation. For MIR tasks, knowledge of how overfitting might occur if same song or same performer appears in train and test (since model could latch on particular timbre).

Programming & Frameworks: - *Python Scientific Stack:* Be proficient with NumPy (for array computations), SciPy (for signal processing utilities like FFT). Practice by manipulating audio signals (e.g., use SciPy to load a WAV, compute its FFT). - *Librosa or Torchaudio:* Use librosa to do tasks like load audio, compute spectrograms, detect onsets. A good exercise: take an audio of a scale played on guitar and use librosa to find the onset times and pitches (librosa.pitch_yin can attempt monophonic pitch tracking). - *PyTorch/TensorFlow:* Choose one and get comfortable building and training models. A simple CNN on spectrogram for note classification (maybe classify which note is being played from a short clip) is a good start. Also learn how to use GPUs and manage data pipelines. - *OpenCV:* Practice basic image processing relevant to guitar video: read a video frame, convert to HSV, maybe try to detect edges (Canny) and see if you can find lines that correspond to frets or strings in a sample image. Familiarize with geometric transforms (cv2.getPerspectiveTransform). Possibly, use OpenCV's findChessboard in a fun way if you have something grid-like as fretboard (or place markers to test perspective warp). - *MediaPipe usage:* Install MediaPipe and run their hand tracking demo. Then try customizing it – feed a guitar playing video and see if it detects the hand, examine the landmarks indexing (which landmark corresponds to index fingertip, etc.). This gets you used to extracting pose data.

By covering these fundamentals, you will be equipped with the basic language and tools of the domain. This level ensures you understand how a guitar produces sound, how sound is represented in data, and the basics of how a computer could learn patterns from that data.

7.2 Intermediate Topics (AMT & CV Core Skills)

Building on the basics, these topics get into the core of automatic music transcription and computer vision as needed:

Audio Modelling & MIR Core: - *Polyphonic Pitch Estimation:* Study classic MIR papers on multi-pitch detection. For instance, papers by Klapuri (2006) on multiple F0 estimation, or more recent deep learning approaches (like Bittner's "deep salience" model). Try implementing a simple multipitch: perhaps using Harmonic Product Spectrum (old technique) to see how multiple pitches might be inferred. - *Timbral Features:* Understand features like MFCCs (Mel-Frequency Cepstral Coefficients) which capture timbre (these are common in instrument identification, not directly transcription, but good to know). Also spectral centroid, roll-off – as these sometimes integrate into onset detection or brightness which might link to technique (a palm mute has different spectral centroid than an open note). - *Self-Supervised Audio Embeddings:* Modern approaches might use embeddings from models like Wav2Vec, or the "MuLan" (music audio language) models. Knowing that such representations exist can help – e.g., a pre-trained model that captures general audio could be fine-tuned for our task. Possibly look at OpenAI's CLAP or similar multimodal embeddings for audio. - *Audio Augmentation Techniques:* To train robust models, one must augment data: learn about adding noise, random EQ (to simulate different amp sounds), time-stretch, pitch-shift (pitch shifting guitar audio essentially simulates tuning changes or different playing positions). Also, mixup (mixing two audio samples) although for transcription that's tricky as it creates polyphony that might not have labels. Knowing augmentation helps expand limited datasets. - *Piano Transcription and other instruments:* Study how others solved piano transcription (as it's the closest parallel). Also see multi-instrument transcription efforts (like multi-f0 tracking in ensemble music), which might give insight into separating guitar from other sounds.

Computer Vision Foundations: - *Camera Geometry:* Learn about perspective projection, focal length, etc. Possibly go through a tutorial on how to calibrate a camera with OpenCV (using a checkerboard) – while not directly used in our system, it builds understanding of how 3D maps to 2D. Understand homography (projective transform) since fretboard mapping is a homography problem if we assume the fretboard is planar. - *Object Detection:* Understand how models like YOLO or RCNN work (just conceptually): input image -> bounding boxes of objects. Perhaps follow a tutorial to detect something simple (maybe train YOLO on images of a guitar neck vs background, to detect guitar neck). This could later help if we want custom detectors. - *Pose Estimation & Tracking:* Already touched via MediaPipe, but now dig deeper: read how MediaPipe Hand works (it uses a palm detector then landmark model). Also look at OpenPose method for hand (which is part of a bigger model). If possible, experiment with OpenPose on images to see difference vs MediaPipe. Also learn about optical flow (Farneback's method or Lucas-Kanade) to track movement between frames – could be useful for tracking fingers between discrete pose outputs. - *Temporal CV:* Understand the basics of how to track objects across frames, data association problem (if you had to track multiple fingers, how to keep their identities consistent). Basic knowledge of Kalman filters or even simpler exponential smoothing for trajectory. - *CV for Hands & Fingers:* Might read a specific paper or two on "vision-based guitar fingering detection" (like the Mark Asmar thesis or older CV papers) for insight into approaches (some used colored gloves or high-contrast markers, etc., which is fine to know what's been tried).

Music Information Retrieval (MIR): - *Beat & Tempo Tracking:* Learn how algorithms detect beat (onset detection + autocorrelation or dynamic programming). Try a beat tracker on a song. This ties in if you

later want to align notes to measures. - *Key & Chord Estimation:* Although our system doesn't strictly need to name chords (just tab them), understanding chord recognition might help, especially if we want to validate transcription (e.g., if our tab says a chord and a separate chord recognition says some chord name, they should match; if not, maybe our transcription missed a note). Learn basic chord recognition from audio (often uses chroma features and pattern matching). - *Music Structure:* Identify repeating sections or riffs. Not directly needed for transcription, but if one wanted to optimize output (maybe refrain from transcribing identical bars twice, just indicate repeat), structure analysis is relevant. Awareness of algorithms for structure (like similarity matrices) could be interesting for a later stage. - *Symbolic MIR vs Audio MIR:* Understand what can be done once you have symbolic data. E.g., computing difficulty of a piece, detecting scale usage, etc. For our scope, symbolic knowledge can feed back: if transcription yields something that's theoretically very odd (like an impossible chord or out-of-key accidentals everywhere), it might hint the model is off. Knowledge in symbolic domain (like common patterns) could be used to refine transcription (like language modeling does for text). - *Pitch Spelling:* On guitar tab, we usually don't care if a note is C# or Db, but in sheet music we do (it depends on key context). Learning how to do pitch spelling (deciding enharmonic equivalents) is part of symbolic music processing. It's advanced but good for final polish to have contextually correct notation.

At the end of Intermediate, you should be able to, for example: - Implement a basic polyphonic transcription system (maybe not state-of-art but functional) that uses a CNN or something on spectrogram. - Use a pose model to track hand and map points to a simple fretboard coordinate. - Understand a research paper in ISMIR or ICASSP in this area (the terminology and methods). - You might have also built small projects: e.g., a program that given a clear photo of a guitar neck with fingers will output which frets have fingers (a static case), or a program that given an isolated guitar audio will output a sequence of MIDI notes (with some errors likely, but conceptually working).

7.3 Advanced Topics (Specialized Knowledge)

Time to delve into the specialization of guitar transcription and multimodal learning:

Guitar-Specific Machine Learning: - *String/Fret Disambiguation Models:* Study Sayegh's DP algorithm (1989) to understand the classical formulation. Then read modern ones: e.g., "Fretting-Transformer" (Hamberger 2025) and "MIDI-to-tab by masked LM" (Edwards 2024). Understand how they encode the problem (what is the input representation and output). - *Playability Models:* Look at research on modeling guitar fingering as a cost optimization. There's also work by Yuchi Zhang (2019) on "Guitar fingerings as continuous optimization" etc. The goal is to see how physical constraints and difficulty metrics can be quantified. - *MIDI-to-Tab Decoding:* If possible, get the code or pseudo-code for a tab inference model and play with it (for instance, give it a MIDI and see how it assigns strings). This will deepen understanding of common errors (like it might choose non-intuitive fingerings if not constrained). - *Alignment for Guitar:* Learn about aligning a score to audio specifically for guitar. For instance, Xavier Riley's work aligning given transcriptions to audio. This includes dynamic time warping or using an HMM where audio frames align to score events. It's advanced but might be useful if one wants to use an existing tab to supervise audio model training (alignment task). - *Hexaphonic Audio:* Know what hexaphonic pickups are and how they simplify transcription (each string isolated – if only we always had that!). Not widely available, but some research uses it (like GuitarSet). Just be aware that some methods assume separated string signals.

Audio-Video Fusion & Multimodal DL: - *Multimodal Attention:* Read a bit on how Transformers handle multimodal (e.g., "Audio-Visual Transformers" in some literature). Also, looking at work in video action recognition with audio (e.g., looking at someone playing piano, match keys pressed to sound). - *Synchronization methods:* If you had to align video and audio that start at different times, research

methods of A/V sync detection. For guitars, maybe detect the moment of first strum in both audio and video (peak sound, big motion). There are algorithms in video coding for A/V sync using cross-correlation of audio with the audio track from video. - *Temporal Fusion Architecture:* Dive into how to merge asynchronous streams. For example, if audio is 44kHz and video 30fps, how do you architect a network that processes those? Solutions include: upsample video info to audio rate (repeat pose data for all audio frames in that interval), or downsample audio features to video fps (like 1 feature per video frame summarizing audio). Understand pros/cons. - *Cross-Modal Training:* A concept: using one modality's strength to supervise the other. E.g., using video-estimated fret positions to help train an audio model to predict string (some kind of cross-modal distillation). Think about creative training setups that use unlabeled data by exploiting audio-video consistency (e.g., as in "synchronize audio and video by ensuring network embeddings match" approaches).

Symbolic Music Modelling: - *MusicXML & MEI deep dive:* Understand the structure of these formats, how guitars are represented (MusicXML has `<staff>` for tab vs notation, `<technical>` tags for string and fret). Maybe practice by manually coding a small MusicXML file with some notes on tab and opening in MuseScore to see if it appears right. - *LilyPond for Guitar:* LilyPond has a section in docs on notating guitar, including how to do tab and notation together, how to indicate fingering, etc. Skim that to know what's possible (like microtonal bends or showing rhythms on tab). - *Score Engraving Rules:* Learn a bit of what human engravers do: e.g., how to break beams in a way that reflects beat, how to handle tuplets, how to layout multiple voices on a staff. This could inform how you present the final output (though tools do a lot automatically). - *Algorithms for Layout:* In case one wants to build an on-the-fly rendering, know that it's complex (LilyPond took years of dev). But maybe a simpler approach: a HTML5 canvas drawing tab lines and notes could be done for quick visual feedback in an app. That requires understanding spacing (e.g., proportional to time, or fixed spacing per beat).

- *Evaluation and Benchmarking Conventions:*
- Look at MIREX contest for multi-F0 or transcription (if any for guitar) to see how they evaluate.
- Understand "frame vs note evaluation" thoroughly; e.g., note onset within 50ms and pitch within semitone counts as correct.
- Alignment metrics: if measuring how close transcribed note timings are to ground truth, maybe use something like "onset F1 with 50ms tolerance" or "notewise precision with onset & offset tolerance".
- For tab-specific, use metrics like in Fretting-Transformer paper: pitch accuracy vs tab accuracy (pitch accuracy ignores which string, tab accuracy requires string correct too).
- Also consider *Levenshtein distance* on tab sequences as a metric (comparing the output tab string to ground truth tab as sequences of tokens).
- Ensure the concept of train/test splitting by song or by player is understood (to avoid overfitting to specific guitar timbres or playing styles).

At the end of Advanced, one should be able to: - Understand and possibly replicate results from recent papers (like implement a small version of Fretting-Transformer, or at least use such a model). - Design experiments to test improvements (e.g., try a new fusion strategy and evaluate it properly). - Be aware of the current limits of technology (knowing open problems we listed, one knows where even SOTA fails). - Contribute new ideas: e.g., think "maybe I can use a Graph Neural Network to model the fretboard and fingers" – which is an advanced concept requiring all knowledge from before plus creativity.

7.4 Practical Skills Development

Knowledge is solidified by hands-on projects and targeted exercises. Here are some ideas and exercises, plus suggested resources to solidify skills:

Project Ideas at Each Level:

- *Foundations Project:* Build a simple "tone detector" for guitar. Record individual notes on each string, then write a program to identify which string a new note comes from using frequency analysis (e.g., by detecting the pitch and comparing to expected tuning frequencies). This uses basic DSP and music theory of tuning.
- *Another Foundation Project:* Transcribe a simple melody by ear and encode it in MusicXML. This trains musical ears and understanding of notation. Alternatively, take a simple tablature and synthesize audio with a programming library (like given tab, generate a MIDI).
- *Intermediate Project:* Implement a monophonic guitar transcription using a neural network. For instance, collect audio of single notes (maybe use GuitarSet single notes from solos) and train a CNN to classify the pitch. Extend it to output multiple pitches for simple two-note chords (multi-label classification). Evaluate on some examples.
- *Intermediate CV Project:* Write a program that takes a video frame of a guitar and automatically finds the fretboard and overlays fret lines. Use edge detection or Hough lines. For moving video, try to keep tracking it. This will expose you to image processing and the challenges with varying lighting or angles.

- *Combined Project:* Build a prototype that uses audio onset detection and video finger detection to identify when and where a note is played: For example, record yourself playing a short riff slowly. Then code: detect onsets in audio for times, at those times find the fingertip positions from video, output a list like "Time 1.2s: finger at 3rd string 5th fret". See if that matches what you played. This is a mini version of the whole system.

- *Advanced Project:* Use a pre-trained model (like Onsets & Frames checkpoint or an open model like MT3) and fine-tune it on a small guitar dataset. Experiment: does fine-tuning on GuitarSet improve transcription of guitar audio vs the base model? This teaches about transfer learning and model training.

- *Advanced Project:* Try to implement the fusion in a learned way: maybe train a small neural network that given audio pitch probabilities and visual finger probabilities outputs a combined probability of each string being active. You could simulate training data for this. It's like a miniature late-fusion NN.
- *Advanced Symbolic Project:* Write a script that takes a sequence of notes with string assignments and outputs a LilyPond file for a tab+notation. That will force you to handle notation details.

Exercises to Master Concepts: - Calculate the spectrogram of an audio sample by hand for a couple of frames (small FFT). Understand what each bin corresponds to musically. - Take a short recording of two-note chords. Manually identify their fundamental frequencies via Fourier analysis (e.g., use numpy FFT and see peaks), verify they correspond to the notes you expect. - Use MediaPipe on some images and log the coordinates of fingertips. Write a small function that given those coordinates and known fretboard corners (assume you manually identify nut and 12th fret positions in the image) computes which fret the finger is nearest. - Given a sequence of notes (like C major scale notes with timings), try to assign frets manually in different ways and consider which is easier to play. This builds intuition for the algorithmic approach. - Write small programs to convert between representations: e.g., from MIDI number to note name, from note name to frequency (using A440 reference). - Evaluate transcriptions: take an existing transcription and a ground truth, and practice computing precision/recall. For example, use a known GuitarPro tab and a slightly different version and manually align and count false positives/negatives. This clarifies evaluation metrics.

Suggested Reading Order for Papers: 1. Start with a broad MIR survey or textbook chapter on AMT (e.g., from Müller's book or a tutorial from ISMIR) to get context. 2. Read some classical guitar transcription papers: e.g., Raphael (2009) on guitar transcription with HMM, or early vision like Paleari 2008 to see the multimodal idea introduction. 3. Onsets & Frames paper (Hawthorne 2018) – since it's foundational for modern AMT. 4. Wiggins & Kim ISMIR 2019 (TabCNN) to see the first deep learning specific to guitar tab. 5. Chen et al. 2022 (ICASSP) – for electric guitar transformer approach and new dataset introduction. 6. Riley et al. ICASSP 2024 (High-res guitar transcription) to see domain adaptation and the GAPS dataset. 7. Hamberger 2025 (Fretting-transformer) and Edwards 2024 (MIDI2Tab) to cover latest tab inference methods. 8. Mitsou et al. 2024 (Multimodal technique dataset) to understand technique classification and dataset creation. 9. Bae et al. 2025 (Audio-Visual bass transcription) to see a recent example of A/V fusion and fingering optimization. 10. TART arXiv 2025 (Technique-Aware Transcription) to see a pipeline integrating techniques, and get references to many recent works (as we saw in its reference list). 11. (For fun/vision) Goldstein & Moses 2018 (Silent video transcription) – see how far vision-only can go.

This order goes from general to specific and from earlier to latest, building context then diving into cutting-edge.

Key Tutorials/Courses: - *Coursera/EdX:* "Audio Signal Processing for Music Applications" by Xavier Serra (Coursera) – great for DSP and MIR basics. - *Stanford MUSIC 320 (or 420):* they often have transcription related content and assignments. - *YouTube channels:* The "AI and Music" lecture series, or papers explained by researchers (some MIR researchers have talks available). - *OpenCV courses:* many free ones to learn about feature detection, etc., which can be applied in our context. - *Magenta's blog:* they have posts explaining Onsets & Frames, etc., which can be approachable. - *MIR Workshops:* ISMIR often has late-breaking demos and some video archives of tutorials. - *PyTorch Tutorials:* specifically for audio (torchaudio), there are official recipes and also unofficial ones (like how to train a simple music transcription model).

GitHub Repositories: - *magenta/magenta:* Has Onsets&Frames code. - *tensorflow/models:* some contain audio recognition or music AI code. - *huggingface/transformers:* contains some audio processing models and could host a pre-trained MT3 or similar (if not now, maybe soon). - *mir_eval library:* for evaluation code. - *MediaPipe examples:* Google Mediapipe GitHub for how to integrate it. - *OpenSource Transcription projects:* e.g., "omnizart" (by Music and Audio Computing Lab Taiwan) – they have a toolkit that includes guitar transcription based on some known models. - *GuitarPro parsers:* like PyGuitarPro (for reading/writing tabs). - *Datasets:* Many datasets have GitHub or webpages: GuitarSet (has a weebly with some code), IDMT-SMT has some example scripts.

Working through implementing and exploring these will greatly solidify knowledge and also reveal the tricky nuances, which is crucial for innovative work.

Finally, **staying updated:** follow ISMIR conferences, IEEE Signal Processing letters, and the archives on arXiv (cs.SD, eess.AS) for new papers. Joining the MIR community forums or Slack can also help for discussions.

By following this curriculum, one would progress from basic understanding of music and signals to being able to contribute to cutting-edge research or development in automatic guitar transcription, which inherently synthesizes those disciplines. Each step builds the confidence and skillset required to not only use existing tools but also innovate new solutions for the unsolved challenges in this domain.

Bibliography

(Key sources referenced above, grouped by type for clarity.)

Surveys & Theses: - Emmanouil Benetos et al. “Automatic Music Transcription: An Overview.” *IEEE Signal Processing Magazine*, 36(1):94-111, 2019. (Comprehensive AMT survey) - Jakob Abesser. *“Automatic Transcription of Bass Guitar Tracks Applied to Music Genre Classification and Sound Synthesis.”* PhD Thesis, TU Ilmenau, 2013. (Focus on bass guitar, includes playing technique detection) - Mark Asmar. *“A Computer Vision-Based Automatic Transcription of Guitar Music from RGBD Videos.”* Master’s Thesis, Polytechnique Montréal, 2022. (Vision-only guitar transcription using depth cameras) - Anna Hamberger et al. “Fretting-Transformer: Encoder-Decoder Model for MIDI-to-Tablature Transcription.” *Proc. ICMC 2025*, pp. 438–445, 2025. (Guitar tab inference model using Transformer) - Domenico Stefani. *“On the Importance of Temporally Precise Onset Annotations for Real-Time MIR: Findings from the AG-PT-set Dataset.” Proc. Audio Mostly (AM’24)*, pp. 270–284, 2024. (Introduces Acoustic Guitar Playing Techniques dataset and emphasizes onset timing)

Datasets: - Xi Yang et al. “GuitarSet: A Dataset for Guitar Transcription.” *Proc. ISMIR 2018*, pp. 451–458, 2018. (Introduces GuitarSet, with hexaphonic recordings and annotations) - Yu-Hua Chen et al. “Towards Automatic Transcription of Polyphonic Electric Guitar Music: A New Dataset and a Multi-Loss Transformer Model.” *Proc. ICASSP 2022*, pp. 786–790, 2022. (Introduces EGDB dataset and Transformer model) - Alexandros Mitsou et al. “A Multimodal Dataset for Electric Guitar Playing Technique Recognition.” *Data in Brief*, vol. 52, 109842, 2024. (549 audio+video clips of electric guitar techniques) - Jackson Loth et al. “GOAT: A Large Dataset of Paired Guitar Audio Recordings and Tablatures.” arXiv: 2509.22655, 2025. (5.9 hours DI guitar audio + tabs, plus 29.5h augmented audio) - Hegel Pedroza et al. “Guitar-TECHS: An Electric Guitar Dataset Covering Techniques, Musical Excerpts, Chords and Scales Using a Diverse Array of Hardware.” *Proc. ICASSP 2025*, (to appear) [9] [1]. (5+ hours, multi-perspective audio, multi-technique dataset)

Audio-Only Transcription Research: - Curtis Hawthorne et al. “Onsets and Frames: Dual-Objective Piano Transcription.” *Proc. ISMIR 2018*, pp. 50–57, 2018. (Piano transcription model that influenced guitar models) - Qiuqiang Kong et al. “High-Resolution Piano Transcription with Pedals by Regressing Onset and Offset Times.” *IEEE/ACM TASLP*, 29:3707–3717, 2021. (Basis for high-res guitar transcription via domain adaptation) - Andrew Wiggins and Youngmoo Kim. “Guitar Tablature Estimation with a Convolutional Neural Network.” *Proc. ISMIR 2019*, pp. 284–291, 2019. (TabCNN model for direct audio-to-tab on acoustic guitar) - Rachel M. Bittner et al. “A Lightweight Instrument-Agnostic Model for Polyphonic Note Transcription and Multi-Pitch Estimation.” arXiv:2212.04340, 2022. (Example of instrument-agnostic transcription approach) - Xavier Riley et al. “High Resolution Guitar Transcription via Domain Adaptation.” *Proc. ICASSP 2024*, pp. 1051–1055, 2024. (Adapts piano model to guitar using GAPS dataset; audio-only transcription focus)

Audio-Visual & Vision-Based Research: - Marco Paleari et al. “Audio-Visual Guitar Transcription.” *Proc. ICME 2008*, pp. 569–572, 2008. (Early system with fretboard tracking, hand detection, audio-visual fusion) - Shir Goldstein and Yael Moses. “Guitar Music Transcription from Silent Video.” *Proc. BMVC 2018*, p. 309, 2018. (Transcribing guitar by analyzing video only) - Hyunwoo Bae et al. “An Audio-Visual Framework for Transcription and Fingering Optimization in Bass Guitar Performance.” *Proc. ACM UIST Adjunct*, pp. 57–59, 2025. (Bass guitar multimodal transcription, uses video for fingering + audio for notes) - Domenico Catalano et al. “Computer-Vision Based Guitar Tabulator.” *Proc. ICASSP 2020*, 4322–4326, 2020. (Uses video of guitar to estimate tablature, focusing on fret detection and pose)

Playing Technique & Other: - Li Su et al. “Sparse Cepstral and Phase Codes for Guitar Playing Technique Classification.” *Proc. ISMIR 2014*, pp. 9–14, 2014. (Guitar technique classification from audio; basis of 6580-sample dataset) - Hegel Pedroza et al. “Leveraging Real Electric Guitar Tones and Effects to Improve Robustness in Guitar Tablature Transcription Modeling.” *Proc. DAFx 2023*, 2024 (forthcoming). (Augmenting training with diverse electric guitar tones) - Samir I. Sayegh. “Fingering for String Instruments with the Optimum Path Paradigm.” *Computer Music Journal*, 13(3):76–84, 1989. (Classic DP algorithm for fingering, often cited in later tab inference work) - Drew N. Edwards et al. “MIDI-to-Tab: Guitar Tablature Inference via Masked Language Modeling.” arXiv:2408.05024, 2024. (New approach using a Transformer to infer tabs from MIDI; demonstrates state-of-art tab accuracy)

Tools & Libraries: - Colin Raffel and Justin Salamon et al. *mir_eval*: “Towards a Standardized Comparison of Audio-to-MIDI Conversion Algorithms.” *Proc. ISMIR 2014*, pp. 347–352, 2014. (Introduced mir_eval library; defines transcription evaluation metrics) - MediaPipe Hands. *CVPR 2020, Demonstration*. (Google’s hand tracking, on-device real-time). - PyGuitarPro Library – an open-source Python library to parse and write Guitar Pro files (GP3/4/5 formats). (Useful for symbolic tablature handling) - Librosa Library – Brian McFee et al. “librosa: Audio and Music Signal Analysis in Python.” *Proc. SciPy 2015*, 2015. (Primary Python library used for audio feature extraction in many MIR projects)

Each of these sources contributed to the concepts and techniques discussed, providing background, algorithms, or data. The combination of peer-reviewed papers, dataset documentation, and existing software documentation forms the knowledge base for state-of-the-art audio-visual guitar transcription development.

---

1 2 5 6 7 8 9 Guitar-TECHS: An Electric Guitar Dataset Covering Techniques, Musical Excerpts, Chords and Scales Using a Diverse Array of Hardware
https://arxiv.org/html/2501.03720v1

3 Hegel Pedroza - Google Scholar
https://scholar.google.com/citations?user=ARJo9foAAAAJ&hl=en

4 Zixun Nicolas Guo (@nicolasguozixun) / Posts / X - Twitter
https://x.com/nicolasguozixun