LIBRARY

# The Hebrish technical library

The method notes and white papers in one volume

AUTHOR
**Daniel Rosehill and Opus 5.5**

DATE
**02/11/2026**

Hebrish is Daniel Rosehill's personal project: speech recognition and speech synthesis models for English with Hebrew words in it, the way English speakers in Israel talk. This volume collects its technical documents. Each is also published on its own at hebrish.app/technical#library, with the date it was written; dated documents keep their original wording, so newer results are on hebrish.app/benchmarks. The release-1 white paper is also available in Hebrew.

| Document | Type | Date |
|---|---|---|
| **How the Hebrish training sentences are made**<br>Where the words and sentences come from, how they are generated and rejected, spelling, and how the test splits are kept apart. | Method note | 02/11/2026 |
| **Hebrish model release 1: Jachnun v1.0 and Sabich v1.0**<br>The first stable models: dataset, training, exports, evaluation and limits, with Shakshuka as work in progress. A dated record; later results are on hebrish.app/benchmarks. | White paper | 21/10/2026 |
| **Deterministic English-Hebrew code-switching text-to-speech**<br>Reading English text with romanized Hebrew in it aloud: detect the Hebrew spans, transliterate, add niqqud, and send each segment to TTS with its language. | White paper | 20/10/2026 |

# Contents

How the Hebrish training sentences are made . . . 3
In short . . . 3
Where the words come from . . . 3
Where the sentences come from . . . 3
How a sentence is generated . . . 4
Rejection policy . . . 5
Spelling and transliteration . . . 5
How the dataset grew . . . 6
Splits and testing . . . 6
Hebrish model release 1: Jachnun v1.0 and Sabich v1.0 . . . 8
1. Release 1 at a glance . . . 8
2. The problem . . . 9
3. Dataset . . . 9
4. Two recipes from one base . . . 10
5. Training . . . 11
6. Exports . . . 11
7. Evaluation . . . 11
8. Results . . . 13
9. In progress: Shakshuka, the streaming line . . . 16
10. Limitations . . . 17
11. Compute . . . 17
12. Links . . . 17
Deterministic English-Hebrew code-switching text-to-speech . . . 19
1. The problem . . . 19
2. Why standard tools miss it . . . 19
3. The pipeline . . . 20
4. Worked examples (real output) . . . 21
5. Deterministic, or a model that switches by itself . . . 21
6. The contract between stages . . . 22
7. What is open . . . 22
References . . . 22

METHOD NOTE

# How the Hebrish training sentences are made

Daniel Rosehill and Opus 5.5 · 02/11/2026 · hebrish.app/whitepaper/hebrish-training-data.pdf

## In short

Hebrish models learn Hebrew words as English speakers say them inside English sentences (*makolet*, *arnona*, *Bituach Leumi*). There was no corpus of that kind of speech, so one was built. Its sentences are short English sentences, each written around one or two target Hebrew terms. Most are written by large language models from a target term and its context, checked by rules and a second model, then read aloud by a person. A sentence a person would not say is rejected at any of those steps. No audio is synthetic: every clip is a human reading.

The recordings are one speaker's voice (Daniel Rosehill's). More voices are welcome: see hebrish.app/get-involved.

> **Numbers in this note**
> Counts come from the dataset's metadata (`meta/train.jsonl`, `meta/test.jsonl`) on Hugging Face, `danielrosehill/English-Hebrew-Codeswitching-Sentences` at revision `ad2f62c6`, read on 2 November 2026: **6,812 clips, 717.0 minutes**. The dataset grows most days, so later revisions differ.

## Where the words come from

The vocabulary is the Hebrish lexicon: **3,642 Hebrew terms in 13 domains** (everyday life, bureaucracy, religious life, the army, food and more), browsable at hebrish.app/lexicon. Each term has one canonical romanized spelling, the way English speakers in Israel commonly write it, and a short meaning. The first list of about 1,100 terms was expanded from Daniel's earlier sentence dataset and dictation history with LLM suggestions; it has grown since through themed vocabulary packs (news, Jewish texts, everyday gaps, Ashkenazi pronunciation).

## Where the sentences come from

| Source | Clips | Share |
|---|---|---|
| Generated in bulk by an LLM from a target term and its context (the sentence bank) | 4,357 | 64.0% |
| Generated on demand: Daniel picks the word in the recording app (optionally with a context hint) and an LLM writes the sentence | 1,763 | 25.9% |
| Taken from existing datasets, mainly Daniel's earlier English-Hebrew-Mixed-Sentences (262) | 606 | 8.9% |
| Typed or written by Daniel himself | 86 | 1.3% |
| **Total** | **6,812** | **100%** |

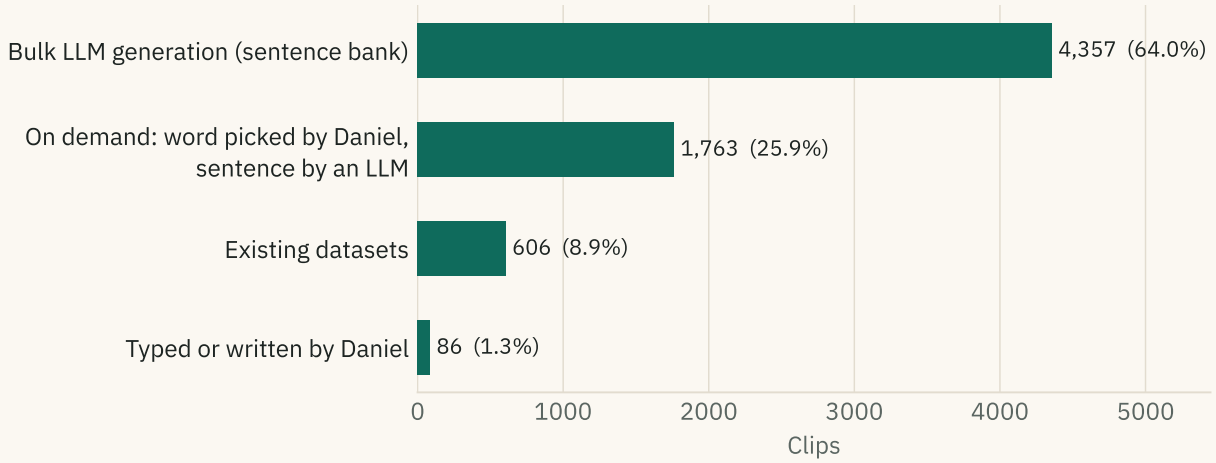



Figure 1: Where the sentence text of the 6,812 clips came from. (`prompt_origin` and `prompt_source`, dataset revision `ad2f62c6`, 02/11/2026)

Field `prompt_origin` in the metadata holds the source class; `prompt_source` names the generator. So about 90% of the sentence text is LLM-written, and for about a quarter of all clips the target word was chosen by hand. An earlier count on the site (19 October 2026) listed 1,403 sentences as "written by hand"; that figure counted the on-demand sentences, whose word is chosen by hand but whose text is generated. This note replaces it.

## How a sentence is generated

The generator gets two kinds of input:

- **The target term**, spelled exactly as in the lexicon, with its meaning. Without the meaning, a pilot with a small model misused about one Hebrew word in four (14 October 2026).
- **Surrounding context**: a register and scenario (a WhatsApp message to a neighbour, an email to an accountant), the vocabulary pack's instruction, and for some terms example phrasings people use, marked "vary them, do not copy".

The prompt requires the term exactly as given (never translated, glossed or written in Hebrew script), used naturally so the context makes its meaning clear, and tells the model to answer `SKIP` rather than force a usage it is unsure of: a wrong usage is worse than a missing one. Sampling temperature is 0.9 to 1, so sentences for the same term differ in structure and setting.

### Models and inference APIs

| When | Generator |
|---|---|
| From 14 Oct 2026 | Claude Sonnet 5.5 through OpenRouter, ten items per request (2,179 clips' sentences). Claude Code subagents and Claude Opus 5.5 wrote the themed packs (1,700 and 216); 57 came from an early DeepSeek V4.1 Flash pilot. |
| From 21 Oct 2026 | Claude Haiku 5.5 through the **Anthropic Message Batches API** (205 so far), moved to the **OpenRouter Batch API** on 1 Nov 2026. |
| From 2 Nov 2026 | Claude Haiku 5.5 through synchronous OpenRouter calls: batch jobs queued for 5 to 10 minutes, too slow for the recording loop. |
| On demand | A fast model through OpenRouter (DeepSeek V4.1 Flash by default, Qwen as fallback), synchronous, because the speaker is waiting for the sentence. |

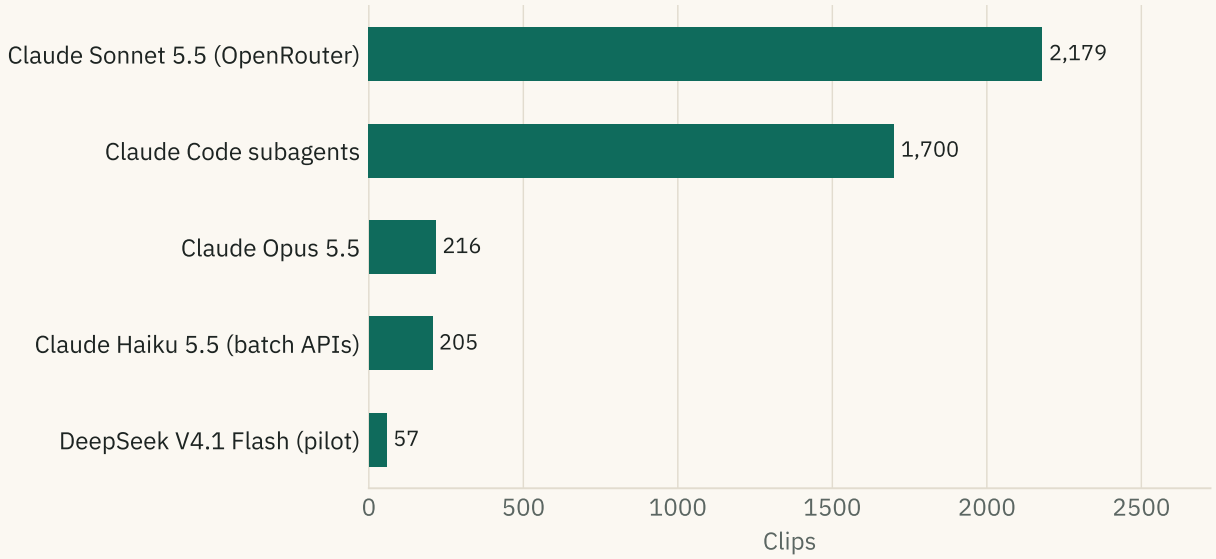



Figure 2: Generated sentences in the dataset by generator (bulk generation only; the 1,763 on-demand sentences are not broken down by model). (`prompt_source`, dataset revision `ad2f62c6`, 02/11/2026)

## Rejection policy

A sentence has to pass three gates before it is trained on.

1. **An automatic gate** (all generators, since 2 November 2026). It rejects a sentence that has Hebrew script, brackets or characters a target model cannot output; is shorter than 6 or longer than 18 words; does not contain each target term exactly once, matched as a whole word; mixes a Hebrew term with a tech term (kept apart so each can be measured); or writes a number of 10 or more in words. A rejected sentence is regenerated.
2. **A reviewer from another model family.** Gemini judges Claude's sentences (temperature 0) and drops any where a Hebrew word is used wrongly or is not one an English-speaking Israeli would drop in there, the sentence does not sound like something a person would dictate, terms are forced together, or it is ungrammatical. A sentence the reviewer cannot rule on after three tries is dropped. From 2 November 2026 this pass is optional, because the third gate catches the same problems.
3. **The reader.** Every sentence is read aloud by a person, who skips any that misuse a word or sound unnatural. Each take is played back on the phone and kept, retaken or deleted. A misread is fixed in the transcript (flag `edited`), so the text always matches the audio. Speaker labels are checked with voice embeddings (hebrish.app/dataset#integrity).

## Spelling and transliteration

Each term is trained in its canonical romanization. Where a term has other common spellings (667 of the first 1,167 terms list some), the bank rewrites them to the canonical form before a sentence is used, so the model learns one target spelling per word. Variants that are ordinary English words are left alone.

The one deliberate exception is pronunciation: **Ashkenazi and Israeli forms** (*Shabbos* and *Shabbat*, *Tosfos* and *Tosafot*) are kept as separate spellings in 93 spelling groups, because they sound different. 792 clips carry that label: 656 Israeli, 135 Ashkenazi, 1 mixed.

**Under consideration**

A wider sample of spellings, generating sentences with the different ways people actually

transliterate a word, could make the model more robust to how users write. If tried, a later dataset revision may prune variants whose transliterations conflict, so each spoken word still maps to one written form.

## How the dataset grew

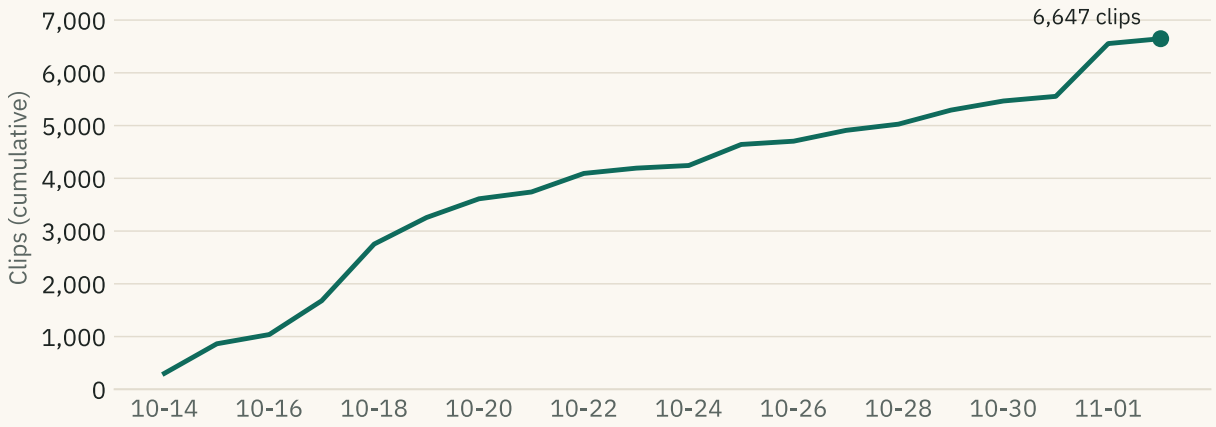



Figure 3: Recorded clips, cumulative, by recording date. (`data/stats.json` on hebrish.app, dataset revision `af910143`, 6,647 clips, read 02/11/2026; the newer revision counted above has 6,812)

## Splits and testing

Terms, not sentences, decide the split. About one term in twelve (by a hash of the term) is held out: any sentence containing a held-out term goes to the test split, so models are scored on Hebrew words they never heard in training as well as on words they did. Words Daniel picks or types himself always go to training.

The public benchmark, **test set v2** (Hebrish/English-Hebrew-Codeswitching-ASR-Benchmark, 25 October 2026), is 129 code-switching clips balanced across the 13 domains, none used by any training run, with 221 term occurrences, 137 of them unseen in training, plus 25 English-only and 25 tech control clips. Full results: hebrish.app/benchmarks.

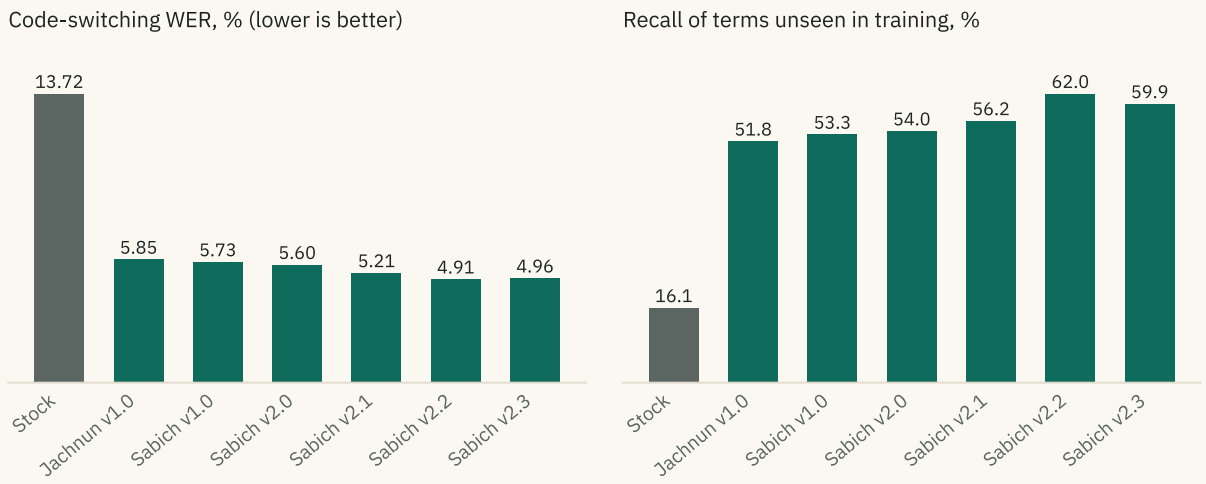



Figure 4: Test set v2: stock Parakeet TDT 0.6B v2 against the Hebrish models trained on this data. Sabich v2.3, the current release, is slightly behind v2.2 on this test set and ahead of it on the held-out split; the differences are small. (`data/results.json test_v2`, benchmark dataset revision `48ec85a301dc`; v2.3 scored 30/10/2026)

Questions about the method: hebrish.app/contact. Dataset card and schema: hebrish.app/dataset.

WHITE PAPER

# Hebrish model release 1: Jachnun v1.0 and Sabich v1.0

Daniel Rosehill · 21/10/2026 · hebrish.app/whitepaper/hebrish-release-1-whitepaper.pdf

Models: `Hebrish/Jachnun` (run `r6-en30`) and `Hebrish/Sabich` (run `r7-all`). Replaces the “Hebrish v1.0” white paper of 2026-10-19 and its 2026-10-20 addendum.

**Abstract**

English speakers in Israel drop Hebrew words into English. English speech recognition has rarely heard those words and writes the nearest English-sounding thing. Hebrish model release 1 is the project’s first stable set of models: two full fine-tunes of NVIDIA Parakeet TDT 0.6B v2, each taught to write every Hebrew word in one canonical romanized spelling (*makolet*, *Bituach Leumi*). **Jachnun v1.0** learned from 339.2 minutes of code-switched read speech plus about 30% English. **Sabich v1.0** learned from the same kind of recordings plus 187.4 minutes of tech and task recordings, and is the model the Hebrish apps recommend. On the public English-Hebrew code-switching benchmark (test set v1, clean subset, 84 clips) Jachnun scores 73.3% Hebrew term recall, 70.1% on terms absent from training and 3.3% WER, and Sabich 73.8%, 68.4% and 3.3%, against 32.6%, 32.5% and 9.5% for stock Parakeet v2. On 422 held-out clips they are level on code-switching (5.42% and 5.30% WER, stock 11.82%), Sabich is better on tech dictation, and plain English did not get worse. A third line, **Shakshuka**, the same recipe on NVIDIA’s Nemotron streaming model, is a proof of concept and not part of this release. All audio, training and test, is one speaker’s voice, so these numbers rank models on that speaker only.

**73.8%**
Sabich term recall, public benchmark (stock v2: 32.6%)

**73.3%**
Jachnun term recall; 70.1% on terms never in training

**3.3%**
WER, public benchmark, both models (stock v2: 9.5%)

**1.54%**
Sabich tech-dictation WER (Jachnun 1.74%, stock 2.51%)

## 1. Release 1 at a glance

**Hebrish** is the project, not a model: English dictation that writes the Hebrew words correctly, free apps for Android, Ubuntu and Windows, and the open models and data behind them. Its model lines are named after Israeli foods. **Model release 1** is the first stable release of the two lines built on Parakeet; each is at version **v1.0**. Before it came five proof-of-concept checkpoints, PoC-1 to PoC-5 (published on Hugging Face as v1 to v5 of `danielrosehill/Parakeet-TDT-0.6B-v2-English-Hebrew-Codeswitching`).

| Model | Base | Trained on | Run · date | In the apps |
|---|---|---|---|---|
| **Sabich v1.0** | **Parakeet TDT 0.6B v2** | **Code-switching plus tech and task recordings (623 min)** | `r7-all` · **2026-10-20** | **Recommended (since 2026-10-21)** |
| **Jachnun v1.0** | **Parakeet TDT 0.6B v2** | **Code-switching plus English (483 min)** | `r6-en30` · **2026-10-19** | **Second choice** |
| Shakshuka PoC-2 | Nemotron Speech Streaming EN 0.6B | Jachnun's recordings and recipe | `r6-nemotron-en30` · 2026-10-19 | Not yet (Section 9) |

Retrains on more recordings will be Jachnun v1.1, v1.2 and so on (likewise Sabich). Shakshuka's first retrain will be Shakshuka v1.0.

## 2. The problem

Many English speakers in Israel speak English with Hebrew words in it: *"grab some bamba from the makolet"*, *"I have to go to Bituach Leumi"*. This is code-switching. English dictation models were trained on English and have rarely or never heard these words, so they substitute the nearest English-sounding word: *Macaulay* for *makolet*. Often the one word that carries the meaning is the one that comes out wrong.

Hebrish keeps English dictation as it was and writes the Hebrew words correctly. "Correctly" needs a definition, because Hebrew words in English text have no single agreed spelling. Hebrish uses one canonical romanized spelling per word, from a public spelling lexicon (`terms.csv`), and writes Hebrew in Latin letters only. Parakeet's tokenizer is a 1,024-token SentencePiece BPE with cased tokens and punctuation but no Hebrew characters, so Hebrew script was never an option; romanization also suits the use case, which is English text.

The base model is NVIDIA Parakeet TDT 0.6B v2 (CC-BY-4.0), a 0.6B-parameter FastConformer TDT model. It is small enough to run on a laptop CPU and a phone, which is where dictation happens.

## 3. Dataset

Training data is Read-Aloud-Voice-Sentences (CC-BY-4.0; access on request since 2026-10-20, see hebrish.app/datasets). Every clip is **one speaker** (Daniel Rosehill, Irish-born, English-speaking, living in Jerusalem) reading a prompted sentence aloud, mostly on one phone, as 16 kHz mono audio with a cased, punctuated transcript and a list of its target terms. Code-switched sentences carry the Hebrew terms; plain-English and task sentences are recorded in the same voice. The table is the split at revision `550bfcaa0f84`, the one Jachnun was trained and first evaluated on.

| Split at `550bfcaa` | Clips | Minutes | Use |
|---|---|---|---|
| Code-switching train (`cs_train`) | 3,224 | 339.2 | Training |
| Code-switching validation (`cs_val`) | 218 | 22.6 | Best-epoch selection |
| Own-English pool | 751 | 95.3 | Training (all of it) |
| LibriSpeech-PC replay pool | 900 | 190.7 | Training (48.5 min drawn) |
| Code-switching test (`test_cs`) | 387 | 43.9 | Evaluation |
| Own-English test | 73 | 9.2 | Forgetting check |
| LibriSpeech test-clean | 200 | 24.6 | Forgetting check (other speakers) |
| Long-form test | 15 files | 44.6 | Long recordings |

### Held-out terms

The split is **term-disjoint for a held-out list**, not for every term. 362 terms are on the held-out list at this revision; a sentence containing any of them goes to test, so those terms never appear in training. A test sentence can also contain other terms that are not on the list, and those can appear

in training. The evaluation therefore reports two numbers: recall on held-out terms (293 occurrences of 118 terms in `test_cs`), which measures generalisation, and recall on test terms also seen in training (278 occurrences), which measures learning. Only the first is evidence that the model writes a word it was never taught.

## 4. Two recipes from one base

Both models are trained from stock `nvidia/parakeet-tdt-0.6b-v2`; neither is trained on top of the other. Only the blend differs. Jachnun learns from code-switched speech and the speaker's plain English. Sabich, named after the pita with a bit of everything, learns from every task the speaker recorded.

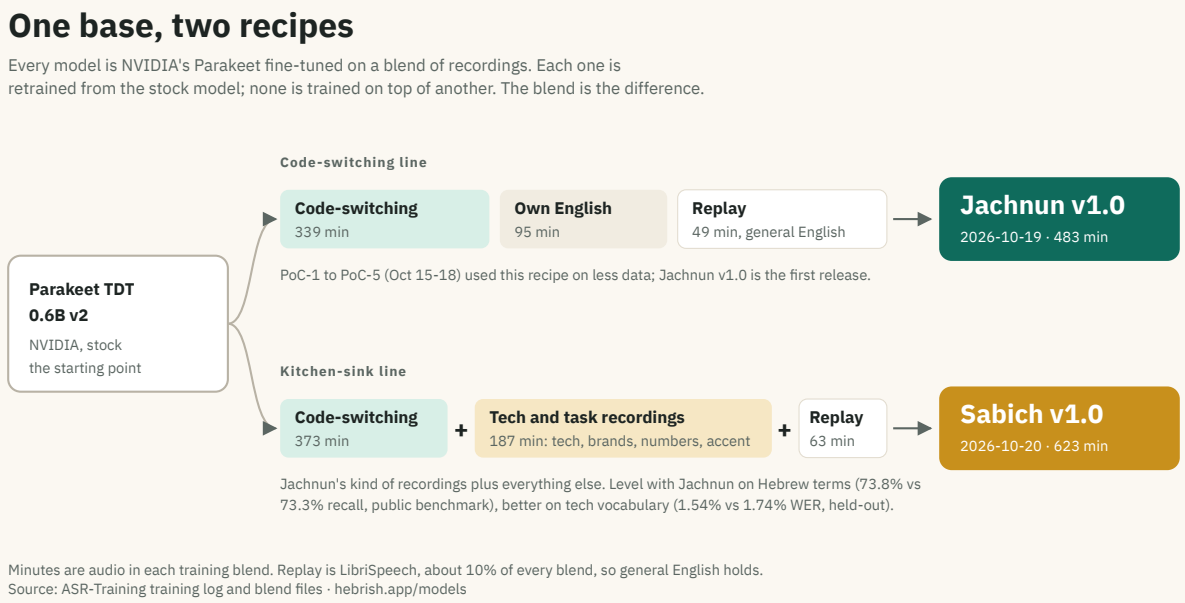



Figure 1. One base, two recipes. Minutes are audio in each training blend.

| Training blend (minutes) | Jachnun v1.0 (`i6-en30`) | Sabich v1.0 (`i7-all`) |
|---|---|---|
| Code-switching | 339.2 | 373.1 |
| Speaker's other recordings | 95.3 (own English) | 187.4 (tech and task) |
| LibriSpeech-PC replay | 48.5 | 62.5 (10%) |
| Total | 483.0 (4,202 rows) | 623.0 |

Sabich's 187.4 minutes of task recordings cover tech vocabulary, brand names, acronyms, numbers and currency, Irish-English, disfluencies, self-corrections and other tasks; with the code-switching that is 560.5 minutes of the speaker's own recordings.

### Why 30% English

A dictation model that learns Hebrew words but loses English is no use. Jachnun's English share was targeted at 30% and came out at 29.8%: the speaker's own English teaches that plain English in this voice should stay plain English, and LibriSpeech replay keeps other voices. The share follows the code-switching and continual-learning literature: fine-tuning only on new-domain data is what causes catastrophic forgetting, and mixing in the original domain plus some replay of general data is the usual remedy. Sabich keeps the replay at 10%; its task recordings are mostly English in the same voice. Section 8 checks whether it worked.

### Why a cumulative full fine-tune, not continued training

Each release starts from stock Parakeet and trains on every take recorded so far. Continuing the previous fine-tune on only new data is what causes catastrophic forgetting, and continuing on all data gives old rows another round of epochs each release, so drift compounds. A run takes under an hour on one GPU, so nothing is saved by continuing. Training and benchmarking are separate steps:

the best epoch is chosen on validation data (part of train, never test), and the benchmark runs afterwards on test.

## 5. Training

| Setting | Jachnun v1.0 | Sabich v1.0 |
|---|---|---|
| Framework | NeMo 2.7.3 (`nemo_toolkit[asr,cu12]`), torch 2.11.0, pinned in a Modal image | |
| Start point | `nvidia/parakeet-tdt-0.6b-v2`, tokenizer kept | |
| Optimiser | AdamW, lr 3e-5, betas 0.9 / 0.98, weight decay 1e-3 | |
| Schedule | Cosine annealing, 10% warmup, min lr = lr / 20 | |
| Epochs / batch | 20 epochs, batch size 16, bf16 mixed precision, seed 7 | |
| Selection | Epoch 18 of 20, validation WER 5.0% (raw, 0.0497) | Epoch 19 of 20, validation WER 4.63% |
| Hardware / time | One NVIDIA L40S (48 GB) on Modal, 41.0 minutes | One NVIDIA L40S on Modal, 52.1 minutes |

### From proof of concept to release 1

| Checkpoint | Run | CS min | Released | Benchmarked |
|---|---|---|---|---|
| PoC-1 | `pilot-en30` | 77.4 | 2026-10-15 | Yes |
| PoC-2 | `r2-en30` | 120.1 | 2026-10-15 | Yes |
| PoC-3 | `r3-en30` | 196.1 | 2026-10-18 | Yes |
| PoC-4 | `r4-en30` | 269.3 | 2026-10-18 | No |
| PoC-5 | `r5-en30` | 294.2 | 2026-10-18 | No |
| **Jachnun v1.0** | `r6-en30` | **339.2** | **2026-10-19** | **Yes, 2026-10-19** |
| **Sabich v1.0** | `r7-all` | **373.1** | **2026-10-20** | **Yes, 2026-10-20** |

PoC-1 to PoC-5 used Jachnun's recipe on less data. Jachnun v1.0 is the sixth checkpoint of that line; it is called v1.0 because it is the first release the project stands by. Sabich v1.0 is not a later checkpoint but a second recipe from stock.

## 6. Exports

Both models are published in the formats on-device runtimes use (Hugging Face, access on request). Jachnun's exports were checked on five clips against fp32 ONNX before any benchmark.

| Format | Repositories | Runs in | Check against full precision |
|---|---|---|---|
| `.nemo` | `Hebrish/Jachnun`, `Hebrish/Sabich` | NeMo (PyTorch) | Reference |
| sherpa-onnx int8 | `Hebrish/Jachnun-sherpa-onnx-int8`, `Hebrish/Sabich-sherpa-onnx-int8` | sherpa-onnx, the Hebrish apps | Jachnun 4/5 |
| ONNX fp32 and int8 | `Hebrish/Jachnun-ONNX`, `Hebrish/Sabich-ONNX` | onnx-asr, Voxtype, the in-browser demo | Jachnun int8 4/5; Sabich fp32 5/5, int8 3/5 |
| GGUF F16, Q8_0, Q4_K_M | `Hebrish/Jachnun-GGUF`, `Hebrish/Sabich-GGUF` | transcribe.cpp, Handy, Offline Voice Input | Jachnun F16 5/5, Q8_0 5/5, Q4_K_M 4/5 |
| MLX | `Hebrish/Jachnun-MLX`, `Hebrish/Sabich-MLX` | parakeet-mlx (Apple Silicon), with `--fp32` | 10 clips: Jachnun 8/10, Sabich 9/10 |

Jachnun's int8 misses are one word: *tzatzumim* where fp32 wrote *satzumim* the reference is *tza'atzuim*. Sabich's int8 misses were one capital letter and one comma; its MLX miss, one comma. In bf16, parakeet-mlx's default, Jachnun decoded one clip as unknown tokens, hence fp32. The benchmark numbers below are for the `.nemo` in PyTorch, not the quantized exports.

## 7. Evaluation

Two test sets answer different questions.

### Public benchmark

The English-Hebrew code-switching ASR benchmark (dataset `Hebrish/English-Hebrew-Codeswitching-ASR-Benchmark`, test set v1, frozen) has 145 clips. The headline **clean subset** is 84 phone recordings (11.8 min) of sentences in no published training set, with 172 Hebrew term occurrences (139 distinct). A separate **English control** subset of 24 clips is the same speaker reading English only. Jachnun was scored in round 3 (2026-10-19), Sabich in round 4 (2026-10-20, pinned at Hugging Face revision `640c1483`).

- **Term recall**: share of term occurrences whose canonical spelling appears in the transcript as a whole word, case-insensitive, hyphen and space interchangeable. **Unseen-term recall** restricts it to terms absent from the public training split (117 of the 172 occurrences).
- **WER**: lowercased, hyphens to spaces, punctuation removed except in-word apostrophes (`jiwer` 3.1.0).
- **Contamination check**: none of the 145 benchmark clips, by id or by text, is in Jachnun's training manifest (`runs/r6-en30/train.jsonl`, 4,202 rows), Sabich's training or validation data, or the Shakshuka run's.

### Held-out splits

The project's own test split is larger. Two revisions are used, and numbers compare only within one: `550bfcaa0f84` (387 code-switched clips, Section 3), on which stock v2, PoC-3, Jachnun and Shakshuka PoC-2 were scored on 2026-10-19; and the newer `f956dbb184c2` (round R7: 422 code-switched clips, 84 own-English clips of which 57 have no Hebrew, 20 tech-dictation clips, 200 LibriSpeech clips, 17 long-form files), on which stock v2, Jachnun and Sabich were scored on 2026-10-20. WER is normalised.

## 8. Results

### Public benchmark

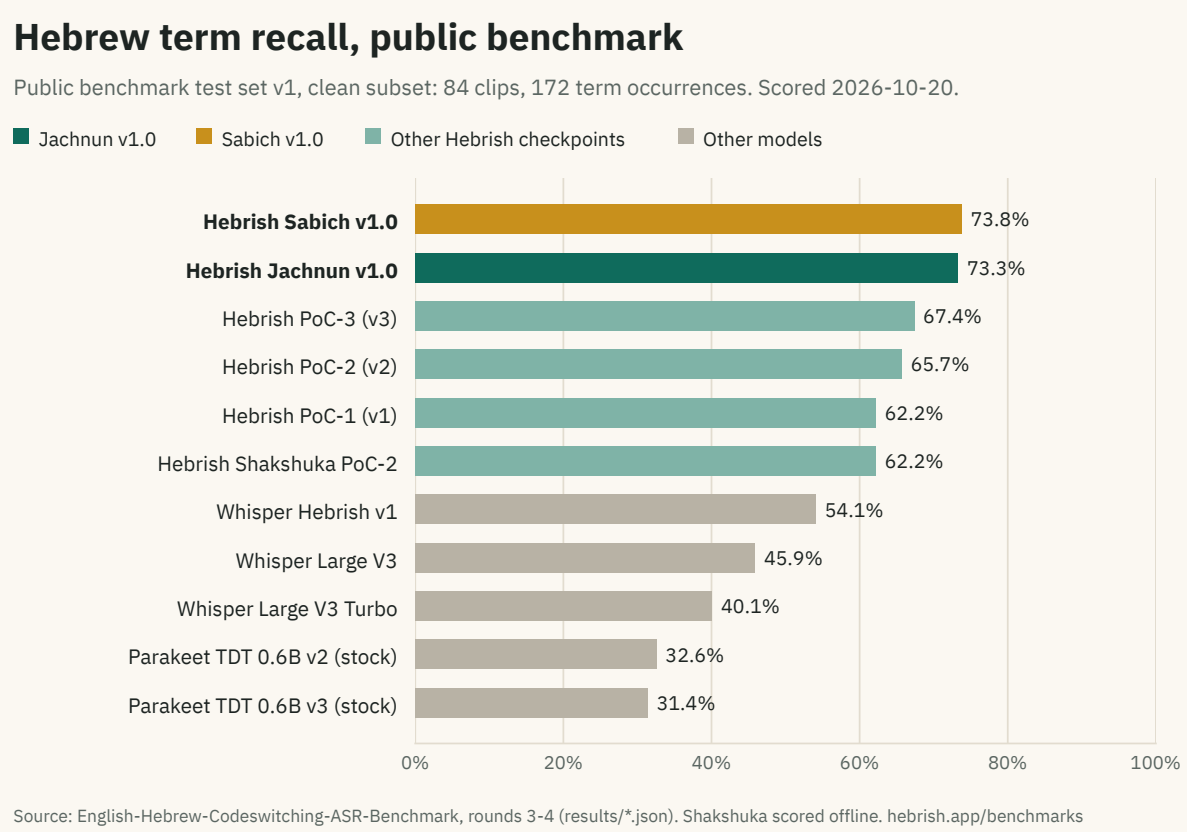



Figure 2. Hebrew term recall on the public benchmark, clean subset, all eleven scored models.

| Model (public benchmark, clean subset) | Term recall | Unseen recall | WER | English control WER |
|---|---|---|---|---|
| **Hebrish Sabich v1.0** | **73.8%** | **68.4%** | **3.27%** | **1.99%** |
| **Hebrish Jachnun v1.0** | **73.3%** | **70.1%** | **3.32%** | **2.17%** |
| Hebrish PoC-3 | 67.4% | 63.2% | 4.0% | 2.5% |
| Hebrish PoC-2 | 65.7% | 60.7% | 4.2% | 2.4% |
| Hebrish PoC-1 | 62.2% | 61.5% | 4.6% | 2.5% |
| Hebrish Shakshuka PoC-2 (offline) | 62.2% | 54.7% | 4.7% | 2.5% |
| Whisper Hebrish v1 | 54.1% | 56.4% | 6.5% | 4.7% |
| Whisper Large V3 | 45.9% | 46.2% | 7.5% | 2.4% |
| Whisper Large V3 Turbo | 40.1% | 39.3% | 8.9% | 2.4% |
| Parakeet TDT 0.6B v2 (stock) | 32.6% | 32.5% | 9.5% | 2.7% |
| Parakeet TDT 0.6B v3 (stock) | 31.4% | 31.6% | 10.5% | 2.7% |

On 172 occurrences, a 2 to 3 point difference in recall is about 4 to 5 hits and within noise. Sabich against Jachnun is one hit of 172 on recall and two of 117 on unseen recall.

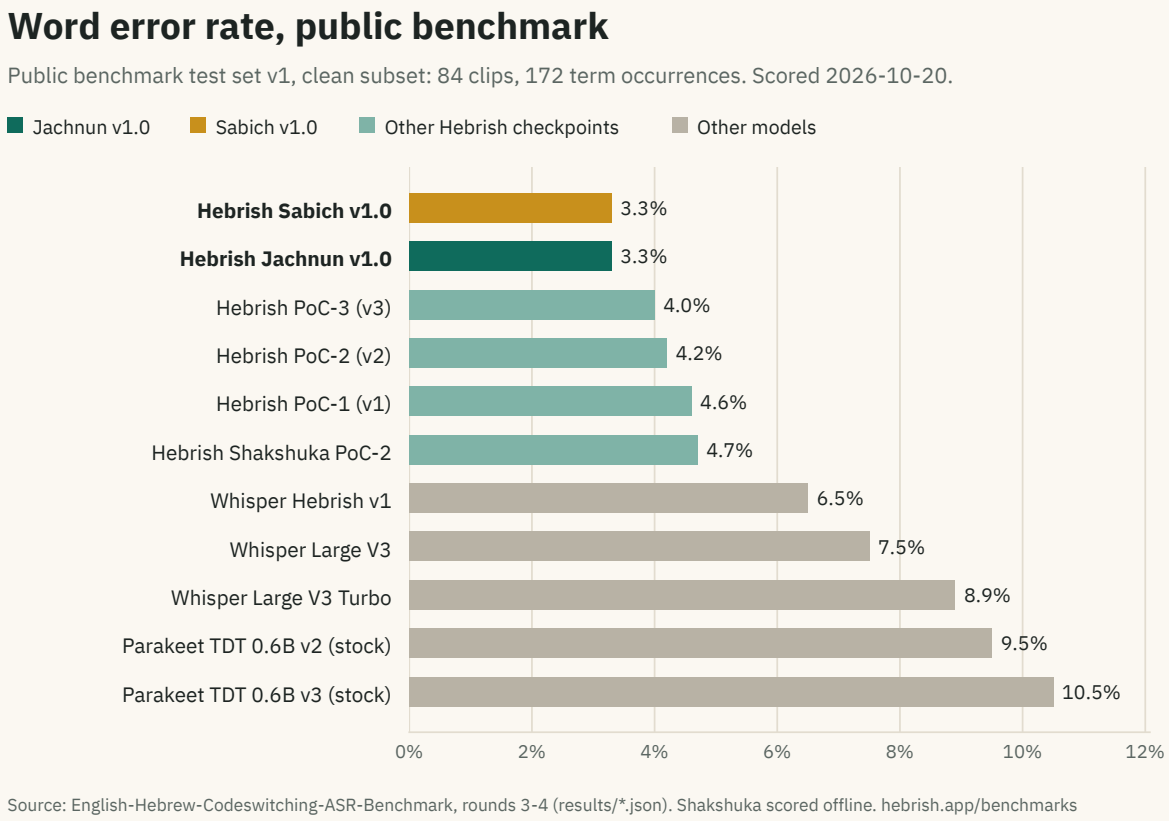



Figure 3. Word error rate on the public benchmark, clean subset, all eleven scored models.

## Jachnun on the held-out split (`550bfcaa`)

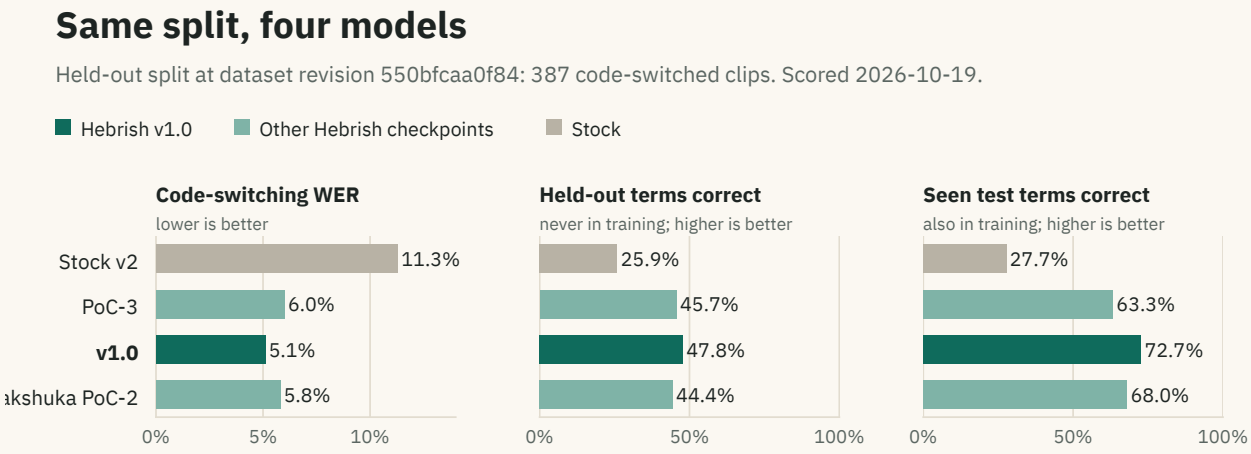



Figure 4. Stock v2, PoC-3, Jachnun v1.0 and Shakshuka PoC-2 on the same held-out split.

| 550bfcaa | CS WER | Held-out terms (293) | Seen terms (278) | Own EN | LibriSpeech | Long-form |
|---|---|---|---|---|---|---|
| Stock Parakeet v2 | 11.3% | 25.9% | 27.7% | 3.1% | 1.2% | 9.6% |
| Hebrish PoC-3 | 6.0% | 45.7% | 63.3% | 2.3% | 1.0% | 5.8% |
| **Jachnun v1.0** | **5.1%** | **47.8%** | **72.7%** | **1.7%** | **1.2%** | **5.0%** |
| Shakshuka PoC-2 (offline) | 5.8% | 44.4% | 68.0% | 3.3% | 2.0% | 5.8% |

## Jachnun and Sabich on the same clips (f956dbb1, round R7)

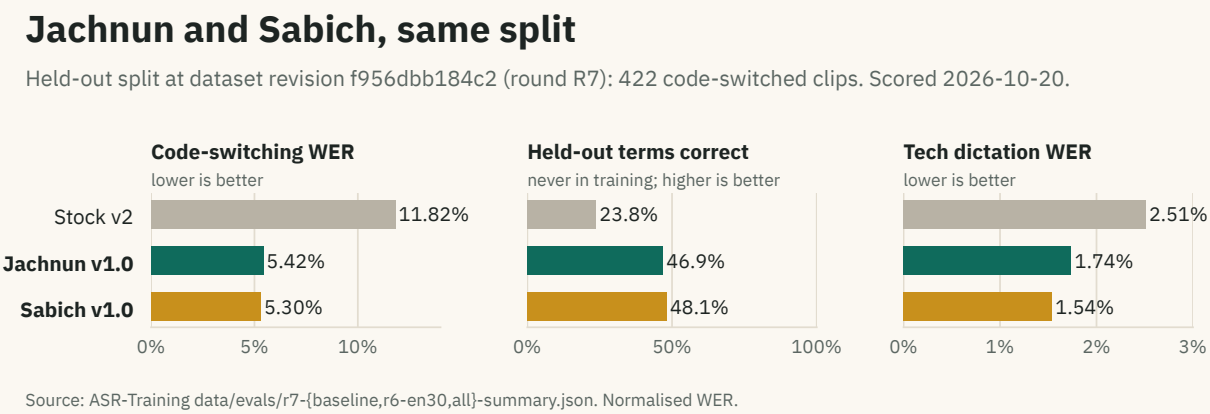



Figure 5. Stock v2, Jachnun v1.0 and Sabich v1.0 on the R7 held-out split. New in this edition.

| f956dbb1 | CS WER | Held-out terms | Own EN | Hebrew-free | Tech | LibriSpeech | Long-form |
|---|---|---|---|---|---|---|---|
| Stock Parakeet v2 | 11.82% | 23.77% | 2.81% | 1.79% | 2.51% | 1.23% | 10.34% |
| **Jachnun v1.0** | **5.42%** | **46.91%** | **1.56%** | **1.23%** | **1.74%** | **1.20%** | **5.84%** |
| **Sabich v1.0** | **5.30%** | **48.15%** | **1.62%** | **1.23%** | **1.54%** | **1.30%** | **5.77%** |

Held-out terms: share of held-out-list term occurrences written correctly (higher is better). Every other column is WER (lower is better). Hebrew-free is the 57-clip subset of own English with no Hebrew words. This revision is newer and larger than 550bfcaa, so Jachnun's numbers differ from the table above.

## English did not get worse

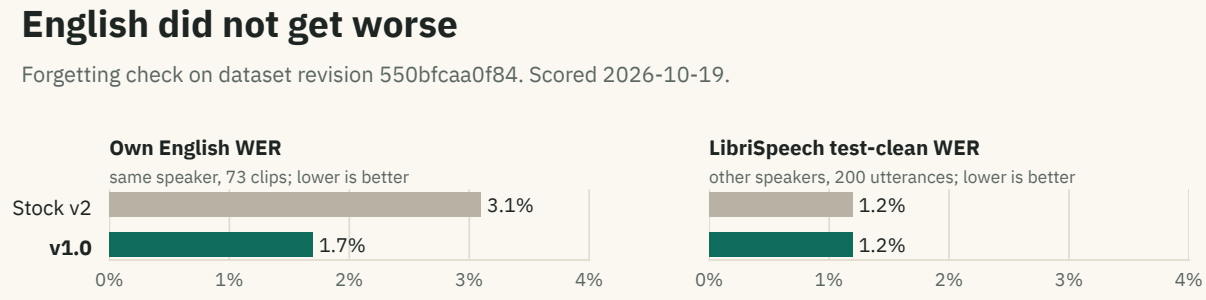



Figure 6. Forgetting check for Jachnun. Own English is the same speaker; LibriSpeech is other speakers.

Jachnun's own-English WER fell from 3.1% to 1.7%, and the public benchmark's English control from 2.7% to 2.2%. On LibriSpeech, other people's English, it scored 1.20% against stock's 1.23%: a noise-level margin, which is the point; it passed the project's forgetting gate. Sabich's own English

(1.62%) and Hebrew-free English (1.23%) stay well below stock (2.81%, 1.79%). On LibriSpeech Sabich scored 1.30% against stock's 1.23%, a strict fail of the gate by about 3 words in 200 clips, which Daniel Rosehill accepted as noise.

### What the results say

- **Hebrew words.** Both models more than double stock Parakeet's term recall on the public benchmark (73.3% and 73.8% against 32.6%) and lead every model scored, including Whisper Large V3 (45.9%) and the earlier Whisper fine-tune (54.1%). Jachnun's unseen-term recall (70.1%) is close to its overall recall, which suggests the gain is not only memorisation of training terms.
- **Code-switched WER** fell from 11.3% (stock) to 5.1% on the `550bfcaa` held-out split, and from 9.5% to 3.3% on the public benchmark.
- **Sabich against Jachnun.** The extra tech and task data cost nothing on code-switching (5.30% against 5.42% WER, 48.15% against 46.91% held-out recall, level on the public benchmark) and gained modestly on tech dictation (1.54% against 1.74%). That is why Sabich became the apps' recommended model on 2026-10-21. The gaps are small and come from one round of tests on one voice.
- **Against PoC-3**, Jachnun is better on every public-benchmark metric. On the held-out split, the gain on test terms seen in training (+9.4 points) is real; the gain on held-out terms (+2.1 points, about 6 of 293 hits) is within noise. More data taught the words it contained; it did not clearly improve guessing words it never heard.

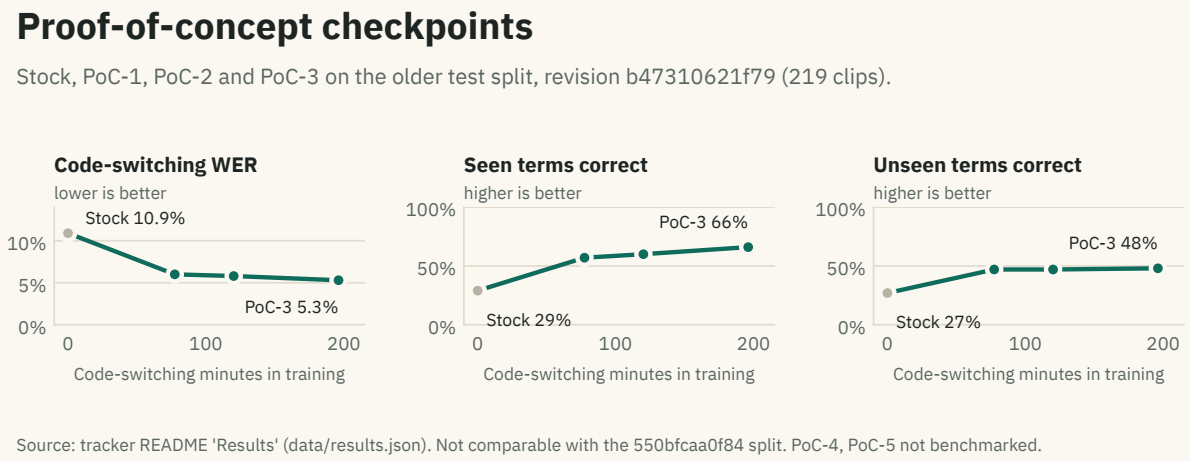



Figure 7. The proof-of-concept checkpoints on the older test split (revision `b47310621f79`, 219 clips). Not comparable with Figures 4 and 5.

## 9. In progress: Shakshuka, the streaming line

> **Not part of release 1.** Shakshuka is under construction. Its checkpoints are proofs of concept, it is not in the Hebrish apps, and its streaming accuracy has not been measured.

Shakshuka applies Jachnun's recordings and recipe to NVIDIA's cache-aware streaming model, `nvidia/nemotron-speech-streaming-en-0.6b`, so that text can appear while the speaker is still talking. Two checkpoints exist, both in `Hebrish/Nemotron-Speech-Streaming-EN-0.6B-English-Hebrew-Codeswitching` (access on request; DOI `10.57967/hf/11107`):

| Checkpoint | Run | Released | Data |
|---|---|---|---|
| Shakshuka PoC-1 | `r5-nemotron-en30` | 2026-10-18 | Trained beside PoC-5; in the repository history |
| Shakshuka PoC-2 | `r6-nemotron-en30` | 2026-10-19 | Jachnun v1.0's data and recipe; best epoch 18, validation WER 6.3% (raw), 45.8 min |

Scored **offline** (full context), Shakshuka PoC-2 trails Jachnun on every metric: 62.2% term recall and 4.7% WER on the public benchmark, 5.8% code-switched WER on the `550bfcaa` held-out split. It wrote none of the 8 number tokens in that test correctly. It has no forgetting verdict, because stock Nemotron was not re-scored on that revision. Next: a first retrain, Shakshuka v1.0, and an evaluation in streaming mode, which is what the line exists for.

## 10. Limitations

- **One speaker's voice.** Every clip, train and test, is the same speaker: Irish-born, English-speaking, living in Jerusalem. Only the LibriSpeech replay and test-clean are other speakers. The results rank models on this speaker and say nothing yet about other accents. More voices are the project's main need.
- **Read speech, mostly one phone.** Prompted sentences read aloud are cleaner than spontaneous dictation.
- **Small test sets.** 84 clean clips and 172 term occurrences on the public benchmark; a 2 to 3 point recall difference is within noise. The held-out splits are larger but have the same speaker.
- **Not every benchmark term is Hebrew.** 22 of the 139 clean-subset terms are English tech labels or product names (*JSON*, *grep*, *Proxmox*, ...), 26 of the 172 occurrences. Excluding them changes no ranks but lowers every model's recall by 2 to 6 points. A relabelled test set will be a new version.
- **A possible held-out leak is being checked.** Two held-out-list terms, *mandatim* and *Beit Safafa*, appear as training target terms at a later dataset revision (`7bfcea1d`). It is not yet confirmed how they were routed at `550bfcaa`. If they were trained on, their held-out hits are not unseen.
- **Held-out gains over PoC-3 are within noise** (Section 8). PoC-4 and PoC-5 were never benchmarked.
- **Quantized exports** were checked on five or ten clips each, not benchmarked.
- **Not evaluated:** a 30-epoch Sabich variant, `r7-all-e30` (validation WER 4.17%), exists but has not been scored on either test set.
- **The canonical spelling is a convention.** A model writing *tsaharon* for *tzaharon* is not wrong in an absolute sense; the benchmark reports lenient recall alongside.

## 11. Compute

| Step | Compute |
|---|---|
| Training run `r6-en30` (Jachnun) | 41.0 min on one L40S (Modal) |
| Training run `r7-all` (Sabich) | 52.1 min on one L40S (Modal) |
| Benchmark round, 2026-10-19 | Six parallel L40S jobs, 19:44–19:46 UTC, about 2.5 min wall each, under 15 GPU-minutes in all |

The binding constraint is recording, not compute. Retraining and benchmarking a model this size takes about an hour of GPU time, so the project retrains on cumulative data whenever enough new audio has been recorded.

## 12. Links

- Website, models and benchmarks: hebrish.app/models · hebrish.app/benchmarks · datasets: hebrish.app/datasets

- Jachnun v1.0: Hebrish/Jachnun · Sabich v1.0: Hebrish/Sabich (access on request); every build at huggingface.co/Hebrish
- Shakshuka: Hebrish/Nemotron-Speech-Streaming-EN-0.6B-English-Hebrew-Codeswitching
- Proof-of-concept record: danielrosehill/Parakeet-TDT-0.6B-v2-English-Hebrew-Codeswitching (PoC-1 to PoC-5)
- Leaderboard: English-Hebrew-Codeswitching-ASR-Leaderboard · benchmark data: `Hebrish/English-Hebrew-Codeswitching-ASR-Benchmark`
- This paper: hebrish.app/whitepaper/hebrish-release-1-whitepaper.pdf · Hebrew: hebrish-release-1-whitepaper-he.pdf

---

Daniel Rosehill. This edition, 2026-10-21, reframes the earlier “Hebrish v1.0” white paper (2026-10-19, with an addendum on Sabich v1.0 of 2026-10-20) as model release 1, covering both models, and adds Section 9 on Shakshuka and Figure 5. No result changed. Text licensed **CC-BY-4.0**. The models inherit CC-BY-4.0 from `nvidia/parakeet-tdt-0.6b-v2`; LibriSpeech-PC, used for replay, is CC-BY-4.0; Shakshuka’s base is under the NVIDIA Open Model License. All results are from one speaker’s voice. Numbers are copied from the project’s result files without rounding beyond their published precision.

WHITE PAPER

# Deterministic English-Hebrew code-switching text-to-speech

Daniel Rosehill · 20/10/2026 · hebrish.app/whitepaper/code-switching-tts-pipeline.pdf

Repository: github.com/Hebrish-ASR/English-Hebrew-Code-Switching-TTS-Pipeline

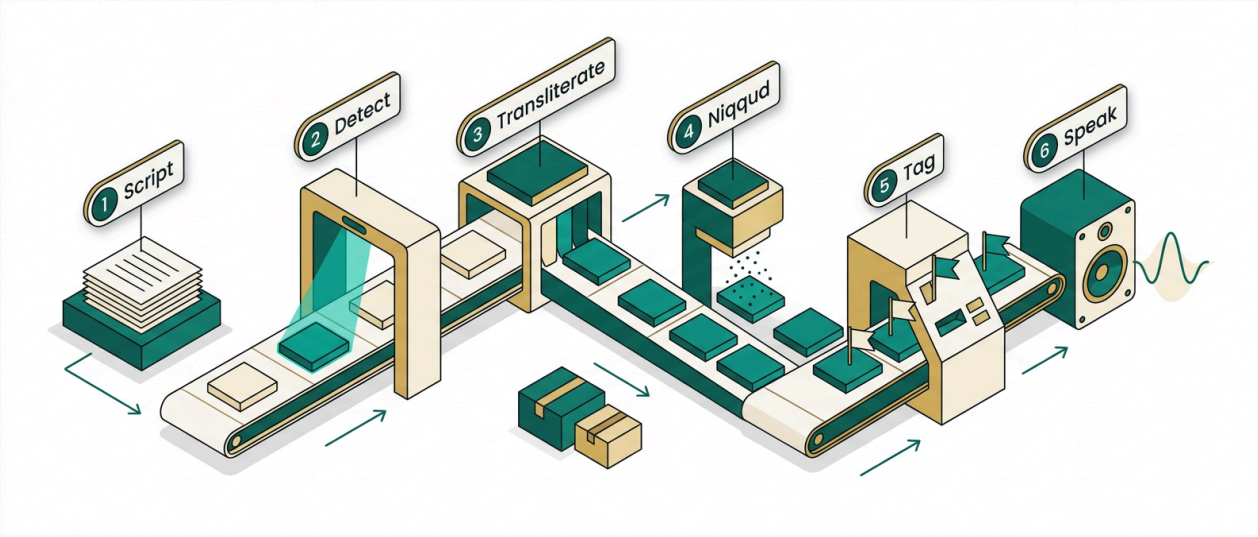



Figure 5: The pipeline as a production line. Illustration generated with an image model; the diagrams below carry the exact text.

**Status.** This is a documented pipeline, not a packaged service. Stages 2 to 5 run end to end on CPU in the repository; the outputs shown here are real. The TTS stage is written against the Chatterbox source and has not been run. Anyone is welcome to package the stages as a container, a Replicate model or a Space, and share it.

## 1. The problem

My Weird Prompts is an AI-generated podcast produced in Jerusalem. An LLM writes each script in English, and Chatterbox TTS voices it. When an episode is about life in Israel, Hebrew terms arrive in Latin letters: *I need to go to Bituach Leumi before the makolet closes.* The script is itself code-switched. An English voice reads *makolet* as “MACK-oh-let”. Words English already knows (*Shabbat*, *kosher*) come out fine; the long tail of everyday Israeli life does not.

## 2. Why standard tools miss it

**The characters give nothing away.** Hebrew letters have their own Unicode block (U+0590 to U+05FF), so Hebrew *script* inside English is trivial to find. Romanized Hebrew is made of the same Basic Latin characters as the English around it; nothing at the character level marks a switch.

**Language identification has no answer.** GlotLID, fastText `lid.176` and `lid218e` have no label for Hebrew in Latin script (`heb_Latn`). Sentence-level language ID would call a 90%-English sentence English anyway. A TTS pipeline needs the character offsets of each Hebrew word: token classification.

**Translating is not a fix.** Many terms have no good English equivalent, and the translation is not what a listener in Israel would say.

## 3. The pipeline




Figure 6: One sentence through the six stages. Hebrew forms are the real stage 4 output.

| # | Stage | Component | Licence | Status |
|---|---|---|---|---|
| 1 | Script | Any LLM writing English with romanized Hebrew | | Input |
| 2 | **Detect** | `Hebrish/Hebrew-Latin-Token-Classifier` (xlm-roberta-base; labels O, B-HE, I-HE) | MIT | Built by Hebrish; run |
| 3 | **Transliterate** | Curated lexicon, then `malper/taatiknet` (ByT5) per word | TaatikNet weights: none stated; data CC-BY-SA 3.0 | Existing; run |
| 4 | **Niqqud** | `dicta-il/dictabert-large-char-menaked`, for unpointed lexicon forms | CC-BY-4.0 | Existing; run |
| 5 | **Tag** | Plain code: one (text, language) segment per piece | | Run |
| 6 | **Speak** | Chatterbox Multilingual, `language_id` per call; or an Azure multilingual voice with SSML `<lang xml:lang="he-IL">` | MIT (Chatterbox) | Written, not run |

### Stage 2: detect

The only Hebrish model in the chain. Span F1 0.941 on a term-disjoint test set (no Hebrew term in the test set appears in training), and no false positives on 58 hard negatives. The labelling test is pronunciation, not etymology: *Shabbat*, *kosher* and *hummus* are not flagged. Use the repository's `predict.py`: the stock transformers pipeline labels sub-tokens and produced stray one-letter spans in our run.

### Stage 3: transliterate

A curated lexicon of expected terms is primary. TaatikNet, the one open model for romanized Hebrew to Hebrew script, is the fallback. It is word-level (trained on about 15K single Wiktionary words), has no sentence context, and returns Hebrew already pointed.

### Stage 4: niqqud

DICTA's nakdan adds vowel points to the unpointed lexicon forms, so the voice does not have to guess how to say them: מַכֹּלֶת → מכולת. Its default output is the standard pointed spelling without vowel letters. It needs `transformers<5`: under 5.x its character tokenizer loads as word-level and the output comes back unpointed, with no error. Phonikud is the alternative when the TTS engine takes phonemes rather than text.

### Stage 5: tag

Split the sentence at every switch:

`en` 'I need to go to ' · `he` בִּטּוּחַ לְאֻמִּי · `en` ' before the ' · `he` מַכֹּלֶת · `en` ' closes.'

### Stage 6: speak

Chatterbox Multilingual supports Hebrew (`"he"`) but takes the language per call, not inline, so each segment is voiced separately with the same voice prompt and the clips are joined with short padding. Two cautions from its source: for Hebrew its tokenizer tries to re-diacritize with `dicta_onnx`, replacing your niqqud, so leave that package uninstalled; and voicing Hebrew with an English reference clip bleeds accent, which `cfg_weight=0` reduces. Among hosted engines, only Azure's multilingual voices accept inline `<lang xml:lang="he-IL">`; Google documents Hebrew in `<lang>` as unsupported and Polly has no Hebrew.

## 4. Worked examples (real output)

| Sentence | Span | Score | Stage 3 | Stages 3-4 output |
|---|---|---|---|---|
| I need to go to Bituach Leumi before the makolet closes. | `Bituach Leumi` | 0.998 | lexicon | בִּטּוּחַ לְאֻמִּי |
| | `makolet` | 0.999 | lexicon | מַכֹּלֶת |
| My savta says the arnona went up again, so I called the iriya. | `savta` | 1.0 | taatiknet | סַוְטָה ✗ (correct: סָבְתָא) |
| | `arnona` | 0.996 | lexicon | אַרְנוֹנָה |
| | `iriya` | 0.998 | taatiknet | אִירִיָה ✗ (correct: עִירִיָה) |
| We spent Shabbat with the mishpacha and the kids ate bamba all afternoon. | `mishpacha` | 0.998 | taatiknet | מִשְׁפָּחָה |
| | `bamba` | 0.993 | taatiknet | בַּמְבָּה |

Both errors come from TaatikNet guessing a spelling without context, which is why the lexicon is primary: adding *savta* and *iriya* to it fixes them. *Shabbat* was correctly left as English.

## 5. Deterministic, or a model that switches by itself

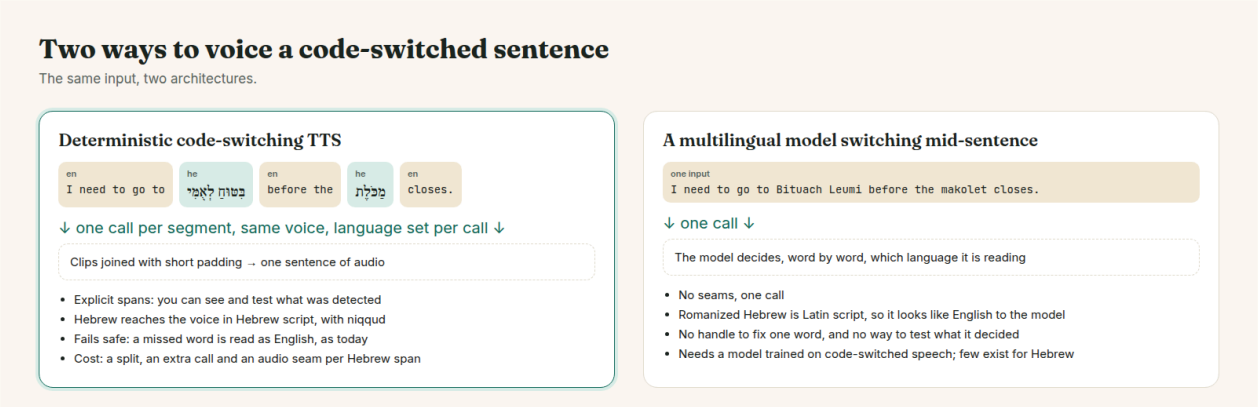



Figure 7: The same sentence, two architectures.

The deterministic pipeline decides every switch in code before the TTS sees the text: what was detected can be read, tested and corrected one word at a time, and a missed word fails safe to today's behaviour. Its cost is a split, an extra TTS call and an audio seam per Hebrew span, so the detector favours precision. The alternative, one multilingual model trusted to switch mid-sentence, gets no script signal from romanized Hebrew and offers no handle on its decisions. A TTS model trained on code-switched English-Hebrew speech could change that; we have not found an open one (searched 2026-10-20, not exhaustively).

## 6. The contract between stages

Each stage reads and extends one JSON object, so any stage can be swapped for another implementation:


```
{
 "text": "I need to go to Bituach Leumi before the makolet closes.",
 "spans": [{"start": 16, "end": 29, "surface": "Bituach Leumi", "score": 0.998, "hebrew": "ביטוח לאומי", "source": "lexicon",
"pointed": "בִּטּוּחַ לְאֻמִּי"}, …],
 "segments": [{"text": "I need to go to ", "lang": "en"},
   {"text": "בִּטּוּחַ לְאֻמִּי", "lang": "he"},
   {"text": " before the ", "lang": "en"},
   {"text": "מַכֹּלֶת", "lang": "he"},
   {"text": " closes.", "lang": "en"}]
}
```


## 7. What is open

- Run stage 6 end to end and listen: how Chatterbox's Hebrew voice reads pointed text with an English reference voice, and how much padding the joins need.
- Package the stages as one service. Anyone is welcome to; tell us and we will link it.
- Replace the demo lexicon with a curated term list.
- The approach carries over to any community that mixes its own words into a majority language in Latin script; only the term list and the stage 3-4 models are Hebrew-specific.

## References

Detector: huggingface.co/Hebrish/Hebrew-Latin-Token-Classifier (original DOI 10.57967/hf/10997) · TaatikNet: huggingface.co/malper/taatiknet, github.com/morrisalp/taatiknet · DICTA nakdan: huggingface.co/dicta-il/dictabert-large-char-menaked · Phonikud: github.com/thewh1teagle/phonikud · Chatterbox: github.com/resemble-ai/chatterbox · Azure SSML: learn.microsoft.com/azure/ai-services/speech-service/speech-synthesis-markup-voice · Google SSML: docs.cloud.google.com/text-to-speech/docs/ssml · Polly languages: docs.aws.amazon.com/polly/latest/dg/supported-languages.html. Code MIT; text and figures CC-BY-4.0.