*Um*s considered harmful

Seongmin Park

hamanlp.org/research/ums

Abstract

We made an interview prep tool that helps you train not to say *um*. You practice answering interview questions aloud, and every time we detect an *um*, we commit a minor crime against humanity on your behalf. Like finding out which side a falling jam toast will land on a new carpet. Sending your voice to a cloud transcription service would have required a server, which would have required money, which would have required a business model, so instead we built tiny grapheme-to-phoneme and audio-to-phoneme neural networks that run entirely in your browser. There's no server. We don't want your voice data. We just want you to stop saying *um*. Please.

1 Introduction

So the other day, I was listening to one of my meeting recordings. And I realized something terrible. I was saying *um* constantly, and it was undermining every smart thing I was saying. The problem is: I say a lot of smart things. This is a major issue. And I imagine not only for me.

I am not a bigot. I know there are some good *um*s. "Um, I think we should break up." That *um* is load-bearing. It's doing emotional work. Take it away and you sound like a psychopath. "Um, I think that's my lunch." That *um* is keeping the peace. Five stars.

But then there's the bad *um*. The *um* that shows up every 5 seconds while you're explaining why you're the right person for the job, always between "I" and "think." It doesn't mean anything. It's just.. *there*. We built a tool that detects the bad ones. The good *um*s are safe. Your breakup speech is safe.

And here's the thing. Nobody tells you. Not because they're bad people. It's just not a thing you bring up, you know? "Hey, great presentation, also you said *um* forty-one times." You can't do that. There's no polite version of that sentence. It lives in the zone of things everyone notices but nobody says, like a slightly wrong pronunciation of a common word. You just let it happen and move on.

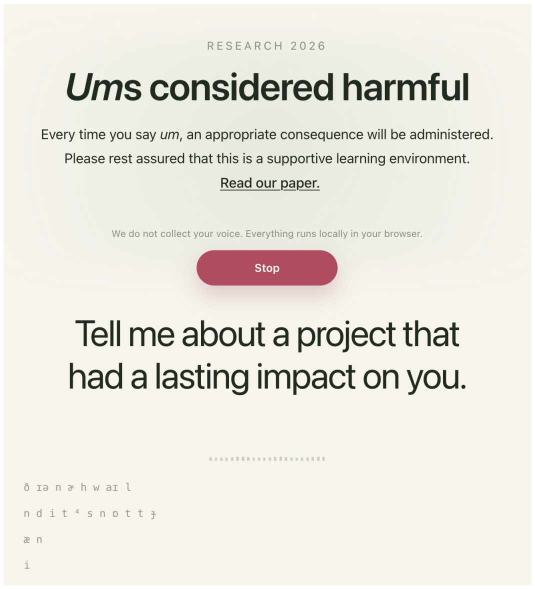
So we made a tool that will tell you. It can do that because it's a cute little neural network that doesn't have to see you at standup tomorrow.

2 What we made

Using our tool, you answer out loud to a question you see on screen, as if you were in a real interview.

While you speak, a small neural network transcribes your speech into phonemes in real time. When it detects that you've said *um*, something mildly unfortunate happens on screen (e.g. a piece of toast falls jam-side down). These accumulate. By the end of a session, the screen is littered with the small wreckages of your *um*s.

No audio is ever sent to a server and no account is needed. We wanted to make something you could open for five minutes of practice before an interview and close without worrying about anyone harvesting your data for profit.


RESEARCH 2026

*Um*s considered harmful

Every time you say *um*, an appropriate consequence will be administered.
Please rest assured that this is a supportive learning environment.
Read our paper.

We do not collect your voice. Everything runs locally in your browser.

Tell me about a project that had a lasting impact on you.

ð ɪə n ɚ h w aɪ l
n d i t ⁴ s n ɒ t t ɟ
æ n
i

Figure 1: We give you behavioral interview questions, the kind that begins with "tell me about a time" and ends with you staring at the ceiling for a few seconds.

2.1 Internal models

The runtime model is a single CTC-based acoustic encoder, 4.8 MB in fp16 precision, shipped as an ONNX file and executed locally via ONNX Runtime Web. It takes raw audio at 16 kHz and produces, for each small time frame, a probability distribution over phoneme tokens. It comes from an open-source phoneme inference library we built.

The model outputs IPA (International Phonetic Alphabet) phonemes, not words.

2.1.1 Training Data

To train an audio-to-phoneme model, you need audio paired with phoneme labels. Which we of course did not have, so we built the training data by building the thing that builds the thing. First, we developed our own grapheme-to-phoneme (G2P) model as an encoder-decoder architecture trained to convert written text into IPA sequences. Once we were confident in the G2P model's quality we used it to generate phoneme labels from existing text transcripts, creating the training targets for the acoustic model. From there, we distilled the G2P model onto the ASR model, so that at runtime, only the audio-to-phoneme model is needed.

2.1.2 Decoding

The decoding process is standard CTC:

1. For each frame, pick the most likely phoneme token.
2. Collapse consecutive identical tokens into one.
3. Remove blank tokens (the CTC alignment placeholders).
4. Split the remaining tokens into words at word boundary markers.

We apply two small biases before picking the most likely token in step 1. A **blank bias** of -0.1 slightly discourages the model from going silent during frames that contain speech, which reduces missed phonemes. An **unknown-token bias** of -1.5 more firmly discourages the model from emitting an "I don't know WTF is this BS audio!" token. We would rather it guess a nearby phoneme than emit nothing, since even a slightly wrong guess is more useful for filler detection than silence. When in doubt, guess. This is good life advice in general, but especially for tiny acoustic models with something to prove. The total model download is about 5 MB. For reference, that is smaller than most photos on your phone.

2.1.3 Why phonemes instead of words?

Two reasons. First, word-level ASR models that are accurate enough to be useful tend to be large, ranging from hundreds of megabytes to several gigabytes. That's a lot to ask someone to download for a practice tool they'll use for five minutes. Our model is three orders of magnitude smaller. It could fit on a floppy disk if floppy disks were still around, which they aren't, but it's nice to know.

Second, phoneme-level output gives us more precise control over what counts as a filler. *Um* is not really a word. It's a sound (a mid-central vowel followed by a bilabial nasal, to be exact). By working at the phoneme level, we can define our detection target in terms of what it actually sounds like, rather than hoping a language model's tokenizer agrees with us about what constitutes a word.

2.2 Listening

When the user presses Start, the system requests microphone access and begins capturing audio at 16 kHz mono. The audio is muted on the output side so the user doesn't hear their own voice through the speakers. This is for everyone's comfort.

2.2.1 Calibration

The first thing the system does is listen to the ambient noise of your room for about a second (such as air conditioning, a fan, or traffic outside). It measures the energy level of this background noise so it can later tell the difference between you speaking and your environment.

The speech detection threshold is set at 3.5 times the measured background noise level, with a minimum floor so that very quiet rooms don't produce a threshold so low that breathing triggers it. This was added because early prototypes were quite responsive to sighing, which, in the context of interview practice, happens more often than you might expect.

2.2.2 Detecting when you're talking

After calibration, the system classifies each incoming audio chunk as either speech or silence. In practice, this step is implemented with **Silero VAD 6.2**, using the calibration pass only to estimate the ambient noise conditions at startup. When speech is detected, the system starts collecting audio for transcription.

Very short segments (under 180 ms) are discarded. They are almost always a creak in the chair or a breath, not meaningful speech. To our knowledge, no one has ever produced a meaningful *um* in under 180 milliseconds, though we remain open to the possibility.

2.3 Detection

Once the model returns a list of phoneme-words for a segment, the detection logic checks each word against a simple rule:

> An *um* is exactly two phonemes:
> - a mid-central-ish vowel, followed by
> - /m/
>
> That is the detector.

2.3.1 The vowel set

The interesting part is defining "mid-central-ish." In theory, the canonical vowel in *um* is /ʌ/ (roughly the vowel in *strut*). In practice, the model does not always output exactly /ʌ/. Depending on the speaker's accent, microphone quality, and room acoustics, it may instead produce /ə/ (schwa), /ɒ/, /ɜ/, or a few other nearby vowels.

We therefore accept a family of eight vowels occupying roughly the same neighborhood of the vowel space:

/ə ɒ ʌ ɘ ɜ ɵ ɜ˞ ʏ/

This set was determined empirically: we recorded ourselves and others saying *um* under a variety of conditions and observed which vowel tokens the model produced. It was an enjoyable afternoon. The resulting set is broad enough to capture natural variation, but narrow enough to avoid common words such as *arm* (/ɑːr m/), *him* (/hɪm/), and *room* (/r uːm/), whose vowels occupy rather different parts of the chart.

Before matching, we also strip diacritics and length markers from the model output, since these may vary without changing the underlying vowel identity.

2.3.2 Catching *um*s early

We don't wait for a speech segment to finish before looking for *ums*. While the user is still talking, the system runs partial transcriptions every 900 ms on the most recent 2 seconds of audio. This means that we can detect an *um* before the sentence in which it lives has finished. To avoid false alarms from imperfect decoding, a partial detection only counts if the same *um* candidate appears in two consecutive partial passes.

There's also a 1.25-second cooldown between detections. This prevents a single drawn-out *uuummm* from being counted as three separate incidents, and gives the user a moment to process the feedback before the next one arrives. We believe in firm but measured consequences.

When the full segment finishes, the system does a final reconciliation: if the complete transcription contains *um*s that the partial system missed, those are caught and accounted for at this stage. No *um* goes unrecorded.

3 Selected user feedback

> *"I didn't ask for this."*
> — Daniel R., participant

> *"The system appears to work primarily by damaging the user's confidence in a targeted and efficient way."*
> — Priya N., evaluation volunteer

> *"I appreciated the system's commitment to improvement over dignity."*
> — Michael T., returning participant

> *"It is reassuring to know my shortcomings can be detected in real time by a machine."*
> — Elena K., respondent

> *"I improved measurably, although not in a manner I would describe as humane."*
> — James L., pilot participant

4 Conclusion

We built a system that tells you when you say *um*. The model is small and the feedback is immediate. In informal testing, users became aware of their *um*s within seconds. They did not thank us for this.

We release the system with no login wall and no analytics, in the hope that it is useful and in the knowledge that it is at minimum annoying. This is, to our knowledge, the correct amount of technology to apply to this problem. We encourage future researchers to apply similarly restrained judgment. Future work may extend detection to *uh*, *like*, and *you know*, though we note that modeling the full space of verbal hesitation may require confronting deeper questions about human communication that we are not qualified to answer and did not sign up for.