PROTOVATE / MULTILINGUAL AI GUIDE

Your users don't speak one language at a time.

Why code-switching quietly breaks language detection, search, voice systems and evaluation - and what teams can do about it.

by Prateek Sharma, AI Engineer



One conversation, two languages: code-switch-aware AI preserves the mix, understands the intent, and responds naturally.

Contents

- What code-switching is
- Five things that quietly break
- See it fail, live - all four sample variants
- What actually works
- How to measure it
- About Protovate and the author
- Work with Protovate

What code-switching is

"mera order abhi tak nahi aaya, can you check status please"

That's a real shape of message, and if your product serves India, Pakistan, the Gulf, Singapore, Nigeria, or Hispanic communities in the US, it's what a large share of your users actually write. Two languages in one sentence, usually typed in the Latin alphabet regardless of which language the words come from.

PLAIN TERMS **Code-switching** is mixing two or more languages inside a single conversation - often inside a single sentence. It isn't sloppiness or poor language skill. For hundreds of millions of bilingual people it's simply how they talk.

Here's why it matters commercially: your system probably handles English fine and Hindi fine, and then fails on the mixture, which is what most of your users are actually sending. The failure is quiet. Nothing crashes. The bot just answers slightly wrong, routes to the wrong queue, or replies in formal Hindi to someone who wrote in Latin script, and your support deflection rate sits lower than it should with nobody able to say why.

We've shipped consumer apps serving English, Hindi and Punjabi users, and voice systems where callers mix languages mid-sentence. Everything below comes out of that. The examples use Hinglish because it's what we know best, but every point transfers to Spanglish, Taglish, Arabizi and the rest.

Your product isn't failing at Hindi. It's failing at *the mix* - which is the language most of your users are actually using.

Five things that quietly break

None of these throw an error. That's what makes them expensive.

Failure explorer

Five places code-switching breaks a normal pipeline.

1 / LANGUAGE DETECTION

Language detection returns one answer to a two-answer question

Symptom: the same user gets English replies one message and Hindi replies the next, with no pattern anyone can see.

Almost every pipeline starts by detecting the language, then routes everything downstream off that single label: which prompt, which voice, which translation step, which support queue. Hand a detector a sentence that's half English and half romanized Hindi and it returns one label with low confidence - essentially a coin flip that changes from message to message.

Every downstream decision then inherits that coin flip. The user experiences a bot with a split personality, and because each individual response is plausible, it rarely gets reported as a bug.

WHAT TO DO INSTEAD

- Stop routing on a single language label. Treat mixed as its own first-class case rather than an edge case of one language.
- Detect at word level if you need the signal at all, and keep the mix ratio as metadata rather than a routing switch.
- Let the model see the original message untouched. Translating to English first throws away the information you need to reply naturally.

2 / SPELLING CHAOS

There is no correct spelling, and your keyword rules assume there is

> Symptom: the same intent matches for one user and misses for the next, depending on how they spell a word that has no official spelling.

When Hindi is written in the Latin alphabet there's no standard. "Nahi" appears as `nahi`, `nahin`, `nhi`, `nai`, `nahii`. "Kyun" shows up as `kyu`, `kyon`, `q`. Every keyword rule, every intent list, every blocklist you wrote against one spelling silently fails on the other four.

This matters more than it sounds. Safety filters, routing rules, analytics tagging and search all sit on exact matching somewhere in the stack. A moderation list that only catches one spelling of an abusive term isn't a moderation list.

WHAT TO DO INSTEAD

- Build a variant map for your highest-value terms - product names, intents, complaint words, abuse terms. A few hundred entries covers most traffic.
- Normalise before matching, never after. Fuzzy matching on top of normalisation catches the long tail.
- Mine real user messages for variants rather than guessing. Your logs already contain the spellings your users use.

3 / SEARCH AND RETRIEVAL

The question is in one language, the documents are in another

> Symptom: search works in testing, where staff type English, and returns nothing useful for real users.

If your help articles, policies and product data are in English and your users ask in mixed script, the retrieval step quietly returns weak matches. The model then answers confidently from whatever it got, which is worse than returning nothing, because nobody can tell from the output that the search failed.

It's an especially nasty failure because the system looks like it's working. The answers are fluent. They're just based on the wrong three documents.

WHAT TO DO INSTEAD

- Search twice - with the message as written and with an English rendering - then merge the results.
- Combine keyword and semantic search. Keyword search catches product names and codes that embeddings blur.
- Test retrieval separately from generation. If you only measure the final answer, you'll never see that the search is the broken part.

4 / VOICE

Speech systems are usually locked to one language at a time

> Symptom: the transcript is fluent nonsense - every word is a real word, but the sentence isn't what the caller said.

Set a speech recogniser to Hindi and it will try to hear English words as Hindi ones. Set it to English and the reverse happens. Because the output is always grammatical, it reads as a confident, complete transcript and nothing downstream can tell it's wrong.

Numbers, names and product terms suffer most - exactly the parts you need to act on. An order number heard wrong isn't a degraded experience, it's a wrong action taken on the wrong record.

WHAT TO DO INSTEAD

- Use a multilingual recognition mode rather than pinning one language per call.
- Supply a vocabulary bias list of your product names, cities, and domain terms. This is the highest-return hour of work in the whole pipeline.

- Read numbers back for confirmation before acting on them, every time.
- For speech output, match the script the user used. Pure formal Hindi read aloud to someone speaking Hinglish sounds like a government announcement.

5 / NOBODY IS MEASURING

There's no benchmark, so most teams don't measure at all

> Symptom: "it works fine" - tested entirely in English by people who speak English at work.

There are standard evaluation sets for English. There aren't for Hinglish or Punjabi-English, so most teams skip measurement entirely and rely on spot-checks by whoever is around. If nobody on the team writes the way the users write, the gap never surfaces.

On one consumer app we found a change that clearly improved English answers made another language measurably worse. We only caught it because the test set covered all three. Without that, it ships, and the complaints arrive as churn rather than bug reports.

WHAT TO DO INSTEAD

- Build a test set from real user messages in each mix you serve. Two hundred per language pair is enough to start.
- Score each language separately. An aggregate number hides exactly the regression you need to see.
- Include heavily-mixed messages deliberately, not just clean ones from each language.

See it fail, live

Type a mixed message below, or pick one of the samples. The left column is what a standard English-first pipeline does with it. The right column is the same message through a code-switch-aware one. Everything runs in your browser.

The point isn't the toy intent matcher. It's that the left column never errors. It produces a fluent, confident, wrong response - and in production that lands in your deflection metrics as a handled conversation.

All four sample variants

SAMPLE 1

mera order abhi tak nahi aaya, can you check status please

ENGLISH-FIRST PIPELINE	
DETECTED	English (55% confidence)
UNDERSTOOD	55% of words
INTENT	order_status (low confidence)
REPLY	Routed to order_status on the English words alone, then answered in formal English and ignored that half the message was not English.

CODE-SWITCH AWARE	
MIX	55% EN / 45% HI
TAGGED	mera HI · order EN · abhi HI · tak HI · nahi HI · aaya HI · can EN · you EN · check EN · status EN · please EN
NORMALISED	mera order abhi tak nahi aaya can you check status please (no changes needed)
INTENT	order_status
REPLY	Aapka order abhi transit mein hai - expected delivery 2 din mein. Tracking link bhej diya hai.

SAMPLE 2

paise kab wapas milenge? refund status batao

ENGLISH-FIRST PIPELINE	
DETECTED	Hindi (57% confidence)
UNDERSTOOD	29% of words
INTENT	no match - fallback
REPLY	A formal Devanagari fallback asks the user to try again, even though the user wrote in Latin script.

CODE-SWITCH AWARE	
MIX	29% EN / 57% HI / 14% unknown
TAGGED	paise HI · kab HI · wapas HI · milenge ? · refund EN · status EN · batao HI
NORMALISED	paisa kab wapas milenge refund status batao (1 fixed)
INTENT	refund
REPLY	Refund process ho gaya hai - paise 3-5 working days mein wapas aa jayenge.

SAMPLE 3

mainu app vich login nahi ho raha, password reset kiddan karna hai

ENGLISH-FIRST PIPELINE	
DETECTED	Hindi (58% confidence)
UNDERSTOOD	33% of words
INTENT	no match - fallback
REPLY	A formal Devanagari fallback asks the user to rephrase instead of resolving the login problem.

CODE-SWITCH AWARE	
MIX	33% EN / 58% HI / 8% unknown
TAGGED	mainu HI · app EN · vich ? · login EN · nahi HI · ho HI · raha HI · password EN · reset EN · kiddan HI · karna HI · hai HI
NORMALISED	mainu app vich login nahi ho raha password reset kiddan karna hai (no changes needed)
INTENT	login_issue
REPLY	Password reset link bhej diya hai registered number par. 10 min mein expire ho jayega.

SAMPLE 4

delivery address change karna hai, kitna time lagega

ENGLISH-FIRST PIPELINE	
DETECTED	English (50% confidence)
UNDERSTOOD	50% of words
INTENT	address_change (low confidence)
REPLY	Routed using only the English terms, then answered in formal English without matching the user's language mix.

CODE-SWITCH AWARE	
MIX	50% EN / 38% HI / 13% unknown
TAGGED	delivery EN · address EN · change EN · karna HI · hai HI · kitna HI · time EN · lagega ?
NORMALISED	delivery address change karna hai kitna time lagega (no changes needed)
INTENT	address_change
REPLY	Address update kar diya hai. Nayi delivery date 2 din aage ho jayegi.

What actually works

Roughly in order of return on effort.

- **Reply in the script they wrote in.** If the user typed Latin letters, answer in Latin letters. Switching them into Devanagari because the words are Hindi is the single most common own-goal, and it reads as cold and official to someone who was being casual.
- **Mirror the mix, don't correct it.** Answering formal Hindi to a Hinglish message is technically right and socially wrong. Instruct the model explicitly to match the user's register and mixture.
- **Normalise spellings before any exact matching.** A few hundred variant mappings covers most traffic and immediately fixes intent routing, analytics and moderation at the same time.
- **Give the model the raw message.** Pre-translating to English destroys exactly the signal you need and introduces a second place to go wrong.
- **Bias your speech vocabulary.** Product names, city names, and domain terms supplied to the recogniser fix most voice transcription pain in an afternoon.
- **Watch your script costs.** The same sentence costs very different amounts depending on the script it's written in, and that difference flows straight into your bill and your context limits.

The cost nobody budgets for

Text in non-Latin scripts is typically split into far more billable pieces than the same meaning in English. That has two consequences: a feature that's affordable in English can be noticeably more expensive in Hindi or Punjabi, and long conversations hit context limits sooner.

SAME SENTENCE, WRITTEN AS	RELATIVE COST	PRACTICAL EFFECT
English	1.0×	Baseline
Romanised Hinglish	~1.3×	Cheap, but harder for models to read accurately
Devanagari / Gurmukhi	~2.5×	Easier for models, markedly more expensive per message

Illustrative ratios - they vary by model and by sentence. Measure yours; the gap is always real and usually larger than teams expect.

Normalisation, concretely

This is the piece teams most often skip and most quickly benefit from. It's not clever and it doesn't need a model.

NORMALISE.PY - THE UNGLAMOROUS FIX

```
# Mine these from your own logs. Guessing produces a map for a dialect
# nobody on your team actually speaks.
VARIANTS = {
    "nahi":  ["nahin", "nhi", "nai", "nahii", "nhii"],
    "kyun":  ["kyu", "kyon", "q", "kiu"],
    "paisa": ["paise", "pese", "paisaa"],
    "wapas": ["vapis", "wapis", "vapas"],
}
LOOKUP = {v: base for base, vs in VARIANTS.items() for v in [base] + vs}

def normalise(text):
    # Normalise for MATCHING only. Keep the original for the model,
    # and for the reply, so the user is answered the way they wrote.
    return " ".join(LOOKUP.get(w, w) for w in text.lower().split())

# Then: intent rules, moderation lists and analytics all match against
# normalise(text) - never against the raw string.
```

THE RULE Normalise for matching. Preserve for answering. The moment you let the normalised version reach the user, you've started correcting how they speak.

How to measure it

You can't buy a benchmark for this, which is precisely why it's a competitive advantage to have one. Building it takes about two days.

- **Collect 200 real messages per language pair.** From support logs, app reviews, WhatsApp threads. Real spellings, real mixing, real typos.
- **Label the correct outcome** for each: the right intent, the right answer, and the script the reply should be in.
- **Score three things separately** - did it understand, did it answer correctly, and did it reply in the right script and register.
- **Report per language, never aggregated.** One combined number is how a regression in your second-biggest market stays invisible for a quarter.
- **Bucket by mix ratio.** Mostly-English, balanced, and mostly-Hindi messages fail differently, and the balanced bucket is usually the worst.
- **Have a native speaker review the reply tone.** Technically-correct answers can still read as condescending or bureaucratic, and only a speaker will catch it.

That last one deserves emphasis. Most of what makes a multilingual product feel good isn't accuracy, it's register - whether the thing sounds like a person from the user's world or a form letter. No automated score measures that. Twenty minutes with a native speaker will tell you more than a week of metrics.

Every competitor serving these users has the same gap. The one that closes it doesn't win on technology - it wins because it's the only one that *sounds normal.*

About Protovate

Protovate builds practical AI-powered software for complex, real-world environments. Led by Brian Pollack and a global team with more than 30 years of experience, Protovate helps organizations innovate responsibly, improve efficiency, and turn emerging technology into solutions that deliver measurable impact.

Over the decades, the Protovate team has worked with organizations including NASA, Johnson & Johnson, Microsoft, Walmart, Covidien, Singtel, LG, Yahoo, and Lowe's.

Originally published on Protovate.AI.

PS

Prateek Sharma

AI engineer at Protovate

Prateek Sharma is an AI engineer at Protovate with over a decade of experience building and integrating intelligent applications across AI, mobile, and full-stack development. He brings a hands-on, practical approach to turning advanced technology into reliable, production-ready systems.

WORK WITH US

Building for users who mix languages?

Protovate builds multilingual assistants, voice systems, document extraction and native mobile products for enterprise and consumer clients.

Talk with our team