ChatGPT

Detecting Breathing via Microphone (Inhale & Exhale)

Detecting human breath (inhalation and exhalation) using a microphone is an active area of development, useful for health monitoring, hands-free game controls, and other interactive applications. Below, we survey existing **open-source projects**, relevant **research and technical articles**, discuss **feasibility considerations** (noise, latency, etc.), and outline common **techniques** for microphone-based breath detection.

Open-Source Projects and Libraries for Breath Detection

- **Interactive Breathing Visualization (Unity)** – An open-source Unity tool that captures live microphone audio and splits it into eight frequency bands. By thresholding specific frequency bands, it distinguishes breathing in versus out in real time [1]. For example, when the low-frequency band exceeds a threshold it classifies as an inhale (moving a virtual object closer), and when other bands spike it marks an exhale (moving the object away) [1]. The project notes that an external mic held close to the mouth is needed, and the detection thresholds must be *tuned per device* for best results [2]. This highlights the practical calibration often required in microphone breath detectors.

- **Breathing-Classification (Python ML)** – A GitHub project exploring *artificial intelligence* techniques for real-time breath sound detection via a standard microphone [3]. It implements and compares multiple methodologies – including deep learning models (e.g. spectrogram-based CNNs and audio embeddings like VGGish) – to find an accurate and efficient approach [3]. The codebase (primarily Python) provides a foundation for training models on breath audio and even incorporates noise reduction (it includes a "DeepFilterNet" module for filtering background noise). This open resource can be adapted or extended for breath-controlled applications.

- **Respiro-en (PyTorch Model)** – An open-source PyTorch implementation of a 2024 research paper on frame-wise breath detection [4]. Though targeted at improving Text-to-Speech, it offers a pretrained network that **detects the positions of breath sounds in speech recordings** [5]. This model was trained on large speech datasets with labeled breath segments, and achieves high accuracy in tagging breathing moments within audio. Developers could potentially repurpose such a model (or its architecture) for real-time detection of user breaths, albeit with adaptation to real-time constraints and non-speech scenarios.

- **Custom Breath Detector Prototype** – Independent developers have also built their own breath-detection tools. For example, Chris Smith created a small app to detect a user's **exhale** using an off-the-shelf microphone [6]. His approach was to monitor the audio's frequency spectrum for peaks in the range characteristic of breath noise, while filtering out frequencies typical of speech [7]. The result was a demo that successfully triggered on breaths and *avoided false positives* when the user was speaking [8]. This prototype (described in a blog post) illustrates how straightforward signal processing can achieve breath detection with minimal hardware.

Note: General audio-processing libraries can aid development of custom detectors. For instance, Python's *Librosa* or *PyAudioAnalysis* provide tools to record audio, compute spectra, and extract features [9], which developers can use to build their own breath detection algorithms.

Research and Technical Articles on Audio-Based Breath Detection

Academic research and technical blogs have investigated microphone-based breathing detection from medical and signal-processing perspectives:

- **Breath Detection Algorithms (Medical Research):** One academic thesis (University of Groningen, 2016) demonstrated a laptop+microphone setup to record and detect breaths [10]. Their algorithm used signal envelope extraction and peak detection to count breaths and even classify intensity ("soft, mild, hard" breaths) [11] [12]. In controlled quiet conditions, it achieved high precision (~94.7%) and recall (~88.9%) for normal breaths [12]. However, the authors reported the method was **sensitive to noise** – performance dropped significantly in a "disturbed environment" (noisy background) [13]. They also discussed challenges in making the detection real-time, such as choosing a processing window size without missing breaths [14].

- **Microphone Array Enhancements:** Recent research has looked at improving breath sensing using multiple microphones. **BuMA (2022)**, for example, is a system that uses a **microphone array with beamforming** and audio filtering to enhance breathing sounds while filtering out ambient noise [15]. In tests up to 30 cm from a person, BuMA improved detection accuracy by about **12%** (achieving ~66% accuracy) over single-microphone filtering methods [16]. It could also estimate breathing metrics like inhale/exhale duration ratios with ~86–92% accuracy [17]. This shows that with specialized hardware (arrays) and DSP algorithms, breathing can be detected more reliably in noisy settings or at a short distance.

- **Smartphone Breathing Apps:** Researchers have successfully utilized smartphone microphones for breathing detection and even medical screening. For instance, a University of Washington project (*SpiroSmart*) turned a smartphone into a spirometer by analyzing exhalation sounds; it reportedly measured lung function within ~5% of a dedicated spirometer's readings [18]. Another example is **QUT Inspire**, a smartphone-based respiratory therapy game that "virtualizes" an incentive spirometer by detecting the user's inhale sound via the phone's mic [19] [20]. In evaluation, the app could reliably detect strong inhalations **at up to 50 cm away**, but for normal breathing it required the phone to be held very close (~5 cm from the mouth) for consistent detection [21]. These studies highlight both the promise and practical limits of phone mics – they are convenient and low-cost, but work best with proper technique (e.g. smaller mouth opening, higher airflow, close range) to generate a clear audio signal [21].

- **Health Monitoring and COVID-19:** During the COVID-19 pandemic, audio-based breath monitoring gained interest. A technical blog by Ross Ashman (2020) describes building a simple classifier to tag audio clips with **breaths vs. coughs vs. speech** [22]. Using open datasets (ESC-50 for breath/cough sounds and Google Speech Commands for spoken numbers) and features extracted with Librosa, he trained a model that achieved about **89% F1-score** at identifying breath sounds (versus cough or speech) in a test set [23]. On the research side, others have investigated detecting respiratory illness via breath audio – e.g. differentiating sleep apnea events or even diagnosing COVID-19 from

recorded breathing patterns [24]. These efforts often emphasize robust algorithms that can handle real-world noise since home environments are rarely perfectly quiet.

Feasibility Considerations (Noise, Latency, False Alarms, Mobile Constraints)

Building a breath-controlled application on consumer devices comes with several practical challenges:

- **Ambient Noise Sensitivity:** Breathing sounds are relatively subtle and easily masked by background noise. Many approaches assume a **quiet environment** for reliable operation [25]. In noisy settings, microphones will pick up other sounds (air conditioning, wind, speech, etc.) which can confuse breath detection algorithms [26]. Researchers note that audio-based breath monitoring "suffers from issues with background noise and other disturbances" [26]. This can lead to missed breaths or **false positives** (e.g. interpreting rustling or wind noise as a breath). To mitigate this, developers often incorporate noise filtering (hardware or software) and focus on frequency ranges that distinguish breathing from ambient sounds. For example, one project explicitly filtered out human voice frequencies to avoid triggering on speech [7] [8]. High-quality mics or microphone arrays with directional pickup can also improve the signal-to-noise ratio, but add cost and complexity.

- **False Positives and Reliability:** Even with noise reduction, false detections can occur. Sudden noises or certain phonetic sounds (like a strong "h" or sigh in speech) might be mistakenly flagged as a breath. In one evaluation, a breath detection algorithm achieved ~94% overall accuracy with false positives around **5% of total detections** [27] – indicating occasional spurious triggers are expected. Achieving low false alarm rates requires careful tuning. Multi-step logic can help (e.g. require a certain pattern or duration of noise before declaring a breath). In practice, a system might need calibration for each user or environment to distinguish genuine breathing from other sounds while keeping missed detections low [28]. Testing in diverse conditions (quiet room, outdoor, with music in background, etc.) is important to refine the algorithm and avoid frustrating the user with misfires.

- **Latency and Responsiveness:** Detecting an inhale/exhale in real time inevitably introduces a small delay, as the system must "hear" enough of the sound to be sure it's a breath. Most solutions process audio in short *windows* (e.g. 100–500 ms chunks). A too-long window adds **latency** – the app might respond noticeably after the breath is done – but a too-short window might not capture the full breath signal or could be dominated by noise. Selecting the window size is a trade-off [29] [14]. For instance, one study noted that if processing a chunk takes longer than the chunk's duration, you risk *losing breaths* that occur in the interim [14]. Efficient code and possibly streaming analysis (processing audio continuously with overlap) can minimize delay. In practice, many interactive demos use a threshold-based trigger which can fire almost immediately when the amplitude or frequency criterion is met – this yields a snappy response, but one must ensure the trigger isn't so early that it responds to non-breath noises. Additionally, providing visual feedback (like a breathing bar or waveform) can help users time their breaths with the system latency.

- **Mobile Device Constraints:** On smartphones or AR/VR headsets, we are constrained by built-in mic characteristics and limited computing resources:

- *Mic frequency response:* Built-in MEMS microphones in phones are designed for voice (~300–3400 Hz) and often employ automatic gain control or noise suppression [30]. Quiet breath sounds (especially inhalations) may be near the lower threshold of what the mic picks up, or the phone's algorithms might treat sustained breath noise as "background" to cancel. Some apps address this by instructing users to breathe forcefully or use a smaller mouth opening to create a higher-frequency whistle-like component [21]. Consistency across devices is another issue – different phone models have different mic sensitivities. For example, the Unity project above required threshold calibration for each microphone and setup [2]. Developers should allow some form of calibration or adaptivity in their apps.
- *Processing power and battery:* Continuous audio processing (especially using ML models) can tax a mobile device. Simpler DSP algorithms (energy detectors, band-pass filters, etc.) are very lightweight and can run easily in real time even on a phone. Machine learning models, if large, might be too slow or energy-intensive for continuous use. To tackle this, researchers have tried **lightweight models** (e.g. using MobileNetV2, a small CNN, to classify respiratory sounds in real time [31]) or offloading heavy computation to the cloud (less ideal for interactive latency). When designing a breath-controlled game, one might prefer a well-optimized model or even a classical approach to conserve battery. It's also wise to manage the duty cycle (perhaps only listen/compute when needed, not 24/7) – one study limited its breath recording sessions to ~1 hour due to memory and processing constraints [13].
- *Device feedback and multi-functionality:* Using the mic for breath detection may conflict with other uses (e.g. voice chat in a game). On mobile, your app might need permission for microphone access and should handle cases where the mic input is interrupted. Fortunately, many modern devices do have multiple mics (for stereo or noise cancellation) – an app could possibly utilize one mic for breath input while another handles speech, if the hardware/OS allows it.

Techniques for Microphone-Based Breath Detection

A variety of signal processing and machine learning techniques are used (often in combination) to detect breathing from audio:

- **Time-Domain Envelope & Thresholding:** A basic approach is to detect the rise and fall of sound intensity caused by breaths. The microphone signal can be rectified (squared) and **low-pass filtered** to extract a smooth amplitude envelope [32]. Breathing creates a slow, oscillatory envelope that stands out from silence. By setting a threshold on this envelope, one can trigger when a breath occurs. Further logic can differentiate inhale vs exhale by the pattern (e.g. inhale may be shorter and sharper intake sound, exhale longer). This method is simple and was used in early research: for example, the Groningen thesis used envelope peak detection to count breaths [33] [34]. Envelope thresholding works well in quiet settings but is easily disturbed by other sounds that also modulate the amplitude. It may require tuning of gain and threshold per environment to reliably catch breaths without false alarms [35].

- **Frequency Band Analysis:** Breathing sounds have a distinctive spectral signature compared to voiced speech or environmental noise. They are often broadband "whooshing" noises concentrated at lower frequencies (though high airflow through a small opening can create higher-frequency components, like a whistle). Techniques in this category apply **filters or FFT analysis** to isolate frequencies where breath noise is strongest. The Unity project above is one example – it splits audio into eight bands and checks specific bands for activity to classify inhale vs exhale [1]. Similarly, Chris

Smith's prototype looked for peaks in a certain frequency range characteristic of exhalation while ignoring other bands (to avoid speech) [7]. In practice, one might use a band-pass filter (for, say, 300–600 Hz) and measure energy in that band as an indicator of breath. Tuning the band is key: too narrow and you might miss breaths (since breath noise spans a spectrum), too broad and you'll include unwanted noise. Some systems use multiple bands – for instance, detecting an inhale might rely on a slight high-frequency "hiss" component, whereas exhale might have more low-frequency energy; combining these cues can improve detection of breath phase.

- **Noise Reduction & Audio Preprocessing:** Given the dominance of ambient noise issues, many pipelines include a preprocessing stage to enhance breathing sounds. This could be as simple as a high-pass filter to remove low-frequency rumble or a spectral subtraction algorithm to filter consistent background hum. More advanced setups use adaptive noise cancellation or reference sensors. In open-source projects, we even see deep learning noise suppression (e.g. *DeepFilterNet* in the Breathing-Classification project) being applied to the audio before analysis. Another advanced approach is **beamforming** with multiple microphones – by focusing the "listening beam" on the user's mouth/nose, off-axis noises can be attenuated. The BuMA research combined beamforming with filtering to specifically amplify breathing sound and suppress typical home noises [15]. While not all consumer devices have mic arrays, some do (laptops, smart speakers, even smartphones with 2–3 mics), and leveraging those can significantly boost detection reliability.

- **Machine Learning Classification:** Machine learning can detect breathing either as an event (classification) or as a continuous state (regression of breathing rate, etc.). There are two main ML approaches:

- *Feature-based classifiers:* These involve extracting features from audio (e.g. Mel-frequency cepstral coefficients, spectral flux, zero-crossing rate, etc.) and feeding them to a classifier like an SVM, random forest, or a simple neural network. The classifier is trained on labeled data (segments with breath vs no-breath, or inhale vs exhale). For example, the COVID-19 blog project used features and trained a Random Forest to label 300 ms audio chunks as "breath", "cough", or "digit" [22] [23]. It obtained roughly 87–89% accuracy for breath detection, demonstrating the viability of classical ML with the right features and data. The advantage is that these models can be lightweight and fast, suitable for real-time use on mobile.

- *Deep learning (end-to-end or frame-wise):* Here, the raw waveform or spectrogram is fed into a deep neural network (CNN, RNN, or Transformer) that learns to detect breaths. For instance, the *Respiro-en* model (Interspeech 2024) uses a neural network to output a probability of breath at each frame of audio [5]. Similarly, research on sleep apnea detection employs CNN or LSTM models to pick out breathing sounds from noisy overnight audio [24]. Deep models can capture subtle patterns (like the slight difference between a gasp and a sigh) and may handle complex noise better if trained on diverse data. However, they need large datasets for training and can be computationally heavy. Some studies have explored compressing models or using efficient architectures (e.g. MobileNetV2 in a respiratory sound classifier [31]) to make them feasible on devices. A well-trained ML model can also provide more than a binary output – e.g. regression for breath intensity or classification into normal vs abnormal breath sounds, which could enrich an application's feedback.

- **Breath Phase and Pattern Analysis:** Beyond detecting individual breaths, some techniques aim to distinguish **inhale vs exhale** phases and measure breathing rate or consistency. Inhale vs exhale

can often be inferred by timing (inhale is usually shorter) or by subtle spectral differences (exhales tend to produce louder, lower-frequency noise when air is expelled forcefully). In-ear microphone research, for example, has shown it's possible to classify breathing phase because the sound profile inside the ear canal differs when you breathe in versus out [36] [37]. With external mics, distinguishing phases may require higher quality audio or multi-sensor input (some setups combine a microphone with a thermistor or IR sensor to detect airflow direction). Nonetheless, software can do a decent job by looking at the envelope shape: a sharp, short burst might indicate an inhale (sniff), while a longer, whooshing decay is an exhale – this was the logic behind some signal-processing algorithms in earlier research [33] [38]. For applications like games, simply detecting any breath vs silence might suffice, but for biofeedback or musical control, identifying inhale vs exhale opens more interaction possibilities.

In summary, **microphone-based breath detection is feasible** on consumer devices and has been demonstrated in both hobby projects and rigorous studies. Open-source tools and libraries exist to jump-start development, from Unity scripts for frequency-based detection [1] to Python notebooks for training breath classifiers [22]. However, developers must tackle real-world challenges like background noise, device variability, and the need for quick, reliable responses. By combining clever signal processing (to amplify the breaths and suppress everything else) with robust algorithms or ML models, one can create breath-controlled applications – whether for health (e.g. breathing exercises, symptom monitoring) or for fun (breath-driven games, musical instruments) – that work consistently on everyday microphones. With ongoing improvements (like better noise cancellation and more on-device AI), the accuracy and usability of microphone breath detection are only getting better [27] [16].

Sources: The information above is drawn from a range of connected sources, including open-source project documentation, academic papers, and developer blogs that specifically address audio-based breath detection. Key references include GitHub project READMEs [39] [3], research findings on noise and accuracy [13] [16], and firsthand reports of implemented systems [6] [22], as cited throughout the text.

---

[1] [2] [39] GitHub - DeniceTuinhof/InteractiveBreathingVisualisation: An interactive breathing visualisation tool that uses a microphone
https://github.com/DeniceTuinhof/InteractiveBreathingVisualisation

[3] GitHub - tomaszsankowski/Breathing-Classification: To explore and implement various artificial intelligence based techniques for real-time detection of breath sounds using audio data captured from a computer microphone. The project aims to compare different methodologies to determine the most accurate, efficient, and practical approach for real-time breath detection.
https://github.com/tomaszsankowski/Breathing-Classification

[4] [5] GitHub - ydqmkkx/Respiro-en: Official implementation of paper: Frame-Wise Breath Detection with Self-Training: An Exploration of Enhancing Breath Naturalness in Text-to-Speech
https://github.com/ydqmkkx/Respiro-en

[6] [7] [8] Audio Based Breath Detection — Chris Smith
http://www.christopherlsmith.com/audio-breath-detection

[9] [22] [23] Cough and breath detection from audio for remote COVID-19 monitoring | by Ross Ashman (PhD) | Medium
https://medium.com/@rossashman/cough-and-breath-detection-from-audio-for-remote-covid-19-monitoring-1fcca2ffdb91

10 11 12 13 14 25 28 29 32 33 34 35 38 fse.studenttheses.ub.rug.nl
https://fse.studenttheses.ub.rug.nl/11311/1/Thesis.pdf

15 [PDF] BuMA: Non-Intrusive Breathing Detection using Microphone Array
https://par.nsf.gov/servlets/purl/10357213

16 BuMA: Non-Intrusive Breathing Detection using Microphone Array
https://dl.acm.org/doi/10.1145/3539490.3539598

17 BuMA: Non-Intrusive Breathing Detection using Microphone Array ...
https://par.nsf.gov/biblio/10357213-buma-non-intrusive-breathing-detection-using-microphone-array

18 App lets you monitor lung health using only a smartphone | UW News
https://www.washington.edu/news/2012/09/18/app-lets-you-monitor-lung-health-using-only-a-smartphone/

19 20 21 30 Seeking Inspiration: Examining the Validity and Reliability of a New Smartphone Respiratory Therapy Exergame App
https://www.mdpi.com/1424-8220/21/19/6472

24 37 Real-Time Detection of Sleep Apnea Based on Breathing Sounds ...
https://pmc.ncbi.nlm.nih.gov/articles/PMC9996414/

26 27 BreathFinder: A Method for Non-Invasive Isolation of Respiratory Cycles Utilizing the Thoracic Respiratory Inductance Plethysmography Signal - PubMed
https://pubmed.ncbi.nlm.nih.gov/39189036/

31 Automated detection of abnormal respiratory sound from electronic ...
https://www.sciencedirect.com/science/article/pii/S0208521623000608

36 Classification of Breathing Phase and Path with In-Ear Microphones
https://www.mdpi.com/1424-8220/24/20/6679