Episode Book

EPISODE #5982

Teaching a Voice Engine to Say "Yalla" Right

PUBLISHED	15 October 2026 at 08:48 UTC
DURATION	27:22
LISTEN	myweirdprompts.com/episode/romanized-hebrew-token-classifier-61085/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-multilingual-v3 (chatterbox-local)
PIPELINE	V5.4
SOURCES	11 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:42 UTC

SYNOPSIS

Daniel built a small token classifier and published it on Hugging Face for one reason: this podcast kept mangling his second language. The model finds Hebrew words written in Latin letters inside English sentences so a voice engine can pronounce them properly. But spotting romanized Hebrew is harder than it sounds — there's no single transliteration standard, the vowels often aren't there to find, and words like chutzpah sit in a gray zone between two languages. This episode unpacks how token classification actually works, what span F1 of 0.941 on unseen words really means, and why a correctly detected word and a correctly pronounced word are two completely different achievements.

TRANSCRIPT



Corn

Yalla, let's get this balagan sorted.



Herman

That's the whole problem in six words, right there. A voice engine reads that line and says yalla like it rhymes with Calla, and balagan like it's some kind of pastry.



Corn

Which is exactly what's been happening on this feed for months. Daniel got tired of hearing us butcher his second language, so he built something about it. He published a small model on Hugging Face, a token classifier that finds Hebrew words written in Latin letters sitting inside English sentences, and the reason he built it is this show. He wants us saying the odd Hebrew word properly instead of flattening it into English.



Herman

And then he asked the three questions underneath it. How does a model like that actually work, why is spotting romanized Hebrew harder than it sounds, and what does it take to get from we found the word to the voice engine saying it right.



Corn

So let's start with the part that sounds simplest and isn't. What a token classifier is actually doing when it reads a sentence.



Herman

The name gives it away, and then it misleads you. Token classification gives a label to every token in the sentence. The classic example is named entity recognition, where you're tagging people and places, so every token carries a B-person, an I-person, or an O for outside any entity. The B marks the start of a span, the I marks continuation, O means nothing's here.



Corn

And Daniel's model is doing the same thing with a different tag set. B-HE and I-HE instead of B-person.



Herman

Right. It's not translating, it's not identifying what language the sentence is in, and it's not synthesizing speech. It finds a span, and it hands that span to something else. That distinction matters, because most of the coverage of this kind of tool blurs all three into one word, language detection, and they're completely different jobs.



Corn

Walk me through the split, then.



Herman

Two stages. Stage one is detection. Find the romanized Hebrew span, return its character offsets. Stage two is pronunciation. Turn that span into phonemes the voice engine will accept. Daniel's model is stage one, and it's the tidy part. Stage two is where the unglamorous work lives.



Corn

And the evaluation criterion isn't what you'd expect from a paper. It's whether a listener who speaks Hebrew winces or doesn't.



Herman

That's the honest metric. You can have a beautiful F1 score and still produce a listening experience that makes a native speaker flinch, because a correctly detected span and a correctly pronounced word are not the same achievement. That gap is the whole episode, really.



Corn

There's also a personal stake here that isn't abstract. This is a model built for a podcast. It's built for us, for this room, so the words we've been mangling come out right.



Herman

And I'll be honest, I noticed the yalla problem months ago and did nothing about it. You flag a mispronunciation once, you think someone will fix it in the pipeline, and then sixty episodes go by.



Corn

Fitting, for the two of us.



Herman

It is.



Corn

And the first thing to get straight is that the model isn't reading words. It's reading tokens, which is not the same thing.



Herman

That's the mechanical detail everybody skips, and it's where a word-spotter lives or dies. Tokenizers don't hand you words. They hand you subwords, and sometimes they hand you pieces smaller than a word. In the Hugging Face tutorial, the handle at-paul-walk gets split into three tokens. The at sign on its own, paul, and then walk with a double hash in front of it, which is the convention for saying this is a continuation of the previous token.



Corn

So the model never sees the word as a unit.



Herman

Never. It sees a sequence of fragments, and it's your job to realign your labels onto those fragments. The tutorial maps tokens back to words with a function that returns word identifiers, then labels only the first subword of each word. Everything after that, and every special token the tokenizer adds, gets a label of minus one hundred, and that value tells the loss function to ignore it entirely. Not penalize it. Ignore it.



Corn

Why does that matter so much for a spotter specifically?



Herman

Because if you label every subword of balagan and train on that, you are training the model to fire on fragments. It learns that the sound bah is a Hebrew signal, and then it lights up in the middle of a perfectly English word that happens to contain those letters. You've built a detector that's technically right about the span and useless in practice, because it's also flagging half of an English sentence. So the discipline is, one label per word, first subword, ignore the rest.



Corn

And the span offsets are what Daniel's model returns.



Herman

Character offsets, so the downstream code knows exactly where to cut. That's the handoff format. Not a transcript, not a translated sentence. Two numbers per span, start and end.



Corn

Then how big is the training run here? Because you called this a small model.



Herman

It is small, and the tutorial numbers are a good yardstick. Two epochs, a learning rate of two times ten to the minus five, batch size sixteen. That's a weekend fine-tune of an existing encoder. You are not pretraining anything. You're taking a model that already understands language reasonably well and teaching it one narrow tagging convention on top.



Corn

Which is a thread we've pulled before, the fine-tuning mechanics.



Herman

We spent a whole episode on how fine-tuning actually works, and the punchline then is the punchline now. The base model does the understanding, the fine-tune does the steering. Two epochs is enough because you're not teaching it Hebrew. You're teaching it where the Hebrew is.



Corn

That's the entire trick, isn't it.



Herman

It is. Which is also why Daniel's numbers are believable. He fine-tuned xlm-roberta-base, an encoder that's already multilingual, with three labels. O, B-HE, I-HE. The model has one job.



Corn

Let's put the numbers on the table, then argue about what they mean.



Herman

Span F1 of zero point nine four one on a term-disjoint test set. Precision zero point nine two eight, recall zero point nine five three. And zero false positives on fifty-eight hard negatives.



Corn

Define term-disjoint for me, because that's the number that actually matters here.



Herman

It means the test set contains words that never appeared during training. Not the same sentence with a different word order. Words the model has literally never seen. That's the hard version of the evaluation, and it's the one you have to do, because the whole point of a spotter for a living language is that it meets new loanwords every week. If your test set shares vocabulary with your training set, your score is decoration.



Corn

So ninety-four percent on words it's never seen.



Herman

On words it's never seen. And there's a wrinkle in how that score is computed that's worth mentioning. The standard tooling reports precision, recall, F1 and accuracy, and it's measuring span-level correctness. Did you mark exactly the right boundaries. And span-level correctness is not the same as pronunciation correctness. A model can score well and still produce a bad listening experience, because the thing we actually care about is what comes out of the speaker.



Corn

Hold that thought, because I want to get to why the target is hard before we get to the pipeline. What's actually in these sentences that makes this nasty?



Herman

The first problem is that there's no standard. Not a weak standard. No standard. There are competing ones, and Wikipedia's article on romanizing Hebrew lists them out. ISO two five nine, the Society of Biblical Literature system, the Academy of the Hebrew Language's nineteen fifty-three scheme and then its two thousand six revision, BGN slash PCGN, the Artscroll convention, and then the one that actually describes how people write, common Israeli informal. And the article says it plainly. In general usage there are often no hard and fast rules in Hebrew-to-English transliteration.



Corn

No hard and fast rules. That's the sentence to tape to the wall.



Herman

And the consequence is that the same Hebrew letter lands on different Latin letters depending on who's writing and which tradition they absorbed. The letter het becomes ch if you grew up with German or Yiddish influence, or h with a dot under it if you're being softer or more Mizrahi or more ancient. The letter tav becomes th in Beith, s in talis, t in Bet.



Corn

One letter, three outcomes, and they're all correct.



Herman

The letter qof is c in Isaac, k in Yitzchak, q in Qehila. The letter tsadi is s, or tz, or ts. And the illustration sentence on that page is the whole problem in one line. For Hanukkah at the synagogue Beith Sheer Chayyim, Isaac donned his talis that Yitzchak sent him from Bet Qehila in Tsfat.



Corn

Say that again slowly, because I heard the same word twice.



Herman

You did. Beith and Bet are the same word, romanized two different ways, in one sentence, both of them perfectly defensible. If you train a model on one convention, it misses the other, and it misses it in the exact register where it matters, which is people writing quickly.



Corn

And Hanukkah itself, H with a C or just an H.



Herman

Chanukah with a C, Hanukkah without, and both are in circulation. This is the data-coverage problem in its purest form.



Corn

The vowels are worse, though. That's the part I keep coming back to.



Herman

Much worse, and it's structural. Hebrew is normally written without vowel marks. They exist, but everyday text doesn't use them. So a romanized form has to guess vowels that aren't in the source. The schwa is the clean example. It can be sounded, or silent, and Israeli Hebrew typically demotes the sounded one toward the silent one, which produces consonant clusters that a reader has to resolve by feel.



Corn

So the romanization is lossy at the level of the phoneme.



Herman

Lossy at exactly the level the detector would need. The detector can find the word correctly, and the pronunciation stage still has to guess, because the information was never there. That's not a bug in Daniel's model. That's a property of the writing system.



Corn

Mine. Balagan. Sababa. Those are easy because we've heard them. Now put a word in front of me I haven't heard and I'll get the vowels wrong nine times out of ten, because I'm guessing.



Herman

And that's the humbling part of the whole discipline. The string doesn't tell you. It never did.



Corn

Then there's the gray-zone problem, which isn't linguistic at all. It's definitional.



Herman

Yalla is borrowed from Arabic, and it means let's go, hurry up. Balagan means a mess. Chutzpah means nerve, or audacity, and it's in English dictionaries and English speech. So the model is being asked a question that isn't a fact about the string. Is chutzpah an English word now, or a Hebrew one?



Corn

And there's no right answer, because it depends on the speaker.



Herman

It depends entirely on intent. An American saying chutzpah in a sentence about a lawsuit is speaking English. Daniel saying it in this room is speaking Hebrew. Same letters, same sounds, different job. So a model that flags it will sometimes be wrong by design, and you can't fix that by training harder. You can only decide which error you'd rather make.



Corn

Flag everything, and you sometimes override a word the speaker meant as English. Flag nothing, and you're back where we started.



Herman

That's the trade, and it's a judgement call, not a definition. Formally, all of this is code-switching detection, and the literature confirms it's hard.



Corn

Give me the state of the art.



Herman

There's a model called MixLID that landed on arXiv just this week, submitted the twelfth of October. It's a token-level language identifier covering one hundred and eleven language and script combinations. And its authors make the point that justifies the entire genre. A single sentence-level label cannot distinguish the languages used in different spans of the same input. Which is exactly our problem. Our sentence is English, except it isn't, three words of the way through.



Corn

And the scoring?



Herman

On the code-switching benchmark, MixLID reaches ninety-three point two percent accuracy and eighty-eight point eight macro F1. That's the best in class. The comparison is what's telling. Polyglot Tagger lands at seventy-one point six and fifty-three point nine. CLD3 at fifty-four point six and thirty-three point nine.



Corn

So the drop from ninety-three to seventy-one to fifty-four is the cost of doing it per-token instead of per-sentence.



Herman

That's exactly what it's showing. Sentence-level language ID is basically solved. Token-level language ID is not, and the numbers fall off a cliff when you demand the finer granularity.



Corn

Which is the position Daniel's model is standing in. That's the neighborhood.



Herman

It is. Though I want to flag something, and this is an honest negative finding rather than a gap in the episode. We could not find a standalone, publicly documented model whose specific job is detecting romanized Hebrew inside English text. Nobody has published that.



Corn

Which is why he built one.



Herman

The closest analogues are MixLID, which handles Hebrew script but as script, not romanized Hebrew sitting in English. There's a model called AnE-LID doing token-level English code-switch detection with classes for English, not English, mixed and other. And then there's a literature running the opposite direction entirely. A paper from two thousand eight on identifying transliterated foreign words in Hebrew script. Foreign words inside Hebrew letters, rather than Hebrew words inside Latin letters.



Corn

And its numbers?



Herman

Precision and recall of eighty and eighty-two on just over four thousand unique words, using a purely statistical method with no lexicon. So the task is studied, it's non-trivial, and it's been approached from the other side of the mirror.



Corn

So Daniel's model is filling a hole that the field left open.



Herman

More or less. The pieces exist. The assembly is his.



Corn

There's one more detail I want on the table, because it's the trick that actually solves the standards problem. How do you train a model to catch Beith and Bet and Chanukah and Hanukkah when they're all correct?



Herman

You manufacture the coverage. MixLID's paper describes exactly this. They augment their training data by substituting English source words with their loanwords, using etymological data to find the pairs. So you take a sentence, you swap in the borrowed form, and now the model has seen both versions. That's the answer to fragmentation. You don't pick a standard and hope. You train on all of them and let the model learn the family resemblance.



Corn

Which for Daniel would mean generating the variant spellings artificially.



Herman

Generating them, yes. If you only had what appeared in the wild, your coverage would be whatever happened to get written down. The augmentation step is what turns a nice idea into something that survives real text.



Corn

And that's the model, more or less. Now the piece that isn't the model.



Herman

Right.



Corn

So say the model works perfectly and finds every romanized Hebrew word. You've got a span. Now what?



Herman

Now you have to turn a string into a pronunciation, and this is where the work stops being tidy. The bridge is called grapheme-to-phoneme conversion, G2P. And the reason it exists is a specific failure. NVIDIA's own documentation says it out loud. TTS models trained on raw graphemes fail to provide a reliable way of correcting wrong pronunciations.



Corn

Which is a polite way of saying you can't tell it what to say.



Herman

You can't tell it what to say, so you hand it phonemes instead, and then the engine has no choice. The documentation's phrasing is that you enforce the desired pronunciation by providing a phonetic transcript. And the case G2P is built for is out-of-vocabulary words, and their examples are proper names and loaner words. Which is precisely our case. Balagan is not in the English dictionary of the voice engine. It has never seen it.



Corn

So the engine would guess from spelling.



Herman

It would guess from English spelling rules, and it would be wrong, because the string balagan looks like an English word and isn't. You can't spell your way there from English rules.



Corn

Fine. What are the models that do this conversion?



Herman

There are two families in NeMo, and the tradeoff is the familiar one. There's a ByT5 text-to-text model, which is the bigger, more flexible one. And there's a G2P Conformer, a smaller CTC model with about twenty times fewer parameters and non-autoregressive inference, which means it runs faster because it doesn't generate the output token by token.



Corn

Twenty times fewer parameters, faster, and presumably slightly worse.



Herman

Slightly worse, and cheaper, and easier to run. That's the trade every time. The larger model handles stranger inputs better. The smaller one is the one you'd actually deploy if you were doing this at scale.



Corn

Alright. So we've got a way to turn the word into phonemes. Then you have to get the voice engine to accept them, and this is where I expect it to get ugly.



Herman

It gets very ugly, because there is no portable way to say this word sounds like this. Every vendor invented its own.



Corn

Give me the tour.



Herman

Microsoft's Azure Speech uses SSML with a phoneme tag. You set the alphabet attribute, and it accepts IPA, or SAPI, or UPS, or x-sampa. And you put the phonetic string in the ph attribute. Then there's a separate custom lexicon, which is a PLS one point zero XML file, capped at one hundred kilobytes, and it's cached for fifteen minutes.



Corn

A hundred kilobytes. That's not a lot of words.



Herman

It's a few thousand entries, which for a podcast is plenty and for a product is nothing. And the cache means a pronunciation change may not take effect for a quarter of an hour, which is a fun thing to discover at midnight before you publish.



Corn

Cartesia, meanwhile.



Herman

Cartesia's pronunciation dictionaries are, in their own description, a simple search and replace. You can give it IPA, or you can give it sounds-like guidance. Their example is a word spelled out as chop-uh-TOO-liss, with the emphasis in capitals. And you apply it by referencing a pronunciation dictionary identifier.



Corn

So one vendor wants a phonetic alphabet, and one accepts me spelling it how it sounds.



Herman

And there are at least four more doing slight variations on the same thing. Telnyx, Fish Audio, FluidAudio, Async. Per-word IPA overrides, custom lexicons, inline phoneme tags. Same idea, different syntax, different file format, different limits.



Corn

Which means every pronunciation fix has to be re-expressed for every engine.



Herman

Every time. And if Daniel ever switches TTS vendor, the entire pronunciation layer has to be rebuilt from scratch. That's not a technical problem, it's a portability problem, and it's the one that costs the most human hours.



Corn

Suddenly the model looks like the easy part.



Herman

The model is the easy part. That's the thing I'd want a listener to take away. The detection is the fun, publishable, benchmark-scoring part. The pronunciation override is the part nobody writes about, and it's where the time goes.



Corn

There's a workflow detail in the NVIDIA guidance that I want to pull out, because it says something about who this serves.



Herman

Their recommendation for modern models is that they use IPA, not the old ARPABET or CMU conventions. And the workflow they suggest is, generate IPA candidates, synthesize the variants, play them for the user, and let the user pick.



Corn

A human.



Herman

A human in the loop. And I'd argue that's correct rather than lazy. Balagan with the stress in the wrong place is a different word to a native speaker. You can't always score that automatically, and you shouldn't pretend you can. Somebody has to listen.



Corn

That's the honest version of the pipeline. It's not fully automated.



Herman

It's not, and the guidance says so.



Corn

Now here's the thing I want to nail down, because it's the insight of the whole episode. Detection accuracy and pronunciation accuracy are separate failure modes.



Herman

Completely separate, and they fail in opposite directions. Failure one is a miss. The detector doesn't flag balagan, so the word goes to the voice engine as English, and it comes out flattened. That's the original problem, untouched. Failure two is the one people don't anticipate. The detector finds the word correctly, hands it to the pronunciation stage, and the pronunciation is still wrong, because Hebrew omitted the vowels and the romanization was lossy and something had to guess.



Corn

So a perfect detector can still produce a wrong-sounding word.



Herman

A perfect detector can absolutely produce a wrong-sounding word. The model did its job. The span is correct. And the listener still winces, because the vowels came out wrong, or the stress landed on the wrong syllable, and that's a different stage's problem that the model never had the information to solve.



Corn

Which kills the fantasy that a high score means a good result.



Herman

It kills it dead. And the reverse is also true. A missed detection doesn't care how good your pronunciation layer is. If the word never gets flagged, it never reaches the override.



Corn

So the two halves are chained, and the weak one sets the ceiling.



Herman

The weak one sets the ceiling, and you can't tell which one is weak by looking at a benchmark. That's why the value of Daniel's model isn't the F1 number. It's that it moves one of the two stages from approximate to reliable, and that's the right place to put the effort, because everything downstream of a miss is wasted work.



Corn

And in the pipeline Daniel actually built?



Herman

Each detected span gets converted to vocalized Hebrew script, and then rendered with Chatterbox Multilingual V3 with the language set to Hebrew, while the rest of the line stays English. Then the pieces are stitched back together.



Corn

Which is a clever dodge, actually. You don't fight the engine's pronunciation rules. You hand it a different language.



Herman

You hand it a different input entirely. The engine isn't guessing at a Latin string anymore, it's reading Hebrew with vowel marks, and the vowel marks are the thing the romanization was missing. That's the fix for the lossy step. Convert from the lossy form back into the form that has the information.



Corn

Right. So where does that leave the pipeline?



Herman

Three stages. Detection, grapheme to phoneme, vendor-specific override. Each one with its own ceiling. And the end experience is the product of all three, which means you can't fix a bad result by improving one of them.



Corn

What was it you said earlier? We found the word is maybe forty percent of the work.



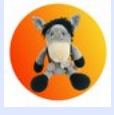
Herman

I'd stand by that, and the forty percent is the percentage that gets the attention.



Corn

One thing I keep turning over. As the voice engines get better at switching languages natively, does the detector become less necessary, or more?



Herman

That's unsettled, and I don't want to pretend I know. The optimistic version is that the engine hears Hebrew and renders it Hebrew and you don't need a spotter at all. The pessimistic version is that the engine's default guess is still wrong for exactly the gray-zone words, chutzpah and yalla and sababa, because those words have two live pronunciations and the engine has to pick one. And if it picks English, the detector is the only thing that saves you.



Corn

And the pipeline doesn't get shorter. It gets more important.



Herman

It does.



Corn

Meanwhile our own feed has been saying yalla wrong for months.



Hilbert

Which word was it you were just saying wrong?



Corn

Yalla.



Hilbert

That one's not the interesting one. The interesting one is sabich.



Herman

Go on, Hilbert.



Hilbert

Everyone in this building says sabich with the stress on the first syllable. Sah-bich. There's a man on Agrippas Street who owns a stall, and he has corrected me about this eleven times, and he's wrong every time.



Corn

He's wrong.



Hilbert

He is wrong, and I've been eating there since before his awning was replaced. The stress goes on the second syllable. Sa-BICH. I've heard it from people who ought to know. I said this to him in the summer, politely, and he told me the whole thing was a Yiddish word anyway, which is nonsense, and then he charged me full price for the pita.



Herman

Full price.



Hilbert

Not a shekel off. Which I paid, because the salad is good, but I wrote him a note.



Corn

You wrote him a note.



Hilbert

Two pages. I typed it, so it's legible. I explained the stress pattern with examples, and I proposed we settle it with his father, who is still alive, and who he says disagrees with me. So I called the father. The father agrees with me.



Herman

The father agrees with you.



Hilbert

The father agrees with me and asked me not to come by the stall on Fridays, because that's his busy time and he doesn't want the argument in front of the customers. So I go on Mondays now. Last Monday it was eighteen shekels for the same pita the son charges twenty-two for, so I'm ahead on the arrangement, and the son still won't say the word the right way.



Corn

Herman, is he right about the stress?



Herman

Honestly? I don't know that word well enough to say, and I'd want to look it up before I contradicted a man who has been eating there since before the awning.



Hilbert

The vowel's fine. It's the stress. Nobody listens to the stress.



Corn

And the son still hasn't moved.



Hilbert

He's moved the pita to eighteen shekels on Mondays. That's not nothing.



Herman

So where does that leave the pipeline?



Corn

It leaves it honest. Three imperfect stages chained together, and the best you can do is make each one reliable enough that the next one isn't guessing about the wrong thing.



Herman

And the open question underneath it is whether native code-switching in the engines makes the detector obsolete or makes it more important, because if the engine's default guess for a loanword is still English, the override layer is the entire game.



Corn

Which is the bit that doesn't have a benchmark. It has a wince test.



Herman

It has a wince test, and the wince test is the one that ships.



Corn

And the shape of this recurs everywhere. Anywhere you're trying to make a machine say a borrowed word right, you've got detection feeding pronunciation, with a lossy step in the middle that nobody warns you about. Daniel's model is small. The problem isn't.



Herman

That's the part I'd want people to carry out of the episode. The F1 number is the easy headline. The vowel marks were the actual discovery.



Corn

Our thanks to producer Hilbert Flumingtop. If you want more of this, try episode eight, Building Your Own Whisper; episode four, If Your Voice Ages, Does Your Fine-Tune Become Useless; and episode eleven, How Does Fine Tuning Work Anyway. This has been My Weird Prompts, the human-AI collaboration podcast. The model is on Hugging Face, the episodes are at my weird prompts dot com, and if you've built something similar, or you've got a word your voice engine keeps mangling, send us your own prompt on



Corn (continued)

Telegram at t dot me slash MWP listener bot.



Herman

We'll be back soon.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **Hugging Face**
   Token classification task guide (primary)
   https://huggingface.co/docs/transformers/en/tasks/token_classification

2. **Wikipedia**
   Romanization of Hebrew (last edited rev. 1217740816)
   https://en.wikipedia.org/wiki/Hebrew_transliteration

3. **arXiv 2610.16601v1**
   MixLID: Token-Level Language Identification in Mixed-Language Text, Hong et al., submitted 2026-10-12
   https://arxiv.org/abs/2610.16601v1

4. **NVIDIA NeMo-Speech**
   Grapheme-to-Phoneme Models (primary)
   https://docs.nvidia.com/nemo/speech/latest/tts/g2p.html

5. **Microsoft Learn**
   Pronunciation with SSML (primary)
   https://learn.microsoft.com/en-us/azure/ai-services/speech-service/speech-synthesis-markup-pronunciation

6. **Cartesia Docs**
   Custom Pronunciations (primary)
   https://docs.cartesia.ai/build-with-cartesia/capability-guides/custom-pronunciations

7. **Hugging Face**
   igorsterner/AnE-LID (token-level English code-switch detection)
   https://huggingface.co/igorsterner/AnE-LID

8. **Springer**
   Identification of Transliterated Foreign Words in Hebrew Script (2008)
   https://link.springer.com/chapter/10.1007/978-3-540-78135-6_40

9. **Jewish Journal**
   Chutzpah' Explains Israel's Spirit of Innovation
   https://jewishjournal.com/culture/arts/books/305792/chutzpah-explains-israels-spirit-of-innovation/

10. **Times of Israel (blogs)**
Basic Hebrew Phrases and Slang You Should Know in Israel
https://blogs.timesofisrael.com/basic-hebrew-phrases-and-slang-you-should-know-in-israel/

11. **arXiv 1904.01989v1**
Subword-Level Language Identification for Intra-Word Code-Switching, Mager et al., NAACL 2019
https://arxiv.org/abs/1904.01989v1